RNAlib-2.2.8

Generated by Doxygen 1.8.11

Contents

1 ViennaRNA Package core - RNAlib **1**

1.1 Introduction . . . 2

2 Parsing and Comparing - Functions to Manipulate Structures **3**

3 Utilities - Odds and Ends **7**

3.1 Producing secondary structure graphs . . . 7

3.2 Producing (colored) dot plots for base pair probabilities . . . 8

3.3 Producing (colored) alignments . . . 9

3.4 RNA sequence related utilities . . . 9

3.5 RNA secondary structure related utilities . . . 9

3.6 Miscellaneous Utilities . . . 10

4 RNAlib API v3.0 **13**

4.1 Introduction . . . 13

4.2 What are the major changes? . . . 13

4.3 How to port your program to the new API . . . 13

4.4 Some Examples using RNAlib API v3.0 . . . 14

5 Scripting Language interface(s) **15**

5.1 Introduction . . . 15

5.1.1 Function renaming scheme . . . 15

5.1.2 Object oriented Interface for data structures . . . 15

5.2 Examples . . . 16

5.2.1 Perl Examples . . . 16

5.2.1.1 Using the Flat Interface . . . 16

5.2.1.2 Using the Object Oriented (OO) Interface . . . 16

5.2.2 Python Examples . . . 16

5.3 How functions, structures, enums, and macro definitions are wrapped . . . 16

6 Input / Output File Formats 19

6.1 File formats for Secondary Structure Constraints . . . 19

6.1.1 Constraints Definition File . . . 19

6.1.1.1 Constraint commands . . . 19

6.1.1.2 Specification of the loop type context . . . 19

6.1.1.3 Controlling the orientation of base pairing . . . 20

6.1.1.4 Sequence coordinates . . . 20

6.1.1.5 Valid constraint commands . . . 20

7 Example - A Small Example Program 23

8 Deprecated List 25

9 Bug List 35

10 Module Index 37

10.1 Modules . . . 37

11 Data Structure Index 39

11.1 Data Structures . . . 39

12 File Index 41

12.1 File List . . . 41

13 Module Documentation 45

13.1 RNA Secondary Structure Prediction . . . 45

13.1.1 Detailed Description . . . 46

13.2 Inverse Secondary Structure Prediction . . . 47

13.2.1 Detailed Description . . . 47

13.2.2 Function Documentation . . . 47

13.2.2.1 inverse_fold(char *start, const char *target) . . . 47

13.2.2.2 inverse_pf_fold(char *start, const char *target) . . . 48

13.2.3 Variable Documentation . . . 48

13.2.3.1 final_cost . . . 48

13.2.3.2 give_up . . . 48
13.2.3.3 inv_verbose . . . 48
13.3 Refolding paths between secondary structues . . . 49
13.3.1 Detailed Description . . . 49
13.4 Free Energy Evaluation for given Sequence / Structure Pairs . . . 50
13.4.1 Detailed Description . . . 50
13.4.2 Function Documentation . . . 50
13.4.2.1 vrna_eval_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . 50
13.5 Processing and Evaluating Decomposed Loops . . . 51
13.5.1 Detailed Description . . . 52
13.5.2 Function Documentation . . . 52
13.5.2.1 E_ExtLoop(int type, int si1, int sj1, vrna_param_t *P) . . . 52
13.5.2.2 exp_E_ExtLoop(int type, int si1, int sj1, vrna_exp_param_t *P) . . . 52
13.5.2.3 E_Stem(int type, int si1, int sj1, int extLoop, vrna_param_t *P) . . . 53
13.5.2.4 exp_E_Stem(int type, int si1, int sj1, int extLoop, vrna_exp_param_t *P) . . . 54
13.5.2.5 get_gquad_matrix(short *S, vrna_param_t *P) . . . 54
13.5.2.6 parse_gquad(const char *struc, int *L, int l[3]) . . . 55
13.5.2.7 backtrack_GQuad_IntLoop(int c, int i, int j, int type, short *S, int *ggg, int *index, int *p, int *q, vrna_param_t *P) . . . 55
13.5.2.8 backtrack_GQuad_IntLoop_L(int c, int i, int j, int type, short *S, int **ggg, int maxdist, int *p, int *q, vrna_param_t *P) . . . 55
13.5.2.9 E_Hairpin(int size, int type, int si1, int sj1, const char *string, vrna_param_t *P) . . . 56
13.5.2.10 exp_E_Hairpin(int u, int type, short si1, short sj1, const char *string, vrna_exp_param_t *P) . . . 57
13.5.2.11 vrna_E_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . 57
13.5.2.12 vrna_E_ext_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . 58
13.5.2.13 vrna_exp_E_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . 58
13.5.2.14 vrna_BT_hp_loop(vrna_fold_compound_t *vc, int i, int j, int en, vrna_bp_stack_t *bp_stack, int *stack_count) . . . 58
13.5.2.15 E_IntLoop(int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t *P) . . . 59
13.5.2.16 exp_E_IntLoop(int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, vrna_exp_param_t *P) . . . 60

13.5.2.17 E_mb_loop_stack(int i, int j, vrna_fold_compound_t *vc) . . . 60
13.5.2.18 vrna_BT_mb_loop(vrna_fold_compound_t *vc, int *i, int *j, int *k, int en, int *component1, int *component2) . . . 61
13.6 Energy Parameter Sets and Boltzmann Factors . . . 62
13.6.1 Detailed Description . . . 63
13.6.2 Data Structure Documentation . . . 64
13.6.2.1 struct vrna_param_s . . . 64
13.6.2.2 struct vrna_exp_param_s . . . 64
13.6.3 Typedef Documentation . . . 65
13.6.3.1 paramT . . . 65
13.6.3.2 pf_paramT . . . 65
13.6.4 Function Documentation . . . 65
13.6.4.1 vrna_params(vrna_md_t *md) . . . 65
13.6.4.2 vrna_params_copy(vrna_param_t *par) . . . 66
13.6.4.3 vrna_exp_params(vrna_md_t *md) . . . 66
13.6.4.4 vrna_exp_params_comparative(unsigned int n_seq, vrna_md_t *md) . . . 67
13.6.4.5 vrna_exp_params_copy(vrna_exp_param_t *par) . . . 67
13.6.4.6 vrna_params_subst(vrna_fold_compound_t *vc, vrna_param_t *par) . . . 68
13.6.4.7 vrna_exp_params_subst(vrna_fold_compound_t *vc, vrna_exp_param_t *params) 68
13.6.4.8 vrna_exp_params_rescale(vrna_fold_compound_t *vc, double *mfe) . . . 69
13.6.4.9 vrna_params_reset(vrna_fold_compound_t *vc, vrna_md_t *md_p) . . . 70
13.6.4.10 vrna_exp_params_reset(vrna_fold_compound_t *vc, vrna_md_t *md_p) . . . 70
13.6.4.11 get_scaled_pf_parameters(void) . . . 71
13.6.4.12 get_boltzmann_factors(double temperature, double betaScale, vrna_md_t md, double pf_scale) . . . 71
13.6.4.13 get_boltzmann_factor_copy(vrna_exp_param_t *parameters) . . . 72
13.6.4.14 get_scaled_alipf_parameters(unsigned int n_seq) . . . 72
13.6.4.15 get_boltzmann_factors_ali(unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale) . . . 72
13.6.4.16 scale_parameters(void) . . . 73
13.6.4.17 get_scaled_parameters(double temperature, vrna_md_t md) . . . 73
13.7 Manipulation of the Prediction Models . . . 74

13.7.1 Detailed Description . . . 78
13.7.2 Data Structure Documentation . . . 78
13.7.2.1 struct vrna_md_s . . . 78
13.7.3 Macro Definition Documentation . . . 81
13.7.3.1 VRNA_MODEL_DEFAULT_TEMPERATURE . . . 81
13.7.3.2 VRNA_MODEL_DEFAULT_PF_SCALE . . . 81
13.7.3.3 VRNA_MODEL_DEFAULT_BETA_SCALE . . . 82
13.7.3.4 VRNA_MODEL_DEFAULT_DANGLES . . . 82
13.7.3.5 VRNA_MODEL_DEFAULT_SPECIAL_HP . . . 82
13.7.3.6 VRNA_MODEL_DEFAULT_NO_LP . . . 82
13.7.3.7 VRNA_MODEL_DEFAULT_NO_GU . . . 82
13.7.3.8 VRNA_MODEL_DEFAULT_NO_GU_CLOSURE . . . 83
13.7.3.9 VRNA_MODEL_DEFAULT_CIRC . . . 83
13.7.3.10 VRNA_MODEL_DEFAULT_GQUAD . . . 83
13.7.3.11 VRNA_MODEL_DEFAULT_UNIQ_ML . . . 83
13.7.3.12 VRNA_MODEL_DEFAULT_ENERGY_SET . . . 83
13.7.3.13 VRNA_MODEL_DEFAULT_BACKTRACK . . . 84
13.7.3.14 VRNA_MODEL_DEFAULT_BACKTRACK_TYPE . . . 84
13.7.3.15 VRNA_MODEL_DEFAULT_COMPUTE_BPP . . . 84
13.7.3.16 VRNA_MODEL_DEFAULT_MAX_BP_SPAN . . . 84
13.7.3.17 VRNA_MODEL_DEFAULT_WINDOW_SIZE . . . 84
13.7.3.18 VRNA_MODEL_DEFAULT_LOG_ML . . . 85
13.7.3.19 VRNA_MODEL_DEFAULT_ALI_OLD_EN . . . 85
13.7.3.20 VRNA_MODEL_DEFAULT_ALI_RIBO . . . 85
13.7.3.21 VRNA_MODEL_DEFAULT_ALI_CV_FACT . . . 85
13.7.3.22 VRNA_MODEL_DEFAULT_ALI_NC_FACT . . . 85
13.7.4 Function Documentation . . . 85
13.7.4.1 vrna_md_set_default(vrna_md_t *md) . . . 85
13.7.4.2 vrna_md_update(vrna_md_t *md) . . . 86
13.7.4.3 vrna_md_option_string(vrna_md_t *md) . . . 86

13.7.4.4 vrna_md_defaults_reset(vrna_md_t *md_p) . . . 86
13.7.4.5 vrna_md_defaults_temperature(double T) . . . 87
13.7.4.6 vrna_md_defaults_temperature_get(void) . . . 87
13.7.4.7 vrna_md_defaults_betaScale(double b) . . . 87
13.7.4.8 vrna_md_defaults_betaScale_get(void) . . . 88
13.7.4.9 vrna_md_defaults_dangles(int d) . . . 88
13.7.4.10 vrna_md_defaults_dangles_get(void) . . . 88
13.7.4.11 vrna_md_defaults_special_hp(int flag) . . . 89
13.7.4.12 vrna_md_defaults_special_hp_get(void) . . . 89
13.7.4.13 vrna_md_defaults_noLP(int flag) . . . 89
13.7.4.14 vrna_md_defaults_noLP_get(void) . . . 90
13.7.4.15 vrna_md_defaults_noGU(int flag) . . . 90
13.7.4.16 vrna_md_defaults_noGU_get(void) . . . 90
13.7.4.17 vrna_md_defaults_noGUclosure(int flag) . . . 91
13.7.4.18 vrna_md_defaults_noGUclosure_get(void) . . . 91
13.7.4.19 vrna_md_defaults_logML(int flag) . . . 91
13.7.4.20 vrna_md_defaults_logML_get(void) . . . 92
13.7.4.21 vrna_md_defaults_circ(int flag) . . . 92
13.7.4.22 vrna_md_defaults_circ_get(void) . . . 92
13.7.4.23 vrna_md_defaults_gquad(int flag) . . . 93
13.7.4.24 vrna_md_defaults_gquad_get(void) . . . 94
13.7.4.25 vrna_md_defaults_uniq_ML(int flag) . . . 94
13.7.4.26 vrna_md_defaults_uniq_ML_get(void) . . . 94
13.7.4.27 vrna_md_defaults_energy_set(int e) . . . 95
13.7.4.28 vrna_md_defaults_energy_set_get(void) . . . 95
13.7.4.29 vrna_md_defaults_backtrack(int flag) . . . 95
13.7.4.30 vrna_md_defaults_backtrack_get(void) . . . 96
13.7.4.31 vrna_md_defaults_backtrack_type(char t) . . . 96
13.7.4.32 vrna_md_defaults_backtrack_type_get(void) . . . 96
13.7.4.33 vrna_md_defaults_compute_bpp(int flag) . . . 97

13.7.4.34 vrna_md_defaults_compute_bpp_get(void) . . . 97
13.7.4.35 vrna_md_defaults_max_bp_span(int span) . . . 97
13.7.4.36 vrna_md_defaults_max_bp_span_get(void) . . . 98
13.7.4.37 vrna_md_defaults_min_loop_size(int size) . . . 98
13.7.4.38 vrna_md_defaults_min_loop_size_get(void) . . . 98
13.7.4.39 vrna_md_defaults_window_size(int size) . . . 98
13.7.4.40 vrna_md_defaults_window_size_get(void) . . . 99
13.7.4.41 vrna_md_defaults_oldAliEn(int flag) . . . 99
13.7.4.42 vrna_md_defaults_oldAliEn_get(void) . . . 99
13.7.4.43 vrna_md_defaults_ribo(int flag) . . . 100
13.7.4.44 vrna_md_defaults_ribo_get(void) . . . 100
13.7.4.45 vrna_md_defaults_cv_fact(double factor) . . . 100
13.7.4.46 vrna_md_defaults_cv_fact_get(void) . . . 101
13.7.4.47 vrna_md_defaults_nc_fact(double factor) . . . 101
13.7.4.48 vrna_md_defaults_nc_fact_get(void) . . . 101
13.7.4.49 vrna_md_defaults_sfact(double factor) . . . 102
13.7.4.50 vrna_md_defaults_sfact_get(void) . . . 102
13.7.4.51 set_model_details(vrna_md_t *md) . . . 102
13.7.5 Variable Documentation . . . 102
13.7.5.1 temperature . . . 103
13.7.5.2 pf_scale . . . 103
13.7.5.3 dangles . . . 103
13.7.5.4 tetra_loop . . . 103
13.7.5.5 noLonelyPairs . . . 104
13.7.5.6 canonicalBPonly . . . 104
13.7.5.7 energy_set . . . 104
13.7.5.8 do_backtrack . . . 104
13.7.5.9 backtrack_type . . . 104
13.7.5.10 nonstandards . . . 104
13.7.5.11 max_bp_span . . . 104

13.8 Constraining the Secondary Structure Predictions and Evaluations . . . 105
13.8.1 Detailed Description . . . 107
13.8.2 Macro Definition Documentation . . . 109
13.8.2.1 VRNA_CONSTRAINT_FILE . . . 109
13.8.2.2 VRNA_CONSTRAINT_SOFT_MFE . . . 109
13.8.2.3 VRNA_CONSTRAINT_SOFT_PF . . . 109
13.8.2.4 VRNA_DECOMP_PAIR_HP . . . 109
13.8.2.5 VRNA_DECOMP_PAIR_IL . . . 110
13.8.2.6 VRNA_DECOMP_PAIR_ML . . . 111
13.8.2.7 VRNA_DECOMP_ML_ML_ML . . . 111
13.8.2.8 VRNA_DECOMP_ML_STEM . . . 112
13.8.2.9 VRNA_DECOMP_ML_ML . . . 112
13.8.2.10 VRNA_DECOMP_ML_UP . . . 113
13.8.2.11 VRNA_DECOMP_ML_ML_STEM . . . 113
13.8.2.12 VRNA_DECOMP_ML_COAXIAL . . . 114
13.8.2.13 VRNA_DECOMP_EXT_EXT . . . 114
13.8.2.14 VRNA_DECOMP_EXT_UP . . . 114
13.8.2.15 VRNA_DECOMP_EXT_STEM . . . 115
13.8.2.16 VRNA_DECOMP_EXT_EXT_EXT . . . 115
13.8.2.17 VRNA_DECOMP_EXT_STEM_EXT . . . 115
13.8.2.18 VRNA_DECOMP_EXT_EXT_STEM . . . 116
13.8.2.19 VRNA_DECOMP_EXT_EXT_STEM1 . . . 116
13.8.3 Function Documentation . . . 116
13.8.3.1 vrna_constraints_add(vrna_fold_compound_t *vc, const char *constraint, unsigned int options) . . . 116
13.8.3.2 vrna_message_constraint_options(unsigned int option) . . . 117
13.8.3.3 vrna_message_constraint_options_all(void) . . . 118
13.9 Data Structures and Preprocessor Macros . . . 119
13.9.1 Detailed Description . . . 120
13.9.2 Data Structure Documentation . . . 121
13.9.2.1 struct vrna_basepair_s . . . 121

13.9.2.2 struct vrna_plist_s . . . 121
13.9.2.3 struct vrna_cpair_s . . . 121
13.9.2.4 struct vrna_sect_s . . . 121
13.9.2.5 struct vrna_bp_stack_s . . . 121
13.9.2.6 struct pu_contrib . . . 121
13.9.2.7 struct interact . . . 121
13.9.2.8 struct pu_out . . . 122
13.9.2.9 struct constrain . . . 122
13.9.2.10 struct duplexT . . . 122
13.9.2.11 struct node . . . 123
13.9.2.12 struct snoopT . . . 123
13.9.2.13 struct dupVar . . . 123
13.9.3 Typedef Documentation . . . 123
13.9.3.1 PAIR . . . 123
13.9.3.2 plist . . . 123
13.9.3.3 cpair . . . 124
13.9.3.4 sect . . . 124
13.9.3.5 bondT . . . 124
13.10 Utilities . . . 125
13.10.1 Detailed Description . . . 127
13.10.2 Macro Definition Documentation . . . 127
13.10.2.1 VRNA_INPUT_FASTA_HEADER . . . 127
13.10.2.2 VRNA_INPUT_CONSTRAINT . . . 127
13.10.3 Function Documentation . . . 127
13.10.3.1 vrna_alloc(unsigned size) . . . 127
13.10.3.2 vrna_realloc(void *p, unsigned size) . . . 128
13.10.3.3 vrna_message_error(const char message[]) . . . 128
13.10.3.4 vrna_message_warning(const char message[]) . . . 128
13.10.3.5 vrna_urn(void) . . . 129
13.10.3.6 vrna_int_urn(int from, int to) . . . 129

13.10.3.7 vrna_time_stamp(void) . . . 129
13.10.3.8 get_line(FILE *fp) . . . 130
13.10.3.9 get_input_line(char **string, unsigned int options) . . . 130
13.10.3.10 vrna_message_input_seq_simple(void) . . . 131
13.10.3.11 vrna_message_input_seq(const char *s) . . . 131
13.10.3.12 vrna_idx_row_wise(unsigned int length) . . . 131
13.10.3.13 vrna_idx_col_wise(unsigned int length) . . . 132
13.10.4 Variable Documentation . . . 132
13.10.4.1 xsubi . . . 132
13.11 Computing Minimum Free Energy (MFE) Structures . . . 133
13.11.1 Detailed Description . . . 133
13.11.2 Function Documentation . . . 134
13.11.2.1 vrna_mfe(vrna_fold_compound_t *vc, char *structure) . . . 134
13.12 Computing Partition Functions and Pair Probabilities . . . 135
13.12.1 Detailed Description . . . 137
13.12.2 Function Documentation . . . 137
13.12.2.1 vrna_mean_bp_distance_pr(int length, FLT_OR_DBL *pr) . . . 137
13.12.2.2 vrna_mean_bp_distance(vrna_fold_compound_t *vc) . . . 137
13.12.2.3 vrna_stack_prob(vrna_fold_compound_t *vc, double cutoff) . . . 138
13.12.2.4 vrna_pf(vrna_fold_compound_t *vc, char *structure) . . . 138
13.12.2.5 vrna_pf_fold(const char *sequence, char *structure, vrna_plist_t **pl) . . . 139
13.12.2.6 vrna_pf_circfold(const char *sequence, char *structure, vrna_plist_t **pl) . . . 139
13.12.2.7 pf_fold_par(const char *sequence, char *structure, vrna_exp_param_↩ t *parameters, int calculate_bppm, int is_constrained, int is_circular) . . . 140
13.12.2.8 pf_fold(const char *sequence, char *structure) . . . 141
13.12.2.9 pf_circ_fold(const char *sequence, char *structure) . . . 142
13.12.2.10 free_pf_arrays(void) . . . 142
13.12.2.11 update_pf_params(int length) . . . 143
13.12.2.12 update_pf_params_par(int length, vrna_exp_param_t *parameters) . . . 143
13.12.2.13 export_bppm(void) . . . 144

13.12.2.14 get_pf_arrays(short **S_p, short **S1_p, char **ptype_p, FLT_OR_DBL **qb↩ _p, FLT_OR_DBL **qm_p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p) . . 144

13.12.2.15 mean_bp_distance(int length) . . . 145

13.12.2.16 mean_bp_distance_pr(int length, FLT_OR_DBL *pr) . . . 145

13.12.2.17 vrna_plist_from_probs(vrna_fold_compound_t *vc, double cut_off) . . . 146

13.12.2.18 assign_plist_from_pr(vrna_plist_t **pl, FLT_OR_DBL *probs, int length, double cutoff) . . . 146

13.13 Compute the structure with maximum expected accuracy (MEA) . . . 147

13.14 Compute the centroid structure . . . 148

13.14.1 Detailed Description . . . 148

13.14.2 Function Documentation . . . 148

13.14.2.1 vrna_centroid(vrna_fold_compound_t *vc, double *dist) . . . 148

13.14.2.2 vrna_centroid_from_plist(int length, double *dist, vrna_plist_t *pl) . . . 149

13.14.2.3 vrna_centroid_from_probs(int length, double *dist, FLT_OR_DBL *probs) . . . 149

13.15 Enumerating Suboptimal Structures . . . 150

13.15.1 Detailed Description . . . 150

13.16 Suboptimal structures according to Zuker et al. 1989 . . . 151

13.16.1 Detailed Description . . . 151

13.16.2 Function Documentation . . . 151

13.16.2.1 vrna_subopt_zuker(vrna_fold_compound_t *vc) . . . 151

13.16.2.2 zukersubopt(const char *string) . . . 152

13.16.2.3 zukersubopt_par(const char *string, vrna_param_t *parameters) . . . 152

13.17 Suboptimal structures within an energy band arround the MFE . . . 153

13.17.1 Detailed Description . . . 153

13.17.2 Function Documentation . . . 153

13.17.2.1 vrna_subopt(vrna_fold_compound_t *vc, int delta, int sorted, FILE *fp) . . . 153

13.17.2.2 subopt(char *seq, char *structure, int delta, FILE *fp) . . . 154

13.17.2.3 subopt_circ(char *seq, char *sequence, int delta, FILE *fp) . . . 154

13.18 Stochastic backtracking in the Ensemble . . . 156

13.18.1 Detailed Description . . . 156

13.18.2 Function Documentation . . . 156

13.18.2.1 vrna_pbacktrack5(vrna_fold_compound_t *vc, int length) . . . 156
13.18.2.2 vrna_pbacktrack(vrna_fold_compound_t *vc) . . . 157
13.18.2.3 pbacktrack(char *sequence) . . . 157
13.18.2.4 pbacktrack_circ(char *sequence) . . . 158
13.18.3 Variable Documentation . . . 158
13.18.3.1 st_back . . . 158
13.19 Calculate Secondary Structures of two RNAs upon Dimerization . . . 159
13.19.1 Detailed Description . . . 159
13.20 MFE Structures of single Nucleic Acid Sequences . . . 160
13.20.1 Detailed Description . . . 160
13.20.2 Function Documentation . . . 161
13.20.2.1 vrna_fold(const char *sequence, char *structure) . . . 161
13.20.2.2 vrna_circfold(const char *sequence, char *structure) . . . 161
13.20.2.3 fold_par(const char *sequence, char *structure, vrna_param_t *parameters, int is_constrained, int is_circular) . . . 162
13.20.2.4 fold(const char *sequence, char *structure) . . . 163
13.20.2.5 circfold(const char *sequence, char *structure) . . . 163
13.20.2.6 free_arrays(void) . . . 164
13.20.2.7 update_fold_params(void) . . . 164
13.20.2.8 update_fold_params_par(vrna_param_t *parameters) . . . 164
13.20.2.9 export_fold_arrays(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p) . . . 164
13.20.2.10 export_fold_arrays_par(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p) . . . 165
13.20.2.11 export_circfold_arrays(int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p) . . . 165
13.20.2.12 export_circfold_arrays_par(int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p) . . . 165
13.20.2.13 LoopEnergy(int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1) . . . 165
13.20.2.14 HairpinE(int size, int type, int si1, int sj1, const char *string) . . . 165
13.20.2.15 initialize_fold(int length) . . . 165
13.21 MFE Structures of two hybridized Sequences . . . 166

13.21.1 Detailed Description . . . 167
13.21.2 Function Documentation . . . 167
13.21.2.1 vrna_cofold(const char *sequence, char *structure) . . . 167
13.21.2.2 cofold(const char *sequence, char *structure) . . . 167
13.21.2.3 cofold_par(const char *string, char *structure, vrna_param_t *parameters, int is_constrained) . . . 168
13.21.2.4 free_co_arrays(void) . . . 168
13.21.2.5 update_cofold_params(void) . . . 168
13.21.2.6 update_cofold_params_par(vrna_param_t *parameters) . . . 168
13.21.2.7 export_cofold_arrays_gq(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **ggg_p, int **indx_p, char **ptype_p) . . . 169
13.21.2.8 export_cofold_arrays(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **indx_p, char **ptype_p) . . . 169
13.21.2.9 get_monomere_mfes(float *e1, float *e2) . . . 170
13.21.2.10 initialize_cofold(int length) . . . 170
13.21.2.11 vrna_mfe_dimer(vrna_fold_compound_t *vc, char *structure) . . . 170
13.22 Partition Function for two hybridized Sequences . . . 172
13.22.1 Detailed Description . . . 173
13.22.2 Data Structure Documentation . . . 173
13.22.2.1 struct vrna_dimer_pf_s . . . 173
13.22.2.2 struct vrna_dimer_conc_s . . . 174
13.22.3 Function Documentation . . . 174
13.22.3.1 vrna_pf_dimer(vrna_fold_compound_t *vc, char *structure) . . . 174
13.22.3.2 vrna_pf_dimer_probs(double FAB, double FA, double FB, vrna_plist_t *prAB, const vrna_plist_t *prA, const vrna_plist_t *prB, int Alength, const vrna_exp_param_t *exp_params) . . . 174
13.22.3.3 vrna_pf_dimer_concentrations(double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double *startconc, const vrna_exp_param_t *exp_params) 175
13.23 Partition Function for two hybridized Sequences as a stepwise Process . . . 176
13.23.1 Detailed Description . . . 176
13.23.2 Function Documentation . . . 177
13.23.2.1 pf_unstru(char *sequence, int max_w) . . . 177

13.23.2.2 pf_interact(const char *s1, const char *s2, pu_contrib *p_c, pu_contrib *p_c2, int max_w, char *cstruc, int incr3, int incr5) . . . 177

13.24 Predicting Consensus Structures from Alignment(s) . . . 179

13.24.1 Detailed Description . . . 180

13.24.2 Function Documentation . . . 180

13.24.2.1 energy_of_alistruct(const char **sequences, const char *structure, int n_seq, float *energy) . . . 180

13.24.2.2 get_alipf_arrays(short ***S_p, short ***S5_p, short ***S3_p, unsigned short ***a2s_p, char ***Ss_p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm_p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p, short **pscore) . . . 181

13.24.2.3 update_alifold_params(void) . . . 182

13.24.2.4 vrna_aln_mpi(char *Alseq[], int n_seq, int length, int *mini) . . . 182

13.24.2.5 get_mpi(char *Alseq[], int n_seq, int length, int *mini) . . . 182

13.24.2.6 encode_ali_sequence(const char *sequence, short *S, short *s5, short *s3, char *ss, unsigned short *as, int circ) . . . 183

13.24.2.7 alloc_sequence_arrays(const char **sequences, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss, int circ) . . . 183

13.24.2.8 free_sequence_arrays(unsigned int n_seq, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss) . . . 184

13.24.3 Variable Documentation . . . 184

13.24.3.1 cv_fact . . . 184

13.24.3.2 nc_fact . . . 184

13.25 MFE Consensus Structures for Sequence Alignment(s) . . . 185

13.25.1 Detailed Description . . . 185

13.25.2 Function Documentation . . . 185

13.25.2.1 vrna_alifold(const char **sequences, char *structure) . . . 185

13.25.2.2 vrna_circalifold(const char **sequences, char *structure) . . . 187

13.25.2.3 alifold(const char **strings, char *structure) . . . 187

13.25.2.4 circalifold(const char **strings, char *structure) . . . 188

13.25.2.5 free_alifold_arrays(void) . . . 189

13.26 Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . 190

13.26.1 Detailed Description . . . 190

13.26.2 Function Documentation . . . 190

13.26.2.1 vrna_pf_alifold(const char **sequences, char *structure, vrna_plist_t **pl) . . . 190
13.26.2.2 vrna_pf_circalifold(const char **sequences, char *structure, vrna_plist_t **pl) . 191
13.26.2.3 alipf_fold_par(const char **sequences, char *structure, vrna_plist_t **pl, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained, int is_circular) 192
13.26.2.4 alipf_fold(const char **sequences, char *structure, vrna_plist_t **pl) 192
13.26.2.5 alipf_circ_fold(const char **sequences, char *structure, vrna_plist_t **pl) 193
13.26.2.6 export_ali_bppm(void) 193
13.26.2.7 free_alipf_arrays(void) 193
13.27 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) 194
13.27.1 Detailed Description 194
13.27.2 Function Documentation 194
13.27.2.1 alipbacktrack(double *prob) 194
13.28 Predicting Locally stable structures of large sequences 195
13.28.1 Detailed Description 195
13.29 Local MFE structure Prediction and Z-scores 196
13.29.1 Detailed Description 196
13.29.2 Function Documentation 196
13.29.2.1 vrna_Lfold(const char *string, int window_size, FILE *file) 196
13.29.2.2 vrna_Lfoldz(const char *string, int window_size, double min_z, FILE *file) 198
13.29.2.3 Lfold(const char *string, char *structure, int maxdist) 198
13.29.2.4 Lfoldz(const char *string, char *structure, int maxdist, int zsc, double min_z) . . 199
13.29.2.5 vrna_mfe_window(vrna_fold_compound_t *vc, FILE *file) 199
13.29.2.6 vrna_mfe_window_zscore(vrna_fold_compound_t *vc, double min_z, FILE *file) 199
13.30 Partition functions for locally stable secondary structures 201
13.30.1 Detailed Description 201
13.30.2 Function Documentation 201
13.30.2.1 update_pf_paramsLP(int length) 201
13.30.2.2 pfl_fold(char *sequence, int winSize, int pairSize, float cutoffb, double **pU, plist **dpp2, FILE *pUfp, FILE *spup) 202
13.30.2.3 putoutpU_prob(double **pU, int length, int ulength, FILE *fp, int energies) . . . 202
13.30.2.4 putoutpU_prob_bin(double **pU, int length, int ulength, FILE *fp, int energies) . 203

13.31 Local MFE consensus structures for Sequence Alignments . . . 204
13.31.1 Detailed Description . . . 204
13.31.2 Function Documentation . . . 204
13.31.2.1 aliLfold(const char **strings, char *structure, int maxdist) . . . 204
13.32 Reading/Writing Energy Parameter Sets from/to File . . . 205
13.32.1 Detailed Description . . . 205
13.32.2 Function Documentation . . . 205
13.32.2.1 read_parameter_file(const char fname[]) . . . 205
13.32.2.2 write_parameter_file(const char fname[]) . . . 206
13.33 Converting Energy Parameter Files . . . 207
13.33.1 Detailed Description . . . 208
13.33.2 Macro Definition Documentation . . . 208
13.33.2.1 VRNA_CONVERT_OUTPUT_ALL . . . 208
13.33.2.2 VRNA_CONVERT_OUTPUT_HP . . . 208
13.33.2.3 VRNA_CONVERT_OUTPUT_STACK . . . 208
13.33.2.4 VRNA_CONVERT_OUTPUT_MM_HP . . . 208
13.33.2.5 VRNA_CONVERT_OUTPUT_MM_INT . . . 208
13.33.2.6 VRNA_CONVERT_OUTPUT_MM_INT_1N . . . 208
13.33.2.7 VRNA_CONVERT_OUTPUT_MM_INT_23 . . . 209
13.33.2.8 VRNA_CONVERT_OUTPUT_MM_MULTI . . . 209
13.33.2.9 VRNA_CONVERT_OUTPUT_MM_EXT . . . 209
13.33.2.10 VRNA_CONVERT_OUTPUT_DANGLE5 . . . 209
13.33.2.11 VRNA_CONVERT_OUTPUT_DANGLE3 . . . 209
13.33.2.12 VRNA_CONVERT_OUTPUT_INT_11 . . . 209
13.33.2.13 VRNA_CONVERT_OUTPUT_INT_21 . . . 209
13.33.2.14 VRNA_CONVERT_OUTPUT_INT_22 . . . 209
13.33.2.15 VRNA_CONVERT_OUTPUT_BULGE . . . 210
13.33.2.16 VRNA_CONVERT_OUTPUT_INT . . . 210
13.33.2.17 VRNA_CONVERT_OUTPUT_ML . . . 210
13.33.2.18 VRNA_CONVERT_OUTPUT_MISC . . . 210

13.33.2.19 VRNA_CONVERT_OUTPUT_SPECIAL_HP . . . 210
13.33.2.20 VRNA_CONVERT_OUTPUT_VANILLA . . . 210
13.33.2.21 VRNA_CONVERT_OUTPUT_NINIO . . . 210
13.33.2.22 VRNA_CONVERT_OUTPUT_DUMP . . . 211
13.33.3 Function Documentation . . . 211
13.33.3.1 convert_parameter_file(const char *iname, const char *oname, unsigned int options) . . . 211
13.34 Classified Dynamic Programming . . . 212
13.34.1 Detailed Description . . . 212
13.35 Distance based partitioning of the Secondary Structure Space . . . 213
13.35.1 Detailed Description . . . 213
13.36 Calculating MFE representatives of a Distance Based Partitioning . . . 214
13.36.1 Detailed Description . . . 215
13.36.2 Data Structure Documentation . . . 215
13.36.2.1 struct vrna_sol_TwoD_t . . . 215
13.36.2.2 struct TwoDfold_vars . . . 215
13.36.3 Typedef Documentation . . . 217
13.36.3.1 vrna_sol_TwoD_t . . . 217
13.36.3.2 TwoDfold_vars . . . 217
13.36.4 Function Documentation . . . 217
13.36.4.1 vrna_mfe_TwoD(vrna_fold_compound_t *vc, int distance1, int distance2) . . . 217
13.36.4.2 vrna_backtrack5_TwoD(vrna_fold_compound_t *vc, int k, int l, unsigned int j) . . . 218
13.36.4.3 get_TwoDfold_variables(const char *seq, const char *structure1, const char *structure2, int circ) . . . 218
13.36.4.4 destroy_TwoDfold_variables(TwoDfold_vars *our_variables) . . . 219
13.36.4.5 TwoDfoldList(TwoDfold_vars *vars, int distance1, int distance2) . . . 219
13.36.4.6 TwoDfold_backtrack_f5(unsigned int j, int k, int l, TwoDfold_vars *vars) . . . 220
13.37 Calculate Partition Functions of a Distance Based Partitioning . . . 221
13.37.1 Detailed Description . . . 221
13.37.2 Data Structure Documentation . . . 222
13.37.2.1 struct vrna_sol_TwoD_pf_t . . . 222

13.37.3 Typedef Documentation . . . 222
13.37.3.1 vrna_sol_TwoD_pf_t . . . 222
13.37.4 Function Documentation . . . 222
13.37.4.1 vrna_pf_TwoD(vrna_fold_compound_t *vc, int maxDistance1, int maxDistance2) . . . 222
13.38 Stochastic Backtracking of Structures from Distance Based Partitioning . . . 224
13.38.1 Detailed Description . . . 224
13.38.2 Function Documentation . . . 224
13.38.2.1 vrna_pbacktrack_TwoD(vrna_fold_compound_t *vc, int d1, int d2) . . . 224
13.38.2.2 vrna_pbacktrack5_TwoD(vrna_fold_compound_t *vc, int d1, int d2, unsigned int length) . . . 225
13.39 Compute the Density of States . . . 226
13.39.1 Detailed Description . . . 226
13.39.2 Variable Documentation . . . 226
13.39.2.1 density_of_states . . . 226
13.40 Hard Constraints . . . 227
13.40.1 Detailed Description . . . 228
13.40.2 Data Structure Documentation . . . 229
13.40.2.1 struct vrna_hc_s . . . 229
13.40.2.2 struct vrna_hc_up_s . . . 230
13.40.3 Macro Definition Documentation . . . 230
13.40.3.1 VRNA_CONSTRAINT_DB . . . 230
13.40.3.2 VRNA_CONSTRAINT_DB_ENFORCE_BP . . . 230
13.40.3.3 VRNA_CONSTRAINT_DB_PIPE . . . 231
13.40.3.4 VRNA_CONSTRAINT_DB_DOT . . . 231
13.40.3.5 VRNA_CONSTRAINT_DB_X . . . 231
13.40.3.6 VRNA_CONSTRAINT_DB_RND_BRACK . . . 231
13.40.3.7 VRNA_CONSTRAINT_DB_INTRAMOL . . . 232
13.40.3.8 VRNA_CONSTRAINT_DB_INTERMOL . . . 232
13.40.3.9 VRNA_CONSTRAINT_DB_GQUAD . . . 232
13.40.3.10 VRNA_CONSTRAINT_DB_DEFAULT . . . 233
13.40.4 Typedef Documentation . . . 233

13.40.4.1 vrna_callback_hc_evaluate . . . 233
13.40.5 Function Documentation . . . 234
13.40.5.1 vrna_hc_init(vrna_fold_compound_t *vc) . . . 234
13.40.5.2 vrna_hc_add_up(vrna_fold_compound_t *vc, int i, char option) . . . 234
13.40.5.3 vrna_hc_add_up_batch(vrna_fold_compound_t *vc, vrna_hc_up_t *constraints) 234
13.40.5.4 vrna_hc_add_bp(vrna_fold_compound_t *vc, int i, int j, char option) . . . 235
13.40.5.5 vrna_hc_add_bp_nonspecific(vrna_fold_compound_t *vc, int i, int d, char option) 235
13.40.5.6 vrna_hc_free(vrna_hc_t *hc) . . . 236
13.40.5.7 vrna_hc_add_from_db(vrna_fold_compound_t *vc, const char *constraint, unsigned int options) . . . 236
13.41 Soft Constraints . . . 237
13.41.1 Detailed Description . . . 238
13.41.2 Data Structure Documentation . . . 238
13.41.2.1 struct vrna_sc_s . . . 238
13.41.3 Typedef Documentation . . . 239
13.41.3.1 vrna_callback_sc_energy . . . 239
13.41.3.2 vrna_callback_sc_exp_energy . . . 240
13.41.3.3 vrna_callback_sc_backtrack . . . 240
13.41.4 Function Documentation . . . 241
13.41.4.1 vrna_sc_init(vrna_fold_compound_t *vc) . . . 241
13.41.4.2 vrna_sc_add_bp(vrna_fold_compound_t *vc, const FLT_OR_DBL **constraints, unsigned int options) . . . 242
13.41.4.3 vrna_sc_add_up(vrna_fold_compound_t *vc, const FLT_OR_DBL *constraints, unsigned int options) . . . 242
13.41.4.4 vrna_sc_remove(vrna_fold_compound_t *vc) . . . 242
13.41.4.5 vrna_sc_free(vrna_sc_t *sc) . . . 242
13.41.4.6 vrna_sc_add_data(vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *free_data) . . . 243
13.41.4.7 vrna_sc_add_f(vrna_fold_compound_t *vc, vrna_callback_sc_energy *f) . . . 243
13.41.4.8 vrna_sc_add_bt(vrna_fold_compound_t *vc, vrna_callback_sc_backtrack *f) . . . 243
13.41.4.9 vrna_sc_add_exp_f(vrna_fold_compound_t *vc, vrna_callback_sc_exp_energy *exp_f) . . . 244
13.42 Incorporating SHAPE reactivity data . . . 245

13.42.1 Detailed Description . . . 245
13.42.2 Function Documentation . . . 246
13.42.2.1 vrna_sc_add_SHAPE_deigan(vrna_fold_compound_t *vc, const double *reactivities, double m, double b, unsigned int options) . . . 246
13.42.2.2 vrna_sc_add_SHAPE_deigan_ali(vrna_fold_compound_t *vc, const char **shape_files, const int *shape_file_association, double m, double b, unsigned int options) . . . 246
13.42.2.3 vrna_sc_add_SHAPE_zarringhalam(vrna_fold_compound_t *vc, const double *reactivities, double b, double default_value, const char *shape_conversion, unsigned int options) . . . 247
13.42.2.4 vrna_sc_SHAPE_to_pr(const char *shape_conversion, double *values, int length, double default_value) . . . 247
13.43 Incorporating ligands binding to specific sequence/structure motifs . . . 249
13.43.1 Detailed Description . . . 249
13.43.2 Function Documentation . . . 249
13.43.2.1 vrna_sc_add_hi_motif(vrna_fold_compound_t *vc, const char *seq, const char *structure, FLT_OR_DBL energy, unsigned int options) . . . 249
13.44 Generate soft constraints from data . . . 251
13.44.1 Detailed Description . . . 252
13.44.2 Macro Definition Documentation . . . 252
13.44.2.1 VRNA_OBJECTIVE_FUNCTION_QUADRATIC . . . 252
13.44.2.2 VRNA_OBJECTIVE_FUNCTION_ABSOLUTE . . . 252
13.44.2.3 VRNA_MINIMIZER_CONJUGATE_FR . . . 252
13.44.2.4 VRNA_MINIMIZER_CONJUGATE_PR . . . 252
13.44.2.5 VRNA_MINIMIZER_VECTOR_BFGS . . . 253
13.44.2.6 VRNA_MINIMIZER_VECTOR_BFGS2 . . . 253
13.44.2.7 VRNA_MINIMIZER_STEEPEST_DESCENT . . . 253
13.44.3 Typedef Documentation . . . 253
13.44.3.1 progress_callback . . . 253
13.44.4 Function Documentation . . . 253
13.44.4.1 vrna_sc_minimize_pertubation(vrna_fold_compound_t *vc, const double *q↩ _prob_unpaired, int objective_function, double sigma_squared, double tau_↩ squared, int algorithm, int sample_size, double *epsilon, double initialStep↩ Size, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback) . . . 253

13.45 The Fold Compound . . . 255
13.45.1 Detailed Description . . . 256
13.45.2 Data Structure Documentation . . . 256
13.45.2.1 struct vrna_fc_s . . . 256
13.45.3 Macro Definition Documentation . . . 263
13.45.3.1 VRNA_STATUS_MFE_PRE . . . 263
13.45.3.2 VRNA_STATUS_MFE_POST . . . 264
13.45.3.3 VRNA_STATUS_PF_PRE . . . 264
13.45.3.4 VRNA_STATUS_PF_POST . . . 264
13.45.3.5 VRNA_OPTION_MFE . . . 264
13.45.3.6 VRNA_OPTION_PF . . . 264
13.45.3.7 VRNA_OPTION_EVAL_ONLY . . . 265
13.45.4 Typedef Documentation . . . 265
13.45.4.1 vrna_callback_free_auxdata . . . 265
13.45.4.2 vrna_callback_recursion_status . . . 265
13.45.5 Enumeration Type Documentation . . . 265
13.45.5.1 vrna_fc_type_e . . . 266
13.45.6 Function Documentation . . . 266
13.45.6.1 vrna_fold_compound(const char *sequence, vrna_md_t *md_p, unsigned int options) . . . 266
13.45.6.2 vrna_fold_compound_comparative(const char **sequences, vrna_md_t *md_p, unsigned int options) . . . 267
13.45.6.3 vrna_fold_compound_free(vrna_fold_compound_t *vc) . . . 268
13.45.6.4 vrna_fold_compound_add_auxdata(vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *f) . . . 268
13.45.6.5 vrna_fold_compound_add_callback(vrna_fold_compound_t *vc, vrna_callback_recursion_status *f) . . . 269
13.46 The Dynamic Programming Matrices . . . 270
13.46.1 Detailed Description . . . 270
13.46.2 Data Structure Documentation . . . 271
13.46.2.1 struct vrna_mx_mfe_s . . . 271
13.46.2.2 struct vrna_mx_pf_s . . . 273

13.46.3 Enumeration Type Documentation . . . 274
13.46.3.1 vrna_mx_type_e . . . 274
13.46.4 Function Documentation . . . 275
13.46.4.1 vrna_mx_add(vrna_fold_compound_t *vc, vrna_mx_type_e type, unsigned int options) . . . 275
13.46.4.2 vrna_mx_mfe_free(vrna_fold_compound_t *vc) . . . 275
13.46.4.3 vrna_mx_pf_free(vrna_fold_compound_t *vc) . . . 276
13.47 Direct refolding paths between two secondary structures . . . 277
13.47.1 Detailed Description . . . 277
13.47.2 Data Structure Documentation . . . 278
13.47.2.1 struct vrna_path_s . . . 278
13.47.3 Typedef Documentation . . . 278
13.47.3.1 path_t . . . 278
13.47.4 Function Documentation . . . 278
13.47.4.1 vrna_path_findpath_saddle(vrna_fold_compound_t *vc, const char *struc1, const char *struc2, int max) . . . 278
13.47.4.2 vrna_path_findpath(vrna_fold_compound_t *vc, const char *s1, const char *s2, int maxkeep) . . . 279
13.47.4.3 find_saddle(const char *seq, const char *struc1, const char *struc2, int max) . . . 279
13.47.4.4 free_path(vrna_path_t *path) . . . 280
13.47.4.5 get_path(const char *seq, const char *s1, const char *s2, int maxkeep) . . . 280
13.48 Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings . . . 281
13.48.1 Detailed Description . . . 282
13.48.2 Macro Definition Documentation . . . 282
13.48.2.1 FILENAME_MAX_LENGTH . . . 282
13.48.2.2 FILENAME_ID_LENGTH . . . 282
13.48.3 Function Documentation . . . 282
13.48.3.1 vrna_random_string(int l, const char symbols[]) . . . 282
13.48.3.2 vrna_hamming_distance(const char *s1, const char *s2) . . . 282
13.48.3.3 vrna_hamming_distance_bound(const char *s1, const char *s2, int n) . . . 283
13.48.3.4 vrna_seq_toRNA(char *sequence) . . . 283
13.48.3.5 vrna_seq_toupper(char *sequence) . . . 283

13.48.3.6 vrna_cut_point_insert(const char *string, int cp) . . . 284
13.48.3.7 vrna_cut_point_remove(const char *string, int *cp) . . . 284
13.49 Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures . . . 285
13.49.1 Detailed Description . . . 287
13.49.2 Data Structure Documentation . . . 287
13.49.2.1 struct vrna_hx_s . . . 287
13.49.3 Function Documentation . . . 287
13.49.3.1 b2HIT(const char *structure) . . . 287
13.49.3.2 b2C(const char *structure) . . . 288
13.49.3.3 b2Shapiro(const char *structure) . . . 288
13.49.3.4 add_root(const char *structure) . . . 288
13.49.3.5 expand_Shapiro(const char *coarse) . . . 289
13.49.3.6 expand_Full(const char *structure) . . . 289
13.49.3.7 unexpand_Full(const char *ffull) . . . 289
13.49.3.8 unweight(const char *wcoarse) . . . 290
13.49.3.9 unexpand_aligned_F(char *align[2]) . . . 290
13.49.3.10 parse_structure(const char *structure) . . . 290
13.49.3.11 vrna_db_pack(const char *struc) . . . 290
13.49.3.12 vrna_db_unpack(const char *packed) . . . 291
13.49.3.13 vrna_ptable(const char *structure) . . . 291
13.49.3.14 vrna_pt_pk_get(const char *structure) . . . 292
13.49.3.15 vrna_ptable_copy(const short *pt) . . . 292
13.49.3.16 vrna_pt_snoop_get(const char *structure) . . . 292
13.49.3.17 vrna_db_from_ptable(short *pt) . . . 293
13.49.3.18 vrna_bp_distance(const char *str1, const char *str2) . . . 293
13.49.3.19 vrna_refBPcnt_matrix(const short *reference_pt, unsigned int turn) . . . 293
13.49.3.20 vrna_refBPdist_matrix(const short *pt1, const short *pt2, unsigned int turn) . . . 293
13.49.3.21 vrna_db_from_bp_stack(vrna_bp_stack_t *bp, unsigned int length) . . . 294
13.49.3.22 vrna_plist(const char *struc, float pr) . . . 294
13.49.3.23 vrna_db_from_plist(vrna_plist_t *pairs, unsigned int n) . . . 294

13.49.3.24 assign_plist_from_db(vrna_plist_t **pl, const char *struc, float pr) . . . 295
13.49.3.25 pack_structure(const char *struc) . . . 295
13.49.3.26 unpack_structure(const char *packed) . . . 296
13.49.3.27 make_pair_table(const char *structure) . . . 296
13.49.3.28 copy_pair_table(const short *pt) . . . 296
13.49.3.29 alimake_pair_table(const char *structure) . . . 298
13.49.3.30 make_pair_table_snoop(const char *structure) . . . 298
13.49.3.31 bp_distance(const char *str1, const char *str2) . . . 298
13.49.3.32 make_referenceBP_array(short *reference_pt, unsigned int turn) . . . 299
13.49.3.33 compute_BPdifferences(short *pt1, short *pt2, unsigned int turn) . . . 299
13.49.3.34 parenthesis_structure(char *structure, vrna_bp_stack_t *bp, int length) . . . 299
13.49.3.35 parenthesis_zuker(char *structure, vrna_bp_stack_t *bp, int length) . . . 299
13.49.3.36 bppm_to_structure(char *structure, FLT_OR_DBL *pr, unsigned int length) . . . 300
13.49.3.37 bppm_symbol(const float *x) . . . 300
13.50 Various utilities for Sequence Alignments, and Comparative Structure Prediction . . . 301
13.50.1 Detailed Description . . . 301
13.50.2 Data Structure Documentation . . . 301
13.50.2.1 struct vrna_pinfo_s . . . 301
13.50.3 Typedef Documentation . . . 302
13.50.3.1 pair_info . . . 302
13.50.4 Function Documentation . . . 302
13.50.4.1 vrna_aln_pinfo(vrna_fold_compound_t *vc, const char *structure, double threshold) 302
13.51 Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments . . . 304
13.51.1 Detailed Description . . . 305
13.51.2 Macro Definition Documentation . . . 305
13.51.2.1 VRNA_OPTION_MULTILINE . . . 305
13.51.2.2 VRNA_CONSTRAINT_MULTILINE . . . 305
13.51.3 Function Documentation . . . 305
13.51.3.1 vrna_file_helixlist(const char *seq, const char *db, float energy, FILE *file) . . . 305
13.51.3.2 vrna_file_connect(const char *seq, const char *db, float energy, const char *identifier, FILE *file) . . . 306

13.51.3.3 vrna_file_bpseq(const char *seq, const char *db, FILE *file) . . . 306
13.51.3.4 vrna_file_json(const char *seq, const char *db, double energy, const char *identifier, FILE *file) . . . 306
13.51.3.5 vrna_file_fasta_read_record(char **header, char **sequence, char ***rest, FILE *file, unsigned int options) . . . 307
13.51.3.6 vrna_extract_record_rest_structure(const char **lines, unsigned int length, unsigned int option) . . . 308
13.51.3.7 vrna_file_SHAPE_read(const char *file_name, int length, double default_value, char *sequence, double *values) . . . 309
13.51.3.8 vrna_file_constraints_read(const char *filename, unsigned int length, unsigned int options) . . . 309
13.51.3.9 vrna_extract_record_rest_constraint(char **cstruc, const char **lines, unsigned int option) . . . 309
13.51.3.10read_record(char **header, char **sequence, char ***rest, unsigned int options) 310
13.52Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More . . . 311
13.52.1 Detailed Description . . . 312
13.52.2 Data Structure Documentation . . . 312
13.52.2.1 struct COORDINATE . . . 312
13.52.3 Macro Definition Documentation . . . 313
13.52.3.1 VRNA_PLOT_TYPE_SIMPLE . . . 313
13.52.3.2 VRNA_PLOT_TYPE_NAVIEW . . . 313
13.52.3.3 VRNA_PLOT_TYPE_CIRCULAR . . . 313
13.52.4 Function Documentation . . . 313
13.52.4.1 aliPS_color_aln(const char *structure, const char *filename, const char *seqs[], const char *names[]) . . . 313
13.52.4.2 simple_xy_coordinates(short *pair_table, float *X, float *Y) . . . 313
13.52.4.3 simple_circplot_coordinates(short *pair_table, float *x, float *y) . . . 314
13.52.4.4 vrna_file_PS_rnaplot(const char *seq, const char *structure, const char *file, vrna_md_t *md_p) . . . 314
13.52.4.5 vrna_file_PS_rnaplot_a(const char *seq, const char *structure, const char *file, const char *pre, const char *post, vrna_md_t *md_p) . . . 315
13.52.4.6 gmIRNA(char *string, char *structure, char *ssfile, char option) . . . 315
13.52.4.7 ssv_rna_plot(char *string, char *structure, char *ssfile) . . . 316
13.52.4.8 svg_rna_plot(char *string, char *structure, char *ssfile) . . . 316
13.52.4.9 xrna_plot(char *string, char *structure, char *ssfile) . . . 316
13.52.4.10PS_rna_plot(char *string, char *structure, char *file) . . . 317
13.52.4.11PS_rna_plot_a(char *string, char *structure, char *file, char *pre, char *post) . . . 317
13.52.4.12PS_rna_plot_a_gquad(char *string, char *structure, char *ssfile, char *pre, char *post) . . . 317
13.52.4.13PS_dot_plot_list(char *seq, char *filename, plist *pl, plist *mf, char *comment) . . . 317
13.52.4.14PS_dot_plot(char *string, char *file) . . . 318
13.52.5 Variable Documentation . . . 318
13.52.5.1 rna_plot_type . . . 318

14 Data Structure Documentation 319

14.1 _struct_en Struct Reference . . . 319

14.1.1 Detailed Description . . . 319

14.2 Postorder_list Struct Reference . . . 319

14.2.1 Detailed Description . . . 319

14.3 swString Struct Reference . . . 319

14.3.1 Detailed Description . . . 320

14.4 Tree Struct Reference . . . 320

14.4.1 Detailed Description . . . 320

14.5 TwoDpfold_vars Struct Reference . . . 320

14.5.1 Detailed Description . . . 321

14.6 vrna_subopt_sol_s Struct Reference . . . 322

14.6.1 Detailed Description . . . 322

15 File Documentation 323

15.1 ViennaRNA/1.8.4_epars.h File Reference . . . 323

15.1.1 Detailed Description . . . 323

15.2 ViennaRNA/1.8.4_intloops.h File Reference . . . 323

15.2.1 Detailed Description . . . 324

15.3 ViennaRNA/2Dfold.h File Reference . . . 324

15.4 ViennaRNA/2Dpfold.h File Reference . . . 325

15.4.1 Function Documentation . . . 326

15.4.1.1 get_TwoDpfold_variables(const char *seq, const char *structure1, char *structure2, int circ) . . . 326

15.4.1.2 destroy_TwoDpfold_variables(TwoDpfold_vars *vars) . . . 326

15.4.1.3 TwoDpfoldList(TwoDpfold_vars *vars, int maxDistance1, int maxDistance2) . . . 326

15.4.1.4 TwoDpfold_pbacktrack(TwoDpfold_vars *vars, int d1, int d2) . . . 327

15.4.1.5 TwoDpfold_pbacktrack5(TwoDpfold_vars *vars, int d1, int d2, unsigned int length) 328

15.5 ViennaRNA/alifold.h File Reference . . . 328

15.5.1 Detailed Description . . . 330

15.6 ViennaRNA/aln_util.h File Reference . . . 330

15.6.1 Detailed Description . . . 331
15.7 ViennaRNA/alphabet.h File Reference . . . 331
15.7.1 Detailed Description . . . 332
15.7.2 Function Documentation . . . 332
15.7.2.1 vrna_ptypes(const short *S, vrna_md_t *md) . . . 332
15.7.2.2 vrna_nucleotide_encode(char c, vrna_md_t *md) . . . 333
15.7.2.3 vrna_nucleotide_decode(int enc, vrna_md_t *md) . . . 333
15.8 ViennaRNA/boltzmann_sampling.h File Reference . . . 333
15.8.1 Detailed Description . . . 334
15.9 ViennaRNA/centroid.h File Reference . . . 335
15.9.1 Detailed Description . . . 336
15.9.2 Function Documentation . . . 336
15.9.2.1 get_centroid_struct_pl(int length, double *dist, vrna_plist_t *pl) . . . 336
15.9.2.2 get_centroid_struct_pr(int length, double *dist, FLT_OR_DBL *pr) . . . 336
15.10 ViennaRNA/cofold.h File Reference . . . 336
15.10.1 Detailed Description . . . 337
15.11 ViennaRNA/constraints.h File Reference . . . 337
15.11.1 Detailed Description . . . 339
15.12 ViennaRNA/constraints_hard.h File Reference . . . 339
15.12.1 Detailed Description . . . 341
15.12.2 Macro Definition Documentation . . . 341
15.12.2.1 VRNA_CONSTRAINT_NO_HEADER . . . 341
15.12.2.2 VRNA_CONSTRAINT_DB_ANG_BRACK . . . 341
15.12.3 Function Documentation . . . 342
15.12.3.1 print_tty_constraint(unsigned int option) . . . 342
15.12.3.2 print_tty_constraint_full(void) . . . 342
15.12.3.3 constrain_ptypes(const char *constraint, unsigned int length, char *ptype, int *↩ BP, int min_loop_size, unsigned int idx_type) . . . 342
15.13 ViennaRNA/constraints_SHAPE.h File Reference . . . 342
15.13.1 Detailed Description . . . 343
15.13.2 Function Documentation . . . 343

15.13.2.1 vrna_sc_SHAPE_parse_method(const char *method_string, char *method, float *param_1, float *param_2) . . . 343
15.14 ViennaRNA/constraints_soft.h File Reference . . . 344
15.14.1 Detailed Description . . . 345
15.15 ViennaRNA/convert_epars.h File Reference . . . 345
15.15.1 Detailed Description . . . 346
15.16 ViennaRNA/data_structures.h File Reference . . . 346
15.17 ViennaRNA/dist_vars.h File Reference . . . 349
15.17.1 Detailed Description . . . 349
15.17.2 Variable Documentation . . . 349
15.17.2.1 edit_backtrack . . . 349
15.17.2.2 cost_matrix . . . 350
15.18 ViennaRNA/dp_matrices.h File Reference . . . 350
15.19 ViennaRNA/duplex.h File Reference . . . 351
15.19.1 Detailed Description . . . 351
15.20 ViennaRNA/edit_cost.h File Reference . . . 351
15.20.1 Detailed Description . . . 351
15.21 ViennaRNA/energy_const.h File Reference . . . 352
15.21.1 Detailed Description . . . 352
15.21.2 Macro Definition Documentation . . . 352
15.21.2.1 GASCONST . . . 352
15.21.2.2 K0 . . . 352
15.21.2.3 INF . . . 353
15.21.2.4 FORBIDDEN . . . 353
15.21.2.5 BONUS . . . 353
15.21.2.6 NBPAIRS . . . 353
15.21.2.7 TURN . . . 353
15.21.2.8 MAXLOOP . . . 353
15.22 ViennaRNA/equilibrium_probs.h File Reference . . . 353
15.22.1 Detailed Description . . . 354
15.23 ViennaRNA/eval.h File Reference . . . 354

15.23.1 Detailed Description . . . 356

15.23.2 Function Documentation . . . 356

15.23.2.1 vrna_eval_structure(vrna_fold_compound_t *vc, const char *structure) . . . 356

15.23.2.2 vrna_eval_covar_structure(vrna_fold_compound_t *vc, const char *structure) . . . 357

15.23.2.3 vrna_eval_structure_simple(const char *string, const char *structure) . . . 357

15.23.2.4 vrna_eval_structure_verbose(vrna_fold_compound_t *vc, const char *structure, FILE *file) . . . 359

15.23.2.5 vrna_eval_structure_v(vrna_fold_compound_t *vc, const char *structure, int verbosity_level, FILE *file) . . . 359

15.23.2.6 vrna_eval_structure_simple_verbose(const char *string, const char *structure, FILE *file) . . . 360

15.23.2.7 vrna_eval_structure_simple_v(const char *string, const char *structure, int verbosity_level, FILE *file) . . . 360

15.23.2.8 vrna_eval_structure_pt(vrna_fold_compound_t *vc, const short *pt) . . . 361

15.23.2.9 vrna_eval_structure_pt_simple(const char *string, const short *pt) . . . 361

15.23.2.10 vrna_eval_structure_pt_verbose(vrna_fold_compound_t *vc, const short *pt, FILE *file) . . . 362

15.23.2.11 vrna_eval_structure_pt_v(vrna_fold_compound_t *vc, const short *pt, int verbosity_level, FILE *file) . . . 362

15.23.2.12 vrna_eval_structure_pt_simple_verbose(const char *string, const short *pt, FILE *file) . . . 363

15.23.2.13 vrna_eval_structure_pt_simple_v(const char *string, const short *pt, int verbosity_level, FILE *file) . . . 363

15.23.2.14 vrna_eval_loop_pt(vrna_fold_compound_t *vc, int i, const short *pt) . . . 364

15.23.2.15 vrna_eval_move(vrna_fold_compound_t *vc, const char *structure, int m1, int m2) 364

15.23.2.16 vrna_eval_move_pt(vrna_fold_compound_t *vc, short *pt, int m1, int m2) . . . 365

15.23.2.17 energy_of_structure(const char *string, const char *structure, int verbosity_level) 365

15.23.2.18 energy_of_struct_par(const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level) . . . 366

15.23.2.19 energy_of_circ_structure(const char *string, const char *structure, int verbosity_level) . . . 366

15.23.2.20 energy_of_circ_struct_par(const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level) . . . 367

15.23.2.21 energy_of_structure_pt(const char *string, short *ptable, short *s, short *s1, int verbosity_level) . . . 367

15.23.2.22 energy_of_struct_pt_par(const char *string, short *ptable, short *s, short *s1, vrna_param_t *parameters, int verbosity_level) . . . 368
15.23.2.23 energy_of_move(const char *string, const char *structure, int m1, int m2) . . . 368
15.23.2.24 energy_of_move_pt(short *pt, short *s, short *s1, int m1, int m2) . . . 369
15.23.2.25 loop_energy(short *ptable, short *s, short *s1, int i) . . . 369
15.23.2.26 energy_of_struct(const char *string, const char *structure) . . . 370
15.23.2.27 energy_of_struct_pt(const char *string, short *ptable, short *s, short *s1) . . . 370
15.23.2.28 energy_of_circ_struct(const char *string, const char *structure) . . . 371
15.24 ViennaRNA/exterior_loops.h File Reference . . . 371
15.24.1 Detailed Description . . . 372
15.25 ViennaRNA/file_formats.h File Reference . . . 372
15.25.1 Detailed Description . . . 373
15.26 ViennaRNA/findpath.h File Reference . . . 374
15.27 ViennaRNA/fold.h File Reference . . . 375
15.27.1 Detailed Description . . . 375
15.28 ViennaRNA/fold_vars.h File Reference . . . 376
15.28.1 Detailed Description . . . 377
15.28.2 Variable Documentation . . . 377
15.28.2.1 RibosumFile . . . 377
15.28.2.2 james_rule . . . 377
15.28.2.3 logML . . . 377
15.28.2.4 cut_point . . . 377
15.28.2.5 base_pair . . . 377
15.28.2.6 pr . . . 377
15.28.2.7 iindx . . . 378
15.29 ViennaRNA/gquad.h File Reference . . . 378
15.29.1 Detailed Description . . . 378
15.30 ViennaRNA/hairpin_loops.h File Reference . . . 379
15.30.1 Detailed Description . . . 380
15.31 ViennaRNA/interior_loops.h File Reference . . . 380
15.31.1 Detailed Description . . . 381

15.32 ViennaRNA/inverse.h File Reference . . . 381
15.32.1 Detailed Description . . . 381
15.33 ViennaRNA/Lfold.h File Reference . . . 382
15.33.1 Detailed Description . . . 382
15.34 ViennaRNA/ligand.h File Reference . . . 382
15.34.1 Detailed Description . . . 383
15.35 ViennaRNA/loop_energies.h File Reference . . . 383
15.35.1 Detailed Description . . . 383
15.36 ViennaRNA/LPfold.h File Reference . . . 384
15.36.1 Detailed Description . . . 384
15.36.2 Function Documentation . . . 384
15.36.2.1 init_pf_foldLP(int length) . . . 384
15.37 ViennaRNA/MEA.h File Reference . . . 385
15.37.1 Detailed Description . . . 385
15.37.2 Function Documentation . . . 385
15.37.2.1 MEA(plist *p, char *structure, double gamma) . . . 385
15.38 ViennaRNA/mfe.h File Reference . . . 386
15.38.1 Detailed Description . . . 386
15.39 ViennaRNA/mm.h File Reference . . . 386
15.39.1 Detailed Description . . . 387
15.40 ViennaRNA/model.h File Reference . . . 387
15.40.1 Detailed Description . . . 391
15.41 ViennaRNA/multibranch_loops.h File Reference . . . 391
15.41.1 Detailed Description . . . 392
15.42 ViennaRNA/naview.h File Reference . . . 393
15.43 ViennaRNA/params.h File Reference . . . 393
15.44 ViennaRNA/part_func.h File Reference . . . 395
15.44.1 Detailed Description . . . 396
15.44.2 Function Documentation . . . 397
15.44.2.1 vrna_pf_float_precision(void) . . . 397

15.44.2.2 stackProb(double cutoff) . . . 397
15.44.2.3 init_pf_fold(int length) . . . 397
15.44.2.4 centroid(int length, double *dist) . . . 397
15.44.2.5 get_centroid_struct_gquad_pr(int length, double *dist) . . . 397
15.44.2.6 mean_bp_dist(int length) . . . 398
15.44.2.7 expLoopEnergy(int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1) . . . 398
15.44.2.8 expHairpinEnergy(int u, int type, short si1, short sj1, const char *string) . . . 398
15.45 ViennaRNA/part_func_co.h File Reference . . . 398
15.45.1 Detailed Description . . . 399
15.45.2 Function Documentation . . . 400
15.45.2.1 co_pf_fold(char *sequence, char *structure) . . . 400
15.45.2.2 co_pf_fold_par(char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained) . . . 400
15.45.2.3 get_plist(vrna_plist_t *pl, int length, double cut_off) . . . 401
15.45.2.4 compute_probabilities(double FAB, double FEA, double FEB, vrna_plist_t *prAB, vrna_plist_t *prA, vrna_plist_t *prB, int Alength) . . . 401
15.45.2.5 get_concentrations(double FEAB, double FEAA, double FEBB, double FEA, double FEB, double *startconc) . . . 401
15.45.2.6 init_co_pf_fold(int length) . . . 402
15.45.2.7 export_co_bppm(void) . . . 402
15.45.2.8 free_co_pf_arrays(void) . . . 403
15.45.2.9 update_co_pf_params(int length) . . . 403
15.45.2.10 update_co_pf_params_par(int length, vrna_exp_param_t *parameters) . . . 403
15.46 ViennaRNA/part_func_up.h File Reference . . . 403
15.46.1 Detailed Description . . . 404
15.47 ViennaRNA/perturbation_fold.h File Reference . . . 404
15.47.1 Detailed Description . . . 405
15.48 ViennaRNA/plot_aln.h File Reference . . . 406
15.48.1 Detailed Description . . . 406
15.49 ViennaRNA/plot_layouts.h File Reference . . . 406
15.49.1 Detailed Description . . . 408

15.50 ViennaRNA/plot_structure.h File Reference . . . 408
15.50.1 Detailed Description . . . 409
15.51 ViennaRNA/profiledist.h File Reference . . . 409
15.51.1 Function Documentation . . . 410
15.51.1.1 profile_edit_distance(const float *T1, const float *T2) . . . 410
15.51.1.2 Make_bp_profile_bppm(FLT_OR_DBL *bppm, int length) . . . 410
15.51.1.3 free_profile(float *T) . . . 410
15.51.1.4 Make_bp_profile(int length) . . . 410
15.52 ViennaRNA/PS_dot.h File Reference . . . 411
15.52.1 Detailed Description . . . 411
15.53 ViennaRNA/read_epars.h File Reference . . . 411
15.54 ViennaRNA/ribo.h File Reference . . . 412
15.54.1 Detailed Description . . . 412
15.55 ViennaRNA/RNAstruct.h File Reference . . . 412
15.55.1 Detailed Description . . . 413
15.56 ViennaRNA/string_utils.h File Reference . . . 414
15.56.1 Detailed Description . . . 415
15.56.2 Function Documentation . . . 415
15.56.2.1 str_uppercase(char *sequence) . . . 415
15.56.2.2 str_DNA2RNA(char *sequence) . . . 415
15.56.2.3 random_string(int l, const char symbols[]) . . . 416
15.56.2.4 hamming(const char *s1, const char *s2) . . . 416
15.56.2.5 hamming_bound(const char *s1, const char *s2, int n) . . . 416
15.57 ViennaRNA/stringdist.h File Reference . . . 416
15.57.1 Detailed Description . . . 417
15.57.2 Function Documentation . . . 417
15.57.2.1 Make_swString(char *string) . . . 417
15.57.2.2 string_edit_distance(swString *T1, swString *T2) . . . 418
15.58 ViennaRNA/structure_utils.h File Reference . . . 418
15.58.1 Detailed Description . . . 420

15.59 ViennaRNA/subopt.h File Reference . . . 421
15.59.1 Detailed Description . . . 422
15.59.2 Typedef Documentation . . . 422
15.59.2.1 SOLUTION . . . 422
15.60 ViennaRNA/treedist.h File Reference . . . 422
15.60.1 Detailed Description . . . 423
15.60.2 Function Documentation . . . 423
15.60.2.1 make_tree(char *struc) . . . 423
15.60.2.2 tree_edit_distance(Tree *T1, Tree *T2) . . . 423
15.60.2.3 free_tree(Tree *t) . . . 423
15.61 ViennaRNA/utils.h File Reference . . . 423
15.61.1 Detailed Description . . . 426
15.61.2 Function Documentation . . . 426
15.61.2.1 print_tty_input_seq(void) . . . 426
15.61.2.2 print_tty_input_seq_str(const char *s) . . . 426
15.61.2.3 warn_user(const char message[]) . . . 427
15.61.2.4 nrerror(const char message[]) . . . 427
15.61.2.5 space(unsigned size) . . . 427
15.61.2.6 xrealloc(void *p, unsigned size) . . . 427
15.61.2.7 init_rand(void) . . . 427
15.61.2.8 urn(void) . . . 427
15.61.2.9 int_urn(int from, int to) . . . 428
15.61.2.10 filecopy(FILE *from, FILE *to) . . . 428
15.61.2.11 time_stamp(void) . . . 428

Bibliography **430**

Index **431**

Chapter 1

ViennaRNA Package core - RNAlib

A Library for folding and comparing RNA secondary structures

Date

1994-2016

Authors

Ivo Hofacker, Peter Stadler, Ronny Lorenz and many more

Table of Contents

- Introduction
- RNA Secondary Structure Prediction
- Parsing and Comparing - Functions to Manipulate Structures
- Utilities - Odds and Ends
- Input / Output File Formats
- RNAlib API v3.0
- Scripting Language interface(s)
- Example - A Small Example Program

1.1 Introduction

The core of the Vienna RNA Package ([10], [8]) is formed by a collection of routines for the prediction and comparison of RNA secondary structures. These routines can be accessed through stand-alone programs, such as RNAfold, RNAdistance etc., which should be sufficient for most users. For those who wish to develop their own programs we provide a library which can be linked to your own code.

This document describes the library and will be primarily useful to programmers. However, it also contains details about the implementation that may be of interest to advanced users. The stand-alone programs are described in separate man pages. The latest version of the package including source code and html versions of the documentation can be found at

http://www.tbi.univie.ac.at/RNA/

Chapter 2

Parsing and Comparing - Functions to Manipulate Structures

Representations of Secondary Structures

The standard representation of a secondary structure is the *bracket notation*, where matching brackets symbolize base pairs and unpaired bases are shown as dots. Alternatively, one may use two types of node labels, 'P' for paired and 'U' for unpaired; a dot is then replaced by '(U)', and each closed bracket is assigned an additional identifier 'P'. We call this the expanded notation. In [6] a condensed representation of the secondary structure is proposed, the so-called homeomorphically irreducible tree (HIT) representation. Here a stack is represented as a single pair of matching brackets labeled 'P' and weighted by the number of base pairs. Correspondingly, a contiguous strain of unpaired bases is shown as one pair of matching brackets labeled 'U' and weighted by its length. Generally any string consisting of matching brackets and identifiers is equivalent to a plane tree with as many different types of nodes as there are identifiers.

Bruce Shapiro proposed a coarse grained representation [14], which, does not retain the full information of the secondary structure. He represents the different structure elements by single matching brackets and labels them as 'H' (hairpin loop), 'I' (interior loop), 'B' (bulge), 'M' (multi-loop), and 'S' (stack). We extend his alphabet by an extra letter for external elements 'E'. Again these identifiers may be followed by a weight corresponding to the number of unpaired bases or base pairs in the structure element. All tree representations (except for the dot-bracket form) can be encapsulated into a virtual root (labeled 'R'), see the example below.

The following example illustrates the different linear tree representations used by the package. All lines show the same secondary structure.

```
a)  .(((((..(((...)))..((..)))))..)).
    (U)(((((U)(U)((((U)(U)(U)P)P)P)(U)(U)(((U)(U)P)P)P)P)(U)P)P)(U)
b)  (U)(((U2)((U3)P3)(U2)((U2)P2)P2)(U)P2)(U)
c)  (((H)(H)M)B)
    ((((((H)S)((H)S)M)S)B)S)
    ((((((H)S)((H)S)M)S)B)S)E)
d)  (((((((H3)S3)((H2)S2)M4)S2)B1)S2)E2)R)
```

Above: Tree representations of secondary structures. a) Full structure: the first line shows the more convenient condensed notation which is used by our programs; the second line shows the rather clumsy expanded notation for completeness, b) HIT structure, c) different versions of coarse grained structures: the second line is exactly Shapiro's representation, the first line is obtained by neglecting the stems. Since each loop is closed by a unique stem, these two lines are equivalent. The third line is an extension taking into account also the external digits. d) weighted coarse structure, this time including the virtual root.

For the output of aligned structures from string editing, different representations are needed, where we put the label on both sides. The above examples for tree representations would then look like:

```
a)  (UU)(P(P(P(P(UU)(UU)(P(P(P(UU)(UU)(UU)P)P)P)(UU)(UU)(P(P(UU)(U...
b)  (UU)(P2(P2(U2U2)(P2(U3U3)P3)(U2U2)(P2(U2U2)P2)P2)(UU)P2)(UU)
c)  (B(M(HH)(HH)M)B)
    (S(B(S(M(S(HH)S)(S(HH)S)M)S)B)S)
    (E(S(B(S(M(S(HH)S)(S(HH)S)M)S)B)S)E)
d)  (R(E2(S2(B1(S2(M4(S3(H3)S3)((H2)S2)M4)S2)B1)S2)E2)R)
```

Aligned structures additionally contain the gap character '_'.

Parsing and Coarse Graining of Structures

Several functions are provided for parsing structures and converting to different representations.

```
char  *expand_Full(const char *structure)
```

Convert the full structure from bracket notation to the expanded notation including root.

```
char *b2HIT (const char *structure)
```

Converts the full structure from bracket notation to the HIT notation including root.

```
char *b2C (const char *structure)
```

Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

```
char *b2Shapiro (const char *structure)
```

Converts the full structure from bracket notation to the *weighted* coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.

```
char  *expand_Shapiro (const char *coarse);
```

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

```
char *add_root (const char *structure)
```

Adds a root to an un-rooted tree in any except bracket notation.

```
char  *unexpand_Full (const char *ffull)
```

Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

```
char  *unweight (const char *wcoarse)
```

Strip weights from any weighted tree.

```
void   unexpand_aligned_F (char *align[2])
```

Converts two aligned structures in expanded notation.

```
void   parse_structure (const char *structure)
```

Collects a statistic of structure elements of the full structure in bracket notation.

See also

RNAstruct.h for prototypes and more detailed description

Distance Measures

A simple measure of dissimilarity between secondary structures of equal length is the base pair distance, given by the number of pairs present in only one of the two structures being compared. I.e. the number of base pairs that have to be opened or closed to transform one structure into the other. It is therefore particularly useful for comparing structures on the same sequence. It is implemented by

```
int bp_distance(const char *str1,
                const char *str2)
```

Compute the "base pair" distance between two secondary structures s1 and s2.

For other cases a distance measure that allows for gaps is preferable. We can define distances between structures as edit distances between trees or their string representations. In the case of string distances this is the same as "sequence alignment". Given a set of edit operations and edit costs, the edit distance is given by the minimum sum of the costs along an edit path converting one object into the other. Edit distances like these always define a metric. The edit operations used by us are insertion, deletion and replacement of nodes. String editing does not pay attention to the matching of brackets, while in tree editing matching brackets represent a single node of the tree. Tree editing is therefore usually preferable, although somewhat slower. String edit distances are always smaller or equal to tree edit distances.

The different level of detail in the structure representations defined above naturally leads to different measures of distance. For full structures we use a cost of 1 for deletion or insertion of an unpaired base and 2 for a base pair. Replacing an unpaired base for a pair incurs a cost of 1.

Two cost matrices are provided for coarse grained structures:

```
/*  Null,   H,   B,   I,   M,   S,   E    */
   {   0,   2,   2,   2,   2,   1,   1},   /* Null replaced */
   {   2,   0,   2,   2,   2, INF, INF},   /* H    replaced */
   {   2,   2,   0,   1,   2, INF, INF},   /* B    replaced */
   {   2,   2,   1,   0,   2, INF, INF},   /* I    replaced */
   {   2,   2,   2,   2,   0, INF, INF},   /* M    replaced */
   {   1, INF, INF, INF, INF,   0, INF},   /* S    replaced */
   {   1, INF, INF, INF, INF, INF,   0},   /* E    replaced */


/* Null,   H,   B,   I,   M,   S,   E   */
   {   0, 100,   5,   5,  75,   5,   5},   /* Null replaced */
   { 100,   0,   8,   8,   8, INF, INF},   /* H    replaced */
   {   5,   8,   0,   3,   8, INF, INF},   /* B    replaced */
   {   5,   8,   3,   0,   8, INF, INF},   /* I    replaced */
   {  75,   8,   8,   8,   0, INF, INF},   /* M    replaced */
   {   5, INF, INF, INF, INF,   0, INF},   /* S    replaced */
   {   5, INF, INF, INF, INF, INF,   0},   /* E    replaced */
```

The lower matrix uses the costs given in [15]. All distance functions use the following global variables:

```
int  cost_matrix;
```

Specify the cost matrix to be used for distance calculations.

```
int   edit_backtrack;
```

Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.

```
char *aligned_line[4];
```

Contains the two aligned structures after a call to one of the distance functions with edit_backtrack set to 1.

See also

utils.h, dist_vars.h and stringdist.h for more details

Functions for Tree Edit Distances

```
Tree   *make_tree (char *struc)
```

Constructs a Tree (essentially the postorder list) of the structure 'struc', for use in tree_edit_distance().

```
float   tree_edit_distance (Tree *T1,
                            Tree *T2)
```

Calculates the edit distance of the two trees.

```
void    free_tree(Tree *t)
```

Free the memory allocated for Tree t.

See also

dist_vars.h and treedist.h for prototypes and more detailed descriptions

Functions for String Alignment

```
swString *Make_swString (char *string)
```

Convert a structure into a format suitable for string_edit_distance().

```
float     string_edit_distance (swString *T1,
                                swString *T2)
```

Calculate the string edit distance of T1 and T2.

See also

dist_vars.h and stringdist.h for prototypes and more detailed descriptions

Functions for Comparison of Base Pair Probabilities

For comparison of base pair probability matrices, the matrices are first condensed into probability profiles which are the compared by alignment.

```
float *Make_bp_profile_bppm ( double *bppm,
                              int length)
```

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

```
float profile_edit_distance ( const float *T1,
                              const float *T2)
```

Align the 2 probability profiles T1, T2
.

See also

ProfileDist.h for prototypes and more details of the above functions

Next Page: Utilities

Chapter 3

Utilities - Odds and Ends

Table of Contents

- Producing secondary structure graphs
- Producing (colored) dot plots for base pair probabilities
- Producing (colored) alignments
- RNA sequence related utilities
- RNA secondary structure related utilities
- Miscellaneous Utilities

3.1 Producing secondary structure graphs

```
int PS_rna_plot ( char *string,
                  char *structure,
                  char *file)
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

```
int PS_rna_plot_a (
            char *string,
            char *structure,
            char *file,
            char *pre,
            char *post)
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

```
int gmlRNA (char *string,
            char *structure,
            char *ssfile,
            char option)
```

Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.

```
int ssv_rna_plot (char *string,
                  char *structure,
                  char *ssfile)
```

Produce a secondary structure graph in SStructView format.

```
int svg_rna_plot (char *string,
                  char *structure,
                  char *ssfile)
```

Produce a secondary structure plot in SVG format and write it to a file.

```
int xrna_plot ( char *string,
                char *structure,
                char *ssfile)
```

Produce a secondary structure plot for further editing in XRNA.

```
int rna_plot_type
```

Switch for changing the secondary structure layout algorithm.

Two low-level functions provide direct access to the graph lauyouting algorithms:

```
int simple_xy_coordinates ( short *pair_table,
                            float *X,
                            float *Y)
```

Calculate nucleotide coordinates for secondary structure plot the *Simple way*

```
int naview_xy_coordinates ( short *pair_table,
                            float *X,
                            float *Y)
```

See also

PS_dot.h and naview.h for more detailed descriptions.

3.2 Producing (colored) dot plots for base pair probabilities

```
int PS_color_dot_plot ( char *string,
                        cpair *pi,
                        char *filename)

int PS_color_dot_plot_turn (char *seq,
                            cpair *pi,
                            char *filename,
                            int winSize)

int PS_dot_plot_list (char *seq,
                      char *filename,
                      plist *pl,
                      plist *mf,
                      char *comment)
```

Produce a postscript dot-plot from two pair lists.

```
int PS_dot_plot_turn (char *seq,
                      struct plist *pl,
                      char *filename,
                      int winSize)
```

See also

PS_dot.h for more detailed descriptions.

3.3 Producing (colored) alignments

```
int PS_color_aln (
            const char *structure,
            const char *filename,
            const char *seqs[],
            const char *names[])
```

Produce PostScript sequence alignment color-annotated by consensus structure.

3.4 RNA sequence related utilities

Several functions provide useful applications to RNA sequences

```
char  *random_string (int l,
                      const char symbols[])
```

Create a random string using characters from a specified symbol set.

```
int   hamming ( const char *s1,
                const char *s2)
```

Calculate hamming distance between two sequences.

```
void str_DNA2RNA(char *sequence);
```

Convert a DNA input sequence to RNA alphabet.

```
void str_uppercase(char *sequence);
```

Convert an input sequence to uppercase.

3.5 RNA secondary structure related utilities

```
char *pack_structure (const char *struc)
```

Pack secondary secondary structure, 5:1 compression using base 3 encoding.

```
char *unpack_structure (const char *packed)
```

Unpack secondary structure previously packed with pack_structure()

```
short *make_pair_table (const char *structure)
```

Create a pair table of a secondary structure.

```
short *copy_pair_table (const short *pt)
```

Get an exact copy of a pair table.

3.6 Miscellaneous Utilities

```
void print_tty_input_seq (void)
```

Print a line to *stdout* that asks for an input sequence.

```
void print_tty_constraint_full (void)
```

Print structure constraint characters to stdout (full constraint support)

```
void print_tty_constraint (unsigned int option)
```

Print structure constraint characters to stdout. (constraint support is specified by option parameter)

```
int   *get_iindx (unsigned int length)
```

```
int   *get_indx (unsigned int length)
```

```
void constrain_ptypes (
                const char *constraint,
                unsigned int length,
                char *ptype,
                int *BP,
                int min_loop_size,
                unsigned int idx_type)
```

Insert constraining pair types according to constraint structure string.

```
char  *get_line(FILE *fp);
```

Read a line of arbitrary length from a stream.

```
unsigned int read_record(
                char **header,
                char **sequence,
                char ***rest,
                unsigned int options);
```

Get a data record from stdin.

```
char  *time_stamp (void)
```

Get a timestamp.

```
void warn_user (const char message[])
```

Print a warning message.

```
void nrerror (const char message[])
```

Die with an error message.

```
void   init_rand (void)
```

Make random number seeds.

```
unsigned short xsubi[3];
```

Current 48 bit random number.

```
double urn (void)
```

get a random number from [0..1]

```
int    int_urn (int from, int to)
```

Generates a pseudo random integer in a specified range.

```
void  *space (unsigned size)
```

Allocate space safely.

```
void  *xrealloc ( void *p,
                  unsigned size)
```

Reallocate space safely.

See also

utils.h for a complete overview and detailed description of the utility functions

Next Page: The new RNAlib API v3.0

Chapter 4

RNAlib API v3.0

4.1 Introduction

With version 2.2 we introduce the new API that will take over the old one in the future version 3.0. By then, backwards compatibility will be broken, and third party applications using RNAlib need to be ported. This switch of API became necessary, since many new features found their way into the RNAlib where a balance between threadsafety and easy-to-use library functions is hard or even impossible to establish. Furthermore, many old functions of the library are present as slightly modified copies of themself to provide a crude way to overload functions.

Therefore, we introduce the new v3.0 API very early in our development stage such that developers have enough time to migrate to the new functions and interfaces. We also started to provide encapsulation of the RNAlib functions, data structures, typedefs, and macros by prefixing them with *vrna_* and *VRNA_* , respectively. Header files should also be included using the *ViennaRNA/* namespace, e.g.

```
#include <ViennaRNA/fold.h>
```

instead of just using

```
#include <fold.h>
```

as required for RNAlib 1.x and 2.x.

This eases the work for programmers of third party applications that would otherwise need to put much effort into renaming functions and data types in their own implementations if their names appear in our library. Since we still provide backward compatibility up to the last version of RNAlib 2.x, this advantage may be fully exploited only starting from v3.0 which will be released in the future. However, our plan is to provide the possibility for an early switch-off mechanism of the backward compatibility in one of our next releases of ViennaRNA Package 2.x.

4.2 What are the major changes?

...

4.3 How to port your program to the new API

...

4.4 Some Examples using RNAlib API v3.0

Below are some example programs and code fragments that show the usage of the new API that is introduced with ViennaRNA version 2.2.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include  <ViennaRNA/data_structures.h>
#include  <ViennaRNA/params.h>
#include  <ViennaRNA/utils.h>
#include  <ViennaRNA/eval.h>
#include  <ViennaRNA/fold.h>
#include  <ViennaRNA/part_func.h>


int main(int argc, char *argv[]){

  char  *seq = "
      AGACGACAAGGUUGAAUCGCACCCACAGUCUAUGAGUCGGUGACAACAUUACGAAAGGCUGUAAAAUCAAUUAUUCACCACAGGGGGCCCCCGUGUCUAG";
  char  *mfe_structure = vrna_alloc(sizeof(char) * (strlen(seq) + 1));
  char  *prob_string   = vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* get a vrna_fold_compound with MFE and PF DP matrices and default model details */
  vrna_fold_compound_t *vc = vrna_fold_compound(seq, NULL,
      VRNA_OPTION_MFE | VRNA_OPTION_PF);

  /* call MFE function */
  double mfe = (double)vrna_mfe(vc, mfe_structure);

  printf("%s\n%s (%6.2f)\n", seq, mfe_structure, mfe);

  /* rescale parameters for Boltzmann factors */
  vrna_exp_params_rescale(vc, &mfe);

  /* call PF function */
  FLT_OR_DBL en = vrna_pf(vc, prob_string);

  /* print probability string and free energy of ensemble */
  printf("%s (%6.2f)\n", prob_string, en);

  /* compute centroid structure */
  double dist;
  char *cent = vrna_centroid(vc, &dist);

  /* print centroid structure, its free energy and mean distance to the ensemble */
  printf("%s (%6.2f d=%6.2f)\n", cent, vrna_eval_structure(vc, cent), dist);

  /* free centroid structure */
  free(cent);

  /* free pseudo dot-bracket probability string */
  free(prob_string);

  /* free mfe structure */
  free(mfe_structure);

  /* free memory occupied by vrna_fold_compound */
  vrna_fold_compound_free(vc);

  return EXIT_SUCCESS;
}
```

Chapter 5

Scripting Language interface(s)

Notes on functions and structures wrapped to the scripting language interface(s)

5.1 Introduction

For an easy integration into scripting languages, we provide an automatically generated interface to the RNAlib C-library, generated with scripting.

5.1.1 Function renaming scheme

The main difference when using a scripting language interface compared to direct calls of RNAlib C functions is, that the prefix 'vrna_' is dropped. For instance, when calling the vrna_fold() function, corresponding calls in perl or python are RNA::fold(), and RNA.fold(), respectively.

Functions that are dedicated to work on specific data structures only, e.g. the vrna_fold_compound_t, are usually not exported at all. Instead, they are attached as object methods of a corresponding class (see Object oriented Interface for data structures for detailed information).

5.1.2 Object oriented Interface for data structures

For data structures, typedefs, and enumerations the 'vrna_' prefixes are dropped as well, together with their suffixes '_s', '_t', and '_e', respectively. Furthermore, data structures are usually transformed into classes and relevant functions of the C-library are attached as methods.

5.2 Examples

5.2.1 Perl Examples

5.2.1.1 Using the Flat Interface

Example 1: "Simple MFE prediction"

```
00001 #!/usr/bin/perl
00002
00003 use warnings;
00004 use strict;
00005
00006 use RNA;
00007
00008 my $seq1 = "CGCAGGGAUACCCGCG";
00009
00010 # compute minimum free energy (mfe) and corresponding structure
00011 my ($ss, $mfe) = RNA::fold($seq1);
00012
00013 # print output
00014 printf "%s [ %6.2f ]\n", $ss, $mfe;
```

5.2.1.2 Using the Object Oriented (OO) Interface

The 'fold_compound' class that serves as an object oriented interface for vrna_fold_compound_t

Example 1: "Simple MFE prediction"

```
00001 #!/usr/bin/perl
00002
00003 use warnings;
00004 use strict;
00005
00006 use RNA;
00007
00008 my $seq1 = "CGCAGGGAUACCCGCG";
00009
00010 # create new fold_compound object
00011 my $fc = new RNA::fold_compound($seq1);
00012
00013 # compute minimum free energy (mfe) and corresponding structure
00014 my ($ss, $mfe) = $fc->mfe();
00015
00016 # print output
00017 printf "%s [ %6.2f ]\n", $ss, $mfe;
```

5.2.2 Python Examples

5.3 How functions, structures, enums, and macro definitions are wrapped

Global vrna_exp_params_rescale (vrna_fold_compound_t *vc, double *mfe)

This function is attached to vrna_fc_s objects as overloaded **exp_params_rescale()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_rescale(), i.e. default scaling of the partition function. Passing an energy in kcal/mol, e.g. as retrieved by a previous call to the *mfe()* method, instructs all subsequent calls to scale the partition function accordingly.

Global vrna_exp_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)

This function is attached to vrna_fc_s objects as overloaded **exp_params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

Global vrna_exp_params_subst (vrna_fold_compound_t *vc, vrna_exp_param_t *params)

This function is attached to vrna_fc_s objects as **exp_params_subst()** method.

Class vrna_fc_s

This data structure is wrapped as an object **fold_compound** with several related functions attached as methods.

A new **fold_compound** can be obtained by calling one of its constructors:

- *fold_compound(seq)* – Initialize with a single sequence, or two concatenated sequences separated by an ampersand character '&' (for cofolding)
- *fold_compound(aln)* – Initialize with a sequence alignment *aln* stored as a list of sequences (with gap characters)

The resulting object has a list of attached methods which in most cases directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *type()* – Get the type of the *fold_compound* (See vrna_fc_type_e)
- *length()* – Get the length of the sequence(s) or alignment stored within the *fold_compound*

Class vrna_md_s

This data structure is wrapped as an object **md** with multiple related functions attached as methods.

A new set of default parameters can be obtained by calling the constructure of **md:**

- *md()* – Initialize with default settings

The resulting object has a list of attached methods which directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *reset()* – vrna_md_set_default()
- *set_from_globals()* – set_model_details()
- *option_string()* – vrna_md_option_string()

Note, that default parameters can be modified by directly setting any of the following global variables. Internally, getting/setting default parameters using their global variable representative translates into calls of the following functions, therefore these wrappers for these functions do not exist in the scripting language interface(s):

global variable	*C* getter	*C* setter
temperature	vrna_md_defaults_temperature_get()	vrna_md_defaults_temperature()
dangles	vrna_md_defaults_dangles_get()	vrna_md_defaults_dangles()
betaScale	vrna_md_defaults_betaScale_get()	vrna_md_defaults_betaScale()
tetra_loop	this is an alias of *special_hp*	
special_hp	vrna_md_defaults_special_hp_get()	vrna_md_defaults_special_hp()
noLonelyPairs	this is an alias of *noLP*	
noLP	vrna_md_defaults_noLP_get()	vrna_md_defaults_noLP()
noGU	vrna_md_defaults_noGU_get()	vrna_md_defaults_noGU()
no_closingGU	this is an alias of *noGUclosure*	
noGUclosure	vrna_md_defaults_noGUclosure_get()	vrna_md_defaults_noGUclosure()

global variable	*C* getter	*C* setter
logML	vrna_md_defaults_logML_get()	vrna_md_defaults_logML()
circ	vrna_md_defaults_circ_get()	vrna_md_defaults_circ()
gquad	vrna_md_defaults_gquad_get()	vrna_md_defaults_gquad()
uniq_ML	vrna_md_defaults_uniq_ML_get()	vrna_md_defaults_uniq_ML()
energy_set	vrna_md_defaults_energy_set_get()	vrna_md_defaults_energy_set()
backtrack	vrna_md_defaults_backtrack_get()	vrna_md_defaults_backtrack()
backtrack_type	vrna_md_defaults_backtrack_type_get()	vrna_md_defaults_backtrack_type()
do_backtrack	this is an alias of *compute_bpp*	
compute_bpp	vrna_md_defaults_compute_bpp_get()	vrna_md_defaults_compute_bpp()
max_bp_span	vrna_md_defaults_max_bp_span_get()	vrna_md_defaults_max_bp_span()
min_loop_size	vrna_md_defaults_min_loop_size_get()	vrna_md_defaults_min_loop_size()
window_size	vrna_md_defaults_window_size_get()	vrna_md_defaults_window_size()
oldAliEn	vrna_md_defaults_oldAliEn_get()	vrna_md_defaults_oldAliEn()
ribo	vrna_md_defaults_ribo_get()	vrna_md_defaults_ribo()
cv_fact	vrna_md_defaults_cv_fact_get()	vrna_md_defaults_cv_fact()
nc_fact	vrna_md_defaults_nc_fact_get()	vrna_md_defaults_nc_fact()
sfact	vrna_md_defaults_sfact_get()	vrna_md_defaults_sfact()

Global vrna_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)

This function is attached to vrna_fc_s objects as overloaded **params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

Global vrna_params_subst (vrna_fold_compound_t *vc, vrna_param_t *par)

This function is attached to vrna_fc_s objects as **params_subst()** method.

Chapter 6

Input / Output File Formats

6.1 File formats for Secondary Structure Constraints

6.1.1 Constraints Definition File

The RNAlib can parse and apply data from constraint definition text files, where each constraint is given as a line of whitespace delimited commands. The syntax we use extends the one used in `mfold` / `UNAfold` where each line begins with a command character followed by a set of positions.
Additionally, we introduce several new commands, and allow for an optional loop type context specifier in form of a sequence of characters, and an orientation flag that enables one to force a nucleotide to pair upstream, or downstream.

6.1.1.1 Constraint commands

The following set of commands is recognized:

- `F` . . . Force
- `P` . . . Prohibit
- `C` . . . Conflicts/Context dependency
- `A` . . . Allow (for non-canonical pairs)
- `E` . . . Soft constraints for unpaired position(s), or base pair(s)

6.1.1.2 Specification of the loop type context

The optional loop type context specifier [WHERE] may be a combination of the following:

- `E` . . . Exterior loop
- `H` . . . Hairpin loop
- `I` . . . Interior loop (enclosing pair)
- `i` . . . Interior loop (enclosed pair)
- `M` . . . Multibranch loop (enclosing pair)
- `m` . . . Multibranch loop (enclosed pair)
- `A` . . . All loops

If no [WHERE] flags are set, all contexts are considered (equivalent to `A`)

6.1.1.3 Controlling the orientation of base pairing

For particular nucleotides that are forced to pair, the following [ORIENTATION] flags may be used:

- `U` . . . Upstream
- `D` . . . Downstream

If no [ORIENTATION] flag is set, both directions are considered.

6.1.1.4 Sequence coordinates

Sequence positions of nucleotides/base pairs are $1-$ based and consist of three positions i, j, and k. Alternativly, four positions may be provided as a pair of two position ranges $[i:j]$, and $[k:l]$ using the '-' sign as delimiter within each range, i.e. $i-j$, and $k-l$.

6.1.1.5 Valid constraint commands

Below are resulting general cases that are considered *valid* constraints:

1. **"Forcing a range of nucleotide positions to be paired"**:
   Syntax:

   ```
   F i 0 k [WHERE] [ORIENTATION]
   ```

   Description:
   Enforces the set of k consecutive nucleotides starting at position i to be paired. The optional loop type specifier [WHERE] allows to force them to appear as closing/enclosed pairs of certain types of loops.
2. **"Forcing a set of consecutive base pairs to form"**:
   Syntax:

   ```
   F i j k [WHERE]
   ```

   Description:
   Enforces the base pairs $(i,j),\ldots,(i+(k-1),j-(k-1))$ to form. The optional loop type specifier [WHERE] allows to specify in which loop context the base pair must appear.
3. **"Prohibiting a range of nucleotide positions to be paired"**:
   Syntax:

   ```
   P i 0 k [WHERE]
   ```

   Description:
   Prohibit a set of k consecutive nucleotides to participate in base pairing, i.e. make these positions unpaired. The optional loop type specifier [WHERE] allows to force the nucleotides to appear within the loop of specific types.
4. **"Probibiting a set of consecutive base pairs to form"**:
   Syntax:

   ```
   P i j k [WHERE]
   ```

Description:
Probibit the base pairs $(i,j),\ldots,(i+(k-1),j-(k-1))$ to form. The optional loop type specifier [WHERE] allows to specify the type of loop they are disallowed to be the closing or an enclosed pair of.

5. **"Prohibiting two ranges of nucleotides to pair with each other"**:
Syntax:

```
P i-j k-l [WHERE]
```

Description:
Prohibit any nucleotide $p \in [i:j]$ to pair with any other nucleotide $q \in [k:l]$. The optional loop type specifier [WHERE] allows to specify the type of loop they are disallowed to be the closing or an enclosed pair of.

6. **"Enforce a loop context for a range of nucleotide positions"**:
Syntax:

```
C i 0 k [WHERE]
```

Description:
This command enforces nucleotides to be unpaired similar to *prohibiting* nucleotides to be paired, as described above. It too marks the corresponding nucleotides to be unpaired, however, the [WHERE] flag can be used to enforce specfic loop types the nucleotides must appear in.

7. **"Remove pairs that conflict with a set of consecutive base pairs"**:
Syntax:

```
C i j k
```

Description:
Remove all base pairs that conflict with a set of consecutive base pairs $(i,j),\ldots,(i+(k-1),j-(k-1))$. Two base pairs (i,j) and (p,q) conflict with each other if $i<p<j<q$, or $p<i<q<j$.

8. **"Allow a set of consecutive (non-canonical) base pairs to form"**:
Syntax:

```
A i j k [WHERE]
```

Description:
This command enables the formation of the consecutive base pairs $(i,j),\ldots,(i+(k-1),j-(k-1))$, no matter if they are *canonical*, or *non-canonical*. In contrast to the above `F` and `W` commands, which remove conflicting base pairs, the `A` command does not. Therefore, it may be used to allow *non-canoncial* base pair interactions. Since the RNAlib does not contain free energy contributions E_{ij} for non-canonical base pairs (i,j), they are scored as the *maximum* of similar, known contributions. In terms of a *Nussinov* like scoring function the free energy of non-canonical base pairs is therefore estimated as

$$E_{ij} = \min\left[\max_{(i,k)\in\{GC,CG,AU,UA,GU,UG\}} E_{ik}, \max_{(k,j)\in\{GC,CG,AU,UA,GU,UG\}} E_{kj}\right].$$

The optional loop type specifier [WHERE] allows to specify in which loop context the base pair may appear.

9. **"Apply pseudo free energy to a range of unpaired nucleotide positions"**:
Syntax:

```
E i 0 k e
```

Description:
Use this command to apply a pseudo free energy of e to the set of k consecutive nucleotides, starting at position i. The pseudo free energy is applied only if these nucleotides are considered unpaired in the recursions, or evaluations, and is expected to be given in $kcal/mol$.

10. **"Apply pseudo free energy to a set of consecutive base pairs"**:
    Syntax

    ```
    E i j k e
    ```

    Use this command to apply a pseudo free energy of e to the set of base pairs $(i, j), \ldots, (i + (k - 1), j - (k - 1))$. Energies are expected to be given in $kcal/mol$.

Chapter 7

Example - A Small Example Program

The following program exercises most commonly used functions of the library. The program folds two sequences using both the mfe and partition function algorithms and calculates the tree edit and profile distance of the resulting structures and base pairing probabilities.

Note

This program uses the old API of RNAlib, which is in part already marked deprecated. Please consult the RNAlib API v3.0 page for details of what changes are necessary to port your implementation to the new API.

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <math.h>
#include  <string.h>
#include  "utils.h"
#include  "fold_vars.h"
#include  "fold.h"
#include  "part_func.h"
#include  "inverse.h"
#include  "RNAstruct.h"
#include  "treedist.h"
#include  "stringdist.h"
#include  "profiledist.h"

void main()
{
   char *seq1="CGCAGGGAUACCCGCG", *seq2="GCGCCCAUAGGGACGC",
        *struct1,* struct2,* xstruc;
   float e1, e2, tree_dist, string_dist, profile_dist, kT;
   Tree *T1, *T2;
   swString *S1, *S2;
   float *pf1, *pf2;
   FLT_OR_DBL *bppm;
   /* fold at 30C instead of the default 37C */
   temperature = 30.;      /* must be set *before* initializing  */

   /* allocate memory for structure and fold */
   struct1 = (char* ) space(sizeof(char)*(strlen(seq1)+1));
   e1 =  fold(seq1, struct1);

   struct2 = (char* ) space(sizeof(char)*(strlen(seq2)+1));
   e2 =  fold(seq2, struct2);

   free_arrays();     /* free arrays used in fold() */

   /* produce tree and string representations for comparison */
   xstruc = expand_Full(struct1);
   T1 = make_tree(xstruc);
   S1 = Make_swString(xstruc);
   free(xstruc);

   xstruc = expand_Full(struct2);
   T2 = make_tree(xstruc);
   S2 = Make_swString(xstruc);
   free(xstruc);

   /* calculate tree edit distance and aligned structures with gaps */
```

```
  edit_backtrack = 1;
  tree_dist = tree_edit_distance(T1, T2);
  free_tree(T1); free_tree(T2);
  unexpand_aligned_F(aligned_line);
  printf("%s\n%s  %3.2f\n", aligned_line[0], aligned_line[1], tree_dist);

  /* same thing using string edit (alignment) distance */
  string_dist = string_edit_distance(S1, S2);
  free(S1); free(S2);
  printf("%s  mfe=%5.2f\n%s  mfe=%5.2f  dist=%3.2f\n",
         aligned_line[0], e1, aligned_line[1], e2, string_dist);

  /* for longer sequences one should also set a scaling factor for
     partition function folding, e.g: */
  kT = (temperature+273.15)*1.98717/1000.;  /* kT in kcal/mol */
  pf_scale = exp(-e1/kT/strlen(seq1));

  /* calculate partition function and base pair probabilities */
  e1 = pf_fold(seq1, struct1);
  /* get the base pair probability matrix for the previous run of pf_fold() */
  bppm = export_bppm();
  pf1 = Make_bp_profile_bppm(bppm, strlen(seq1));

  e2 = pf_fold(seq2, struct2);
  /* get the base pair probability matrix for the previous run of pf_fold() */
  bppm = export_bppm();
  pf2 = Make_bp_profile_bppm(bppm, strlen(seq2));

  free_pf_arrays();  /* free space allocated for pf_fold() */

  profile_dist = profile_edit_distance(pf1, pf2);
  printf("%s  free energy=%5.2f\n%s  free energy=%5.2f  dist=%3.2f\n",
         aligned_line[0], e1, aligned_line[1], e2, profile_dist);

  free_profile(pf1); free_profile(pf2);
}
```

In a typical Unix environment you would compile this program using:

```
cc ${OPENMP_CFLAGS} -c example.c -I${hpath}
```

and link using

```
cc ${OPENMP_CFLAGS} -o example -L${lpath} -lRNA -lm
```

where *${hpath}* and *${lpath}* point to the location of the header files and library, respectively.

Note

As default, the RNAlib is compiled with build-in *OpenMP* multithreading support. Thus, when linking your own object files to the library you have to pass the compiler specific *${OPENMP_CFLAGS}* (e.g. '-fopenmp' for **gcc**) even if your code does not use openmp specific code. However, in that case the *OpenMP* flags may be ommited when compiling example.c

Chapter 8

Deprecated List

Global alifold (const char **strings, char *structure)

Usage of this function is discouraged! Use vrna_alifold(), or vrna_mfe() instead!

Global alimake_pair_table (const char *structure)

Use vrna_pt_ali_get() instead!

Global alipbacktrack (double *prob)

Use vrna_pbacktrack() instead!

Global alipf_circ_fold (const char **sequences, char *structure, vrna_plist_t **pl)

Use vrna_pf() instead

Global alipf_fold (const char **sequences, char *structure, vrna_plist_t **pl)

Use vrna_pf() instead

Global alipf_fold_par (const char **sequences, char *structure, vrna_plist_t **pl, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained, int is_circular)

Use vrna_pf() instead

Global assign_plist_from_db (vrna_plist_t **pl, const char *struc, float pr)

Use vrna_plist() instead

Global assign_plist_from_pr (vrna_plist_t **pl, FLT_OR_DBL *probs, int length, double cutoff)

Use vrna_plist_from_probs() instead!

Global base_pair

Do not use this variable anymore!

Global bondT

Use vrna_bp_stack_t instead!

Global bp_distance (const char *str1, const char *str2)

Use vrna_bp_distance instead

Global bppm_symbol (const float *x)

Use vrna_bpp_symbol() instead!

Global bppm_to_structure (char *structure, FLT_OR_DBL *pr, unsigned int length)

Use vrna_db_from_probs() instead!

Global centroid (int length, double *dist)

This function is deprecated and should not be used anymore as it is not threadsafe!

Global circalifold (const char **strings, char *structure)

Usage of this function is discouraged! Use vrna_alicircfold(), and vrna_mfe() instead!

Global circfold (const char *sequence, char *structure)

Use vrna_circfold(), or vrna_mfe() instead!

Global co_pf_fold (char *sequence, char *structure)

{Use vrna_pf_dimer() instead!}

Global co_pf_fold_par (char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained)

Use vrna_pf_dimer() instead!

Global cofold (const char *sequence, char *structure)

use vrna_mfe_dimer() instead

Global cofold_par (const char *string, char *structure, vrna_param_t *parameters, int is_constrained)

use vrna_mfe_dimer() instead

Global compute_BPdifferences (short *pt1, short *pt2, unsigned int turn)

Use vrna_refBPdist_matrix() instead

Global compute_probabilities (double FAB, double FEA, double FEB, vrna_plist_t *prAB, vrna_plist_t *prA, vrna_plist_t *prB, int Alength)

{ Use vrna_pf_dimer_probs() instead!}

Global constrain_ptypes (const char *constraint, unsigned int length, char *ptype, int *BP, int min_loop_size, unsigned int idx_type)

Do not use this function anymore! Structure constraints are now handled through vrna_hc_t and related functions.

Global copy_pair_table (const short *pt)

Use vrna_ptable_copy() instead

Global cpair

Use vrna_cpair_t instead!

Global cv_fact

See vrna_md_t.cv_fact, and vrna_mfe() to avoid using global variables

Global destroy_TwoDfold_variables (TwoDfold_vars *our_variables)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Global destroy_TwoDpfold_variables (TwoDpfold_vars *vars)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

Global energy_of_alistruct (const char **sequences, const char *structure, int n_seq, float *energy)

Usage of this function is discouraged! Use vrna_eval_structure(), and vrna_eval_covar_structure() instead!

Global energy_of_circ_struct (const char *string, const char *structure)

This function is deprecated and should not be used in future programs Use energy_of_circ_structure() instead!

Global energy_of_circ_struct_par (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)

Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

Global energy_of_circ_structure (const char *string, const char *structure, int verbosity_level)

Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

Global energy_of_move (const char *string, const char *structure, int m1, int m2)

Use vrna_eval_move() instead!

Global energy_of_move_pt (short *pt, short *s, short *s1, int m1, int m2)

Use vrna_eval_move_pt() instead!

Global energy_of_struct (const char *string, const char *structure)

This function is deprecated and should not be used in future programs! Use energy_of_structure() instead!

Global energy_of_struct_par (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)

Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

Global energy_of_struct_pt (const char *string, short *ptable, short *s, short *s1)

This function is deprecated and should not be used in future programs! Use energy_of_structure_pt() instead!

Global energy_of_struct_pt_par (const char *string, short *ptable, short *s, short *s1, vrna_param_t *parameters, int verbosity_level)

Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

Global energy_of_structure (const char *string, const char *structure, int verbosity_level)

Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

Global energy_of_structure_pt (const char *string, short *ptable, short *s, short *s1, int verbosity_level)

Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

Global expHairpinEnergy (int u, int type, short si1, short sj1, const char *string)

Use exp_E_Hairpin() from loop_energies.h instead

Global expLoopEnergy (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)

Use exp_E_IntLoop() from loop_energies.h instead

Global export_ali_bppm (void)

Usage of this function is discouraged! The new vrna_fold_compound_t allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to vrna_pf(), or any of the old API calls for consensus structure partition function folding.

Global export_circfold_arrays (int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

Global export_circfold_arrays_par (int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p)

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

Global export_co_bppm (void)

This function is deprecated and will be removed soon! The base pair probability array is available through the vrna_fold_compound_t data structure, and its associated vrna_mx_pf_t member.

Global export_cofold_arrays (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **indx_p, char **ptype_p)

folding matrices now reside within the vrna_fold_compound_t. Thus, this function will only work in conjunction with a prior call to the deprecated functions cofold() or cofold_par()

Global export_cofold_arrays_gq (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **ggg_p, int **indx_p, char **ptype_p)

folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to cofold() or cofold_par()

Global export_fold_arrays (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

Global export_fold_arrays_par (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p)

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

Global filecopy (FILE *from, FILE *to)

Use vrna_file_copy() instead!

Global fold (const char *sequence, char *structure)

use vrna_fold(), or vrna_mfe() instead!

Global fold_par (const char *sequence, char *structure, vrna_param_t *parameters, int is_constrained, int is_circular)

use vrna_mfe() instead!

Global free_alifold_arrays (void)

Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced vrna_fold_compound_t is handled by vrna_vrna_fold_compound_free()

Global free_alipf_arrays (void)

Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with vrna_) will be not affected!

Global free_arrays (void)

See vrna_fold(), vrna_circfold(), or vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

Global free_co_arrays (void)

This function will only free memory allocated by a prior call of cofold() or cofold_par(). See vrna_mfe_dimer() for how to use the new API

Global free_co_pf_arrays (void)

This function will be removed for the new API soon! See vrna_pf_dimer(), vrna_fold_compound(), and vrna_↩ fold_compound_free() for an alternative

Global free_pf_arrays (void)

See vrna_fold_compound_t and its related functions for how to free memory occupied by the dynamic programming matrices

Global get_alipf_arrays (short ***S_p, short ***S5_p, short ***S3_p, unsigned short ***a2s_p, char *↩ Ss_p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm_p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p, short **pscore)**

It is discouraged to use this function! The new vrna_fold_compound_t allows direct access to all necessary consensus structure prediction related variables!

Global get_boltzmann_factor_copy (vrna_exp_param_t *parameters)

Use vrna_exp_params_copy() instead!

Global get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)

Use vrna_exp_params() instead!

Global get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)

Use vrna_exp_params_comparative() instead!

Global get_centroid_struct_gquad_pr (int length, double *dist)

This function is deprecated and should not be used anymore as it is not threadsafe!

Global get_centroid_struct_pl (int length, double *dist, vrna_plist_t *pl)

This function was renamed to vrna_centroid_from_plist()

Global get_centroid_struct_pr (int length, double *dist, FLT_OR_DBL *pr)

This function was renamed to vrna_centroid_from_probs()

Global get_concentrations (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double *startconc)

{ Use vrna_pf_dimer_concentrations() instead!}

Global get_monomere_mfes (float *e1, float *e2)

{This function is obsolete and will be removed soon!}

Global get_mpi (char *Alseq[], int n_seq, int length, int *mini)

Use vrna_aln_mpi() as a replacement

Global get_plist (vrna_plist_t *pl, int length, double cut_off)

{ This function is deprecated and will be removed soon!} use assign_plist_from_pr() instead!

Global get_scaled_alipf_parameters (unsigned int n_seq)

Use vrna_exp_params_comparative() instead!

Global get_scaled_parameters (double temperature, vrna_md_t md)

Use vrna_params() instead!

Global get_scaled_pf_parameters (void)

Use vrna_exp_params() instead!

Global get_TwoDfold_variables (const char *seq, const char *structure1, const char *structure2, int circ)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩ _TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Global get_TwoDpfold_variables (const char *seq, const char *structure1, char *structure2, int circ)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩ _TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

Global HairpinE (int size, int type, int si1, int sj1, const char *string)

{This function is deprecated and will be removed soon. Use E_Hairpin() instead!}

Global hamming (const char *s1, const char *s2)

Use vrna_hamming_distance() instead!

Global hamming_bound (const char *s1, const char *s2, int n)

Use vrna_hamming_distance_bound() instead!

Global iindx

Do not use this variable anymore!

Global init_co_pf_fold (int length)

{ This function is deprecated and will be removed soon!}

Global init_pf_fold (int length)

This function is obsolete and will be removed soon!

Global init_rand (void)

Use vrna_init_rand() instead!

Global initialize_cofold (int length)

{This function is obsolete and will be removed soon!}

Global initialize_fold (int length)

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

Global int_urn (int from, int to)

Use vrna_int_urn() instead!

Global Lfold (const char *string, char *structure, int maxdist)

Use vrna_mfe_window() instead!

Global Lfoldz (const char *string, char *structure, int maxdist, int zsc, double min_z)

Use vrna_mfe_window_zscore() instead!

Global loop_energy (short *ptable, short *s, short *s1, int i)

Use vrna_eval_loop_pt() instead!

Global LoopEnergy (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1)

{This function is deprecated and will be removed soon. Use E_IntLoop() instead!}

Global Make_bp_profile (int length)

This function is deprecated and will be removed soon! See Make_bp_profile_bppm() for a replacement

Global make_pair_table (const char *structure)

Use vrna_ptable() instead

Global make_pair_table_snoop (const char *structure)

Use vrna_pt_snoop_get() instead!

Global make_referenceBP_array (short *reference_pt, unsigned int turn)

Use vrna_refBPcnt_matrix() instead

Global mean_bp_dist (int length)

This function is not threadsafe and should not be used anymore. Use mean_bp_distance() instead!

Global mean_bp_distance (int length)

Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

Global mean_bp_distance_pr (int length, FLT_OR_DBL *pr)

Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

Global nc_fact

See vrna_md_t.nc_fact, and vrna_mfe() to avoid using global variables

Global nrerror (const char message[])

Use vrna_message_error() instead!

Global pack_structure (const char *struc)

Use vrna_db_pack() as a replacement

Global PAIR

Use vrna_basepair_t instead!

Global pair_info

Use vrna_pinfo_t instead!

Global paramT

Use vrna_param_t instead!

Global parenthesis_structure (char *structure, vrna_bp_stack_t *bp, int length)

use vrna_parenthesis_structure() instead

Global parenthesis_zuker (char *structure, vrna_bp_stack_t *bp, int length)

use vrna_parenthesis_zuker instead

Global path_t

Use vrna_path_t instead!

Global pbacktrack_circ (char *sequence)

Use vrna_pbacktrack() instead.

Global pf_circ_fold (const char *sequence, char *structure)

Use vrna_pf() instead!

Global pf_fold_par (const char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained, int is_circular)

Use vrna_pf() instead

Global pf_paramT

Use vrna_exp_param_t instead!

Global plist

Use vrna_plist_t instead!

Global pr

Do not use this variable anymore!

Global print_tty_constraint (unsigned int option)

Use vrna_message_constraints() instead!

Global print_tty_constraint_full (void)

Use vrna_message_constraint_options_all() instead!

Global print_tty_input_seq (void)

Use vrna_message_input_seq_simple() instead!

Global print_tty_input_seq_str (const char *s)

Use vrna_message_input_seq() instead!

Global PS_dot_plot (char *string, char *file)

This function is deprecated and will be removed soon! Use PS_dot_plot_list() instead!

Global PS_rna_plot (char *string, char *structure, char *file)

Use vrna_file_PS_rnaplot() instead!

Global PS_rna_plot_a (char *string, char *structure, char *file, char *pre, char *post)

Use vrna_file_PS_rnaplot_a() instead!

Global PS_rna_plot_a_gquad (char *string, char *structure, char *ssfile, char *pre, char *post)

Use vrna_file_PS_rnaplot_a() instead!

Global random_string (int l, const char symbols[])

Use vrna_random_string() instead!

Global read_record (char **header, char **sequence, char ***rest, unsigned int options)

This function is deprecated! Use vrna_file_fasta_read_record() as a replacment.

Global scale_parameters (void)

Use vrna_params() instead!

Global sect

Use vrna_sect_t instead!

Global set_model_details (vrna_md_t *md)

This function will vanish as soon as backward compatibility of RNAlib is dropped (expected in version 3). Use vrna_md_set_default() instead!

Global SOLUTION

Use vrna_subopt_solution_t instead!

Global space (unsigned size)

Use vrna_alloc() instead!

Global st_back

set the *uniq_ML* flag in vrna_md_t before passing it to vrna_fold_compound().

Global stackProb (double cutoff)

Use vrna_stack_prob() instead!

Global str_DNA2RNA (char *sequence)

Use vrna_seq_toRNA() instead!

Global str_uppercase (char *sequence)

Use vrna_seq_toupper() instead!

Global temperature

Use vrna_md_defaults_temperature(), and vrna_md_defaults_temperature_get() to change, and read the global default temperature settings

Global time_stamp (void)

Use vrna_time_stamp() instead!

Global TwoDfold_backtrack_f5 (unsigned int j, int k, int l, TwoDfold_vars *vars)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩
_TwoD(), vrna_mfe_TwoD(), vrna_backtrack5_TwoD(), and vrna_fold_compound_free() instead!

Global TwoDfold_vars

This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Global TwoDfoldList (TwoDfold_vars *vars, int distance1, int distance2)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Global TwoDpfold_pbacktrack (TwoDpfold_vars *vars, int d1, int d2)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack_TwoD(), and vrna_fold_compound_free() instead!

Global TwoDpfold_pbacktrack5 (TwoDpfold_vars *vars, int d1, int d2, unsigned int length)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack5_TwoD(), and vrna_fold_compound_free() instead!

Class TwoDpfold_vars

This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

Global TwoDpfoldList (TwoDpfold_vars *vars, int maxDistance1, int maxDistance2)

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

Global unpack_structure (const char *packed)

Use vrna_db_unpack() as a replacement

Global update_alifold_params (void)

Usage of this function is discouraged! The new API uses vrna_fold_compound_t to lump all folding related necessities together, including the energy parameters. Use vrna_update_fold_params() to update the energy parameters within a vrna_fold_compound_t.

Global update_co_pf_params (int length)

Use vrna_exp_params_subst() instead!

Global update_co_pf_params_par (int length, vrna_exp_param_t *parameters)

Use vrna_exp_params_subst() instead!

Global update_cofold_params (void)

See vrna_params_subst() for an alternative using the new API

Global update_cofold_params_par (vrna_param_t *parameters)

See vrna_params_subst() for an alternative using the new API

Global update_fold_params (void)

For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

Global update_fold_params_par (vrna_param_t *parameters)

For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

Global update_pf_params (int length)

Use vrna_exp_params_subst() instead

Global update_pf_params_par (int length, vrna_exp_param_t *parameters)

Use vrna_exp_params_subst() instead

Global urn (void)

Use vrna_urn() instead!

Global VRNA_CONSTRAINT_FILE

Use 0 instead!

Global VRNA_CONSTRAINT_MULTILINE

see vrna_extract_record_rest_structure()

Global VRNA_CONSTRAINT_NO_HEADER

This mode is not supported anymore!

Global VRNA_CONSTRAINT_SOFT_MFE

This flag has no meaning anymore, since constraints are now always stored!

Global VRNA_CONSTRAINT_SOFT_PF

Use VRNA_OPTION_PF instead!

Global vrna_exp_param_s::id

This attribute will be removed in version 3

Global vrna_extract_record_rest_constraint (char ∗∗cstruc, const char ∗∗lines, unsigned int option)

Use vrna_extract_record_rest_structure() instead!

Global vrna_fc_s::pscore_pf_compat

This attribute will vanish in the future!

Global vrna_fc_s::ptype_pf_compat

This attribute will vanish in the future! It's meant for backward compatibility only!

Global warn_user (const char message[])

Use vrna_message_warning() instead!

Global xrealloc (void ∗p, unsigned size)

Use vrna_realloc() instead!

Global zukersubopt (const char ∗string)

use vrna_zukersubopt() instead

Global zukersubopt_par (const char ∗string, vrna_param_t ∗parameters)

use vrna_zukersubopt() instead

Chapter 9

Bug List

Global vrna_subopt_zuker (vrna_fold_compound_t *vc)

Due to resizing, any pre-existing constraints will be lost!

Chapter 10

Module Index

10.1 Modules

Here is a list of all modules:

RNA Secondary Structure Prediction . . . 45
Computing Minimum Free Energy (MFE) Structures . . . 133
MFE Structures of single Nucleic Acid Sequences . . . 160
MFE Structures of two hybridized Sequences . . . 166
MFE Consensus Structures for Sequence Alignment(s) . . . 185
Local MFE structure Prediction and Z-scores . . . 196
Calculating MFE representatives of a Distance Based Partitioning . . . 214
Computing Partition Functions and Pair Probabilities . . . 135
Compute the structure with maximum expected accuracy (MEA) . . . 147
Compute the centroid structure . . . 148
Partition Function for two hybridized Sequences . . . 172
Partition Function for two hybridized Sequences as a stepwise Process . . . 176
Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . 190
Partition functions for locally stable secondary structures . . . 201
Calculate Partition Functions of a Distance Based Partitioning . . . 221
Enumerating Suboptimal Structures . . . 150
Suboptimal structures according to Zuker et al. 1989 . . . 151
Suboptimal structures within an energy band arround the MFE . . . 153
Stochastic backtracking in the Ensemble . . . 156
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) . . . 194
Stochastic Backtracking of Structures from Distance Based Partitioning . . . 224
Calculate Secondary Structures of two RNAs upon Dimerization . . . 159
MFE Structures of two hybridized Sequences . . . 166
Partition Function for two hybridized Sequences . . . 172
Partition Function for two hybridized Sequences as a stepwise Process . . . 176
Predicting Consensus Structures from Alignment(s) . . . 179
MFE Consensus Structures for Sequence Alignment(s) . . . 185
Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . 190
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) . . . 194
Local MFE consensus structures for Sequence Alignments . . . 204
Predicting Locally stable structures of large sequences . . . 195
Local MFE structure Prediction and Z-scores . . . 196
Partition functions for locally stable secondary structures . . . 201
Local MFE consensus structures for Sequence Alignments . . . 204

Classified Dynamic Programming . . . 212
Distance based partitioning of the Secondary Structure Space . . . 213
Calculating MFE representatives of a Distance Based Partitioning . . . 214
Calculate Partition Functions of a Distance Based Partitioning . . . 221
Stochastic Backtracking of Structures from Distance Based Partitioning . . . 224
Compute the Density of States . . . 226
Inverse Secondary Structure Prediction . . . 47
Refolding paths between secondary structues . . . 49
Direct refolding paths between two secondary structures . . . 277
Free Energy Evaluation for given Sequence / Structure Pairs . . . 50
Processing and Evaluating Decomposed Loops . . . 51
Energy Parameter Sets and Boltzmann Factors . . . 62
Reading/Writing Energy Parameter Sets from/to File . . . 205
Converting Energy Parameter Files . . . 207
Manipulation of the Prediction Models . . . 74
Constraining the Secondary Structure Predictions and Evaluations . . . 105
Hard Constraints . . . 227
Soft Constraints . . . 237
Incorporating SHAPE reactivity data . . . 245
Incorporating ligands binding to specific sequence/structure motifs . . . 249
Generate soft constraints from data . . . 251
Data Structures and Preprocessor Macros . . . 119
The Fold Compound . . . 255
The Dynamic Programming Matrices . . . 270
Utilities . . . 125
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings . . . 281
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures . . . 285
Various utilities for Sequence Alignments, and Comparative Structure Prediction . . . 301
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments . . . 304
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More . . . 311

Chapter 11

Data Structure Index

11.1 Data Structures

Here are the data structures with brief descriptions:

_struct_en
: Data structure for energy_of_move() . . . 319

Postorder_list
: Postorder data structure . . . 319

swString
: Some other data structure . . . 319

Tree
: Tree data structure . . . 320

TwoDpfold_vars
: Variables compound for 2Dfold partition function folding . . . 320

vrna_subopt_sol_s
: Solution element from subopt.c . . . 322

Chapter 12

File Index

12.1 File List

Here is a list of all documented files with brief descriptions:

ViennaRNA/1.8.4_epars.h
Free energy parameters for parameter file conversion . . . 323
ViennaRNA/1.8.4_intloops.h
Free energy parameters for interior loop contributions needed by the parameter file conversion functions . . . 323
ViennaRNA/2Dfold.h . . . 324
ViennaRNA/2Dpfold.h . . . 325
ViennaRNA/**ali_plex.h** . . . **??**
ViennaRNA/alifold.h
Compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments . . . 328
ViennaRNA/aln_util.h
Various utility- and helper-functions for sequence alignments and comparative structure prediction . . . 330
ViennaRNA/alphabet.h
Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets . . . 331
ViennaRNA/boltzmann_sampling.h
Boltzmann Sampling of secondary structures from the ensemble . . . 333
ViennaRNA/centroid.h
Centroid structure computation . . . 335
ViennaRNA/cofold.h
MFE version of cofolding routines . . . 336
ViennaRNA/constraints.h
Functions and data structures for constraining secondary structure predictions and evaluation . . . 337
ViennaRNA/constraints_hard.h
Functions and data structures for handling of secondary structure hard constraints . . . 339
ViennaRNA/constraints_SHAPE.h
This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints . . . 342
ViennaRNA/constraints_soft.h
Functions and data structures for secondary structure soft constraints . . . 344
ViennaRNA/convert_epars.h
Functions and definitions for energy parameter file format conversion . . . 345
ViennaRNA/data_structures.h . . . 346

ViennaRNA/dist_vars.h
Global variables for Distance-Package . . . 349
ViennaRNA/dp_matrices.h . . . 350
ViennaRNA/duplex.h
Duplex folding function declarations.. . . . 351
ViennaRNA/edit_cost.h
Global variables for Edit Costs included by treedist.c and stringdist.c . . . 351
ViennaRNA/energy_const.h . . . 352
ViennaRNA/**energy_par.h** . . . **??**
ViennaRNA/equilibrium_probs.h
Equilibrium Probability implementations . . . 353
ViennaRNA/eval.h
Functions and variables related to energy evaluation of sequence/structure pairs . . . 354
ViennaRNA/exterior_loops.h
Energy evaluation of exterior loops for MFE and partition function calculations . . . 371
ViennaRNA/file_formats.h
Functions dealing with file formats for RNA sequences, structures, and alignments . . . 372
ViennaRNA/findpath.h . . . 374
ViennaRNA/fold.h
MFE calculations for single RNA sequences . . . 375
ViennaRNA/fold_vars.h
Here all all declarations of the global variables used throughout RNAlib . . . 376
ViennaRNA/gquad.h
Various functions related to G-quadruplex computations . . . 378
ViennaRNA/hairpin_loops.h
Energy evaluation of hairpin loops for MFE and partition function calculations . . . 379
ViennaRNA/interior_loops.h
Energy evaluation of interior loops for MFE and partition function calculations . . . 380
ViennaRNA/**intl11.h** . . . **??**
ViennaRNA/**intl11dH.h** . . . **??**
ViennaRNA/**intl21.h** . . . **??**
ViennaRNA/**intl21dH.h** . . . **??**
ViennaRNA/**intl22.h** . . . **??**
ViennaRNA/**intl22dH.h** . . . **??**
ViennaRNA/inverse.h
Inverse folding routines . . . 381
ViennaRNA/Lfold.h
Predicting local MFE structures of large sequences . . . 382
ViennaRNA/ligand.h
Functions for incorporation of ligands binding to haipirn and interior loop motifs . . . 382
ViennaRNA/loop_energies.h
Energy evaluation for MFE and partition function calculations . . . 383
ViennaRNA/LPfold.h
Function declarations of partition function variants of the Lfold algorithm . . . 384
ViennaRNA/MEA.h
Computes a MEA (maximum expected accuracy) structure . . . 385
ViennaRNA/mfe.h
MFE calculations for single RNA sequences . . . 386
ViennaRNA/mm.h
Several Maximum Matching implementations . . . 386
ViennaRNA/model.h
The model details data structure and its corresponding modifiers . . . 387
ViennaRNA/**move_set.h** . . . **??**
ViennaRNA/multibranch_loops.h
Energy evaluation of multibranch loops for MFE and partition function calculations . . . 391
ViennaRNA/naview.h . . . 393
ViennaRNA/**pair_mat.h** . . . **??**
ViennaRNA/params.h . . . 393

ViennaRNA/part_func.h
Partition function of single RNA sequences . . . 395
ViennaRNA/part_func_co.h
Partition function for two RNA sequences . . . 398
ViennaRNA/part_func_up.h
Partition Function Cofolding as stepwise process . . . 403
ViennaRNA/perturbation_fold.h
Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments . . . 404
ViennaRNA/**PKplex.h** . . . **??**
ViennaRNA/**plex.h** . . . **??**
ViennaRNA/plot_aln.h
Various functions for plotting Sequence / Structure Alignments . . . 406
ViennaRNA/plot_layouts.h
Secondary structure plot layout algorithms . . . 406
ViennaRNA/plot_structure.h
Various functions for plotting RNA secondary structures . . . 408
ViennaRNA/**ProfileAln.h** . . . **??**
ViennaRNA/profiledist.h . . . 409
ViennaRNA/PS_dot.h
Various functions for plotting RNA secondary structures, dot-plots and other visualizations . . . 411
ViennaRNA/read_epars.h . . . 411
ViennaRNA/ribo.h
Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments . . . 412
ViennaRNA/RNAstruct.h
Parsing and Coarse Graining of Structures . . . 412
ViennaRNA/**snofold.h** . . . **??**
ViennaRNA/**snoop.h** . . . **??**
ViennaRNA/string_utils.h
General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package . . . 414
ViennaRNA/stringdist.h
Functions for String Alignment . . . 416
ViennaRNA/structure_utils.h
Various utility- and helper-functions for secondary structure parsing, converting, etc . . . 418
ViennaRNA/subopt.h
RNAsubopt and density of states declarations . . . 421
ViennaRNA/**svm_utils.h** . . . **??**
ViennaRNA/treedist.h
Functions for Tree Edit Distances . . . 422
ViennaRNA/utils.h
General utility- and helper-functions used throughout the *ViennaRNA Package* . . . 423

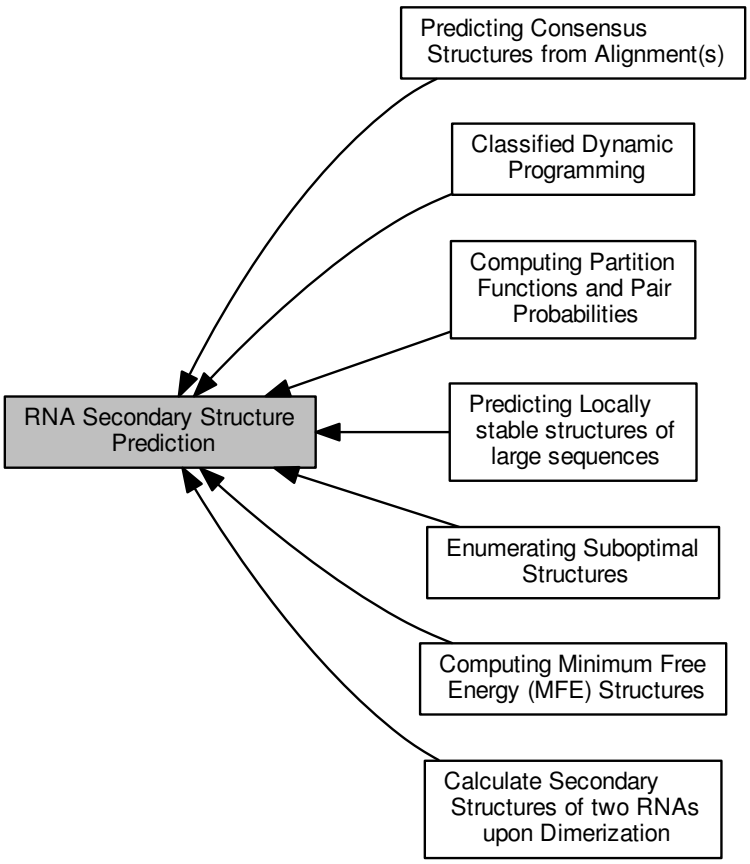
Chapter 13

Module Documentation

13.1 RNA Secondary Structure Prediction

This module contains all functions related to thermodynamic folding of RNAs.

Collaboration diagram for RNA Secondary Structure Prediction:



Modules

- Computing Minimum Free Energy (MFE) Structures

  This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.
- Computing Partition Functions and Pair Probabilities

  This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.
- Enumerating Suboptimal Structures
- Calculate Secondary Structures of two RNAs upon Dimerization

  Predict structures formed by two molecules upon hybridization.
- Predicting Consensus Structures from Alignment(s)

  compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments
- Predicting Locally stable structures of large sequences
- Classified Dynamic Programming

Files

- file mm.h

  Several Maximum Matching implementations.

13.1.1 Detailed Description

This module contains all functions related to thermodynamic folding of RNAs.

13.2 Inverse Secondary Structure Prediction

Files

- file inverse.h

  Inverse folding routines.

Functions

- float inverse_fold (char *start, const char *target)

  Find sequences with predefined structure.
- float inverse_pf_fold (char *start, const char *target)

  Find sequence that maximizes probability of a predefined structure.

Variables

- char * symbolset

  This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.
- float final_cost
- int give_up
- int inv_verbose

13.2.1 Detailed Description

We provide two functions that search for sequences with a given structure, thereby inverting the folding routines.

13.2.2 Function Documentation

13.2.2.1 float inverse_fold (char * *start*, const char * *target*)

```
#include <ViennaRNA/inverse.h>
```

Find sequences with predefined structure.

This function searches for a sequence with minimum free energy structure provided in the parameter 'target', starting with sequence 'start'. It returns 0 if the search was successful, otherwise a structure distance in terms of the energy difference between the search result and the actual target 'target' is returned. The found sequence is returned in 'start'. If give_up is set to 1, the function will return as soon as it is clear that the search will be unsuccessful, this speeds up the algorithm if you are only interested in exact solutions.

Parameters

start	The start sequence
target	The target secondary structure in dot-bracket notation

Returns

The distance to the target in case a search was unsuccessful, 0 otherwise

13.2.2.2 float inverse_pf_fold (char * *start*, const char * *target*)

```
#include <ViennaRNA/inverse.h>
```

Find sequence that maximizes probability of a predefined structure.

This function searches for a sequence with maximum probability to fold into the provided structure 'target' using the partition function algorithm. It returns $-kT \cdot \log(p)$ where p is the frequency of 'target' in the ensemble of possible structures. This is usually much slower than inverse_fold().

Parameters

start	The start sequence
target	The target secondary structure in dot-bracket notation

Returns

The distance to the target in case a search was unsuccessful, 0 otherwise

13.2.3 Variable Documentation

13.2.3.1 float final_cost

```
#include <ViennaRNA/inverse.h>
```

when to stop inverse_pf_fold()

13.2.3.2 int give_up

```
#include <ViennaRNA/inverse.h>
```

default 0: try to minimize structure distance even if no exact solution can be found

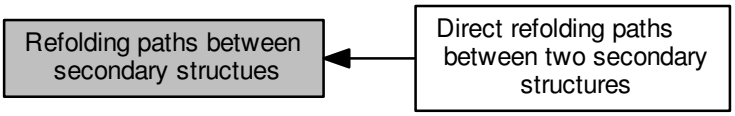
13.2.3.3 int inv_verbose

```
#include <ViennaRNA/inverse.h>
```

print out substructure on which inverse_fold() fails

13.3 Refolding paths between secondary structues

Collaboration diagram for Refolding paths between secondary structues:



Modules

- Direct refolding paths between two secondary structures

  Implementation of heuristics to explore optimal (re-)folding paths between two secondary structures.

13.3.1 Detailed Description

13.4 Free Energy Evaluation for given Sequence / Structure Pairs

This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

Files

- file eval.h

  Functions and variables related to energy evaluation of sequence/structure pairs.

Functions

- int vrna_eval_ext_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate free energy of an exterior hairpin loop.
- int vrna_eval_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate free energy of a hairpin loop.

13.4.1 Detailed Description

This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

13.4.2 Function Documentation

13.4.2.1 int vrna_eval_hp_loop (vrna_fold_compound_t * *vc*, int *i*, int *j*)

```
#include <ViennaRNA/hairpin_loops.h>
```

Evaluate free energy of a hairpin loop.

Note

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_VC_TYPE_SINGLE or VRNA_VC_TYPE_ALIGNMENT

Parameters

vc	The vrna_fold_compound_t for the particular energy evaluation
i	5'-position of the base pair
j	3'-position of the base pair

Returns

Free energy of the hairpin loop closed by (i, j) in deka-kal/mol

13.5 Processing and Evaluating Decomposed Loops

Files

- file exterior_loops.h

  Energy evaluation of exterior loops for MFE and partition function calculations.
- file gquad.h

  Various functions related to G-quadruplex computations.
- file hairpin_loops.h

  Energy evaluation of hairpin loops for MFE and partition function calculations.
- file interior_loops.h

  Energy evaluation of interior loops for MFE and partition function calculations.
- file loop_energies.h

  Energy evaluation for MFE and partition function calculations.
- file multibranch_loops.h

  Energy evaluation of multibranch loops for MFE and partition function calculations.

Functions

- int E_ExtLoop (int type, int si1, int sj1, vrna_param_t *P)
- FLT_OR_DBL exp_E_ExtLoop (int type, int si1, int sj1, vrna_exp_param_t *P)
- int E_Stem (int type, int si1, int sj1, int extLoop, vrna_param_t *P)
- FLT_OR_DBL exp_E_Stem (int type, int si1, int sj1, int extLoop, vrna_exp_param_t *P)
- int * get_gquad_matrix (short *S, vrna_param_t *P)

  Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.
- int parse_gquad (const char *struc, int *L, int l[3])
- PRIVATE int backtrack_GQuad_IntLoop (int c, int i, int j, int type, short *S, int *ggg, int *index, int *p, int *q, vrna_param_t *P)
- PRIVATE int backtrack_GQuad_IntLoop_L (int c, int i, int j, int type, short *S, int **ggg, int maxdist, int *p, int *q, vrna_param_t *P)
- PRIVATE int E_Hairpin (int size, int type, int si1, int sj1, const char *string, vrna_param_t *P)

  Compute the Energy of a hairpin-loop.
- PRIVATE FLT_OR_DBL exp_E_Hairpin (int u, int type, short si1, short sj1, const char *string, vrna_exp_↩ param_t *P)

  Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop.
- int vrna_E_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.
- int vrna_E_ext_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.
- FLT_OR_DBL vrna_exp_E_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  High-Level function for hairpin loop energy evaluation (partition function variant)
- int vrna_BT_hp_loop (vrna_fold_compound_t *vc, int i, int j, int en, vrna_bp_stack_t *bp_stack, int *stack↩ _count)

  Backtrack a hairpin loop closed by (i, j).
- PRIVATE int E_IntLoop (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t *P)
- PRIVATE FLT_OR_DBL exp_E_IntLoop (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, vrna_exp_param_t *P)
- int E_stack (int i, int j, vrna_fold_compound_t *vc)

  Evaluate energy of a base pair stack closed by (i,j)
- int vrna_BT_stack (vrna_fold_compound_t *vc, int *i, int *j, int *en, vrna_bp_stack_t *bp_stack, int *stack↩ _count)

Backtrack a stacked pair closed by (i, j).

- int vrna_BT_int_loop (vrna_fold_compound_t *vc, int *i, int *j, int en, vrna_bp_stack_t *bp_stack, int *stack_count)

  Backtrack an interior loop closed by (i, j).
- int E_mb_loop_stack (int i, int j, vrna_fold_compound_t *vc)

  Evaluate energy of a multi branch helices stacking onto closing pair (i,j)
- int vrna_BT_mb_loop (vrna_fold_compound_t *vc, int *i, int *j, int *k, int en, int *component1, int *component2)

  Backtrack the decomposition of a multi branch loop closed by (i, j).

13.5.1 Detailed Description

13.5.2 Function Documentation

13.5.2.1 int E_ExtLoop (int *type*, int *si1*, int *sj1*, vrna_param_t * *P*)

```
#include <ViennaRNA/exterior_loops.h>
```

Compute the Energy contribution of an Exterior loop stem

This definition is a wrapper for the E_Stem() funtion. It is substituted by an E_Stem() funtion call with argument extLoop=1, so the energy contribution returned reflects a stem introduced in an exterior-loop.
As for the parameters si1 and sj1 of the substituted E_Stem() function, you can inhibit to take 5'-, 3'-dangles or mismatch contributions to be taken into account by passing -1 to these parameters.

See also

E_Stem()

Parameters

type	The pair type of the stem-closing pair
si1	The 5'-mismatching nucleotide
sj1	The 3'-mismatching nucleotide
P	The datastructure containing scaled energy parameters

Returns

The energy contribution of the introduced exterior-loop stem

13.5.2.2 FLT_OR_DBL exp_E_ExtLoop (int *type*, int *si1*, int *sj1*, vrna_exp_param_t * *P*)

```
#include <ViennaRNA/exterior_loops.h>
```

This is the partition function variant of E_ExtLoop()

See also

E_ExtLoop()

Returns

The Boltzmann weighted energy contribution of the introduced exterior-loop stem

13.5.2.3 int E_Stem (int *type*, int *si1*, int *sj1*, int *extLoop*, vrna_param_t * *P*)

```
#include <ViennaRNA/exterior_loops.h>
```

Compute the energy contribution of a stem branching off a loop-region

This function computes the energy contribution of a stem that branches off a loop region. This can be the case in multiloops, when a stem branching off increases the degree of the loop but also *immediately interior base pairs* of an exterior loop contribute free energy. To switch the bahavior of the function according to the evaluation of a multiloop- or exterior-loop-stem, you pass the flag 'extLoop'. The returned energy contribution consists of a TerminalAU penalty if the pair type is greater than 2, dangling end contributions of mismatching nucleotides adjacent to the stem if only one of the si1, sj1 parameters is greater than 0 and mismatch energies if both mismatching nucleotides are positive values. Thus, to avoid incooperating dangling end or mismatch energies just pass a negative number, e.g. -1 to the mismatch argument.

This is an illustration of how the energy contribution is assembled:

```
      3'  5'
      |   |
      X - Y
5'-si1     sj1-3'
```

Here, (X,Y) is the base pair that closes the stem that branches off a loop region. The nucleotides si1 and sj1 are the 5'- and 3'- mismatches, respectively. If the base pair type of (X,Y) is greater than 2 (i.e. an A-U or G-U pair, the TerminalAU penalty will be included in the energy contribution returned. If si1 and sj1 are both nonnegative numbers, mismatch energies will also be included. If one of sij or sj1 is a negtive value, only 5' or 3' dangling end contributions are taken into account. To prohibit any of these mismatch contributions to be incoorporated, just pass a negative number to both, si1 and sj1. In case the argument extLoop is 0, the returned energy contribution also includes the *internal-loop-penalty* of a multiloop stem with closing pair type.

See also

E_MLstem()
E_ExtLoop()

Note

This function is threadsafe

Parameters

type	The pair type of the first base pair un the stem
si1	The 5'-mismatching nucleotide
sj1	The 3'-mismatching nucleotide
extLoop	A flag that indicates whether the contribution reflects the one of an exterior loop or not
P	The datastructure containing scaled energy parameters

Returns

The Free energy of the branch off the loop in dcal/mol

13.5.2.4 FLT_OR_DBL exp_E_Stem (int *type*, int *si1*, int *sj1*, int *extLoop*, vrna_exp_param_t * *P*)

```
#include <ViennaRNA/exterior_loops.h>
```

Compute the Boltzmann weighted energy contribution of a stem branching off a loop-region

This is the partition function variant of E_Stem()

See also

E_Stem()

Note

This function is threadsafe

Returns

The Boltzmann weighted energy contribution of the branch off the loop

13.5.2.5 int* get_gquad_matrix (short * *S*, vrna_param_t * *P*)

```
#include <ViennaRNA/gquad.h>
```

Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.

At each position ij in the matrix, the minimum free energy of any G-quadruplex delimited by i and j is stored. If no G-quadruplex formation is possible, the matrix element is set to INF. Access the elements in the matrix via matrix[indx[j]+i]. To get the integer array indx see get_jindx().

See also

get_jindx(), encode_sequence()

Parameters

S	The encoded sequence
P	A pointer to the data structure containing the precomputed energy contributions

Returns

A pointer to the G-quadruplex contribution matrix

13.5.2.6 int parse_gquad (const char * *struc*, int * *L*, int *l[3]*)

```
#include <ViennaRNA/gquad.h>
```

given a dot-bracket structure (possibly) containing gquads encoded by '+' signs, find first gquad, return end position or 0 if none found Upon return L and l[] contain the number of stacked layers, as well as the lengths of the linker regions. To parse a string with many gquads, call parse_gquad repeatedly e.g. end1 = parse_gquad(struc, &L, l); ... ; end2 = parse_gquad(struc+end1, &L, l); end2+=end1; ... ; end3 = parse_gquad(struc+end2, &L, l); end3+=end2; ... ;

13.5.2.7 PRIVATE int backtrack_GQuad_IntLoop (int *c*, int *i*, int *j*, int *type*, short * *S*, int * *ggg*, int * *index*, int * *p*, int * *q*, vrna_param_t * *P*)

```
#include <ViennaRNA/gquad.h>
```

backtrack an interior loop like enclosed g-quadruplex with closing pair (i,j)

Parameters

c	The total contribution the loop should resemble
i	position i of enclosing pair
j	position j of enclosing pair
type	base pair type of enclosing pair (must be reverse type)
S	integer encoded sequence
ggg	triangular matrix containing g-quadruplex contributions
index	the index for accessing the triangular matrix
p	here the 5' position of the gquad is stored
q	here the 3' position of the gquad is stored
P	the datastructure containing the precalculated contibutions

Returns

1 on success, 0 if no gquad found

13.5.2.8 PRIVATE int backtrack_GQuad_IntLoop_L (int *c*, int *i*, int *j*, int *type*, short * *S*, int ** *ggg*, int *maxdist*, int * *p*, int * *q*, vrna_param_t * *P*)

```
#include <ViennaRNA/gquad.h>
```

backtrack an interior loop like enclosed g-quadruplex with closing pair (i,j) with underlying Lfold matrix

Parameters

c	The total contribution the loop should resemble
i	position i of enclosing pair
j	position j of enclosing pair
type	base pair type of enclosing pair (must be reverse type)
S	integer encoded sequence
ggg	triangular matrix containing g-quadruplex contributions
p	here the 5' position of the gquad is stored
q	here the 3' position of the gquad is stored
P	the datastructure containing the precalculated contibutions

Returns

1 on success, 0 if no gquad found

13.5.2.9 PRIVATE int E_Hairpin (int *size*, int *type*, int *si1*, int *sj1*, const char * *string*, vrna_param_t * *P*)

```
#include <ViennaRNA/hairpin_loops.h>
```

Compute the Energy of a hairpin-loop.

To evaluate the free energy of a hairpin-loop, several parameters have to be known. A general hairpin-loop has this structure:

```
  a3 a4
a2     a5
a1     a6
  X - Y
  |   |
  5'  3'
```

where X-Y marks the closing pair [e.g. a **(G,C)** pair]. The length of this loop is 6 as there are six unpaired nucleotides (a1-a6) enclosed by (X,Y). The 5' mismatching nucleotide is a1 while the 3' mismatch is a6. The nucleotide sequence of this loop is "a1.a2.a3.a4.a5.a6"

Note

The parameter sequence should contain the sequence of the loop in capital letters of the nucleic acid alphabet if the loop size is below 7. This is useful for unusually stable tri-, tetra- and hexa-loops which are treated differently (based on experimental data) if they are tabulated.

See also

scale_parameters()
vrna_param_t

Warning

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!
Energy evaluation may change due to updates in global variable "tetra_loop"

Parameters

size	The size of the loop (number of unpaired nucleotides)
type	The pair type of the base pair closing the hairpin
si1	The 5'-mismatching nucleotide
sj1	The 3'-mismatching nucleotide
string	The sequence of the loop
P	The datastructure containing scaled energy parameters

Returns

The Free energy of the Hairpin-loop in dcal/mol

13.5.2.10 PRIVATE FLT_OR_DBL exp_E_Hairpin (int *u*, int *type*, short *si1*, short *sj1*, const char * *string*, vrna_exp_param_t * *P*)

```
#include <ViennaRNA/hairpin_loops.h>
```

Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop.

multiply by scale[u+2]

See also

get_scaled_pf_parameters()
vrna_exp_param_t
E_Hairpin()

Warning

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!
Energy evaluation may change due to updates in global variable "tetra_loop"

Parameters

u	The size of the loop (number of unpaired nucleotides)
type	The pair type of the base pair closing the hairpin
si1	The 5'-mismatching nucleotide
sj1	The 3'-mismatching nucleotide
string	The sequence of the loop
P	The datastructure containing scaled Boltzmann weights of the energy parameters

Returns

The Boltzmann weight of the Hairpin-loop

13.5.2.11 int vrna_E_hp_loop (vrna_fold_compound_t * *vc*, int *i*, int *j*)

```
#include <ViennaRNA/hairpin_loops.h>
```

Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.

This function evaluates the free energy of a hairpin loop

In case the base pair is not allowed due to a constraint conflict, this function returns INF.

Note

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_VC_TYPE_SINGLE or VRNA_VC_TYPE_ALIGNMENT

Parameters

vc	The vrna_fold_compound_t that stores all relevant model settings
i	The 5' nucleotide of the base pair (3' to evaluate the pair as exterior hairpin loop)
j	The 3' nucleotide of the base pair (5' to evaluate the pair as exterior hairpin loop)

Returns

The free energy of the hairpin loop in 10cal/mol

13.5.2.12 int vrna_E_ext_hp_loop (vrna_fold_compound_t * *vc*, int *i*, int *j*)

```
#include <ViennaRNA/hairpin_loops.h>
```

Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.

Note

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_VC_TYPE_SINGLE or VRNA_VC_TYPE_ALIGNMENT

13.5.2.13 FLT_OR_DBL vrna_exp_E_hp_loop (vrna_fold_compound_t * *vc*, int *i*, int *j*)

```
#include <ViennaRNA/hairpin_loops.h>
```

High-Level function for hairpin loop energy evaluation (partition function variant)

See also

vrna_E_hp_loop() for it's free energy counterpart

Note

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_VC_TYPE_SINGLE or VRNA_VC_TYPE_ALIGNMENT

13.5.2.14 int vrna_BT_hp_loop (vrna_fold_compound_t * *vc*, int *i*, int *j*, int *en*, vrna_bp_stack_t * *bp_stack*, int * *stack_count*)

```
#include <ViennaRNA/hairpin_loops.h>
```

Backtrack a hairpin loop closed by (i, j).

Note

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_VC_TYPE_SINGLE or VRNA_VC_TYPE_ALIGNMENT

13.5.2.15 int E_IntLoop (int *n1*, int *n2*, int *type*, int *type_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*, vrna_param_t * *P*)

```
#include <ViennaRNA/interior_loops.h>
```

Compute the Energy of an interior-loop

This function computes the free energy ΔG of an interior-loop with the following structure:

```
     3'  5'
     |   |
     U - V
a_n       b_1
 .         .
 .         .
 .         .
a_1       b_m
     X - Y
     |   |
     5'  3'
```

This general structure depicts an interior-loop that is closed by the base pair (X,Y). The enclosed base pair is (V,U) which leaves the unpaired bases a_1-a_n and b_1-b_n that constitute the loop. In this example, the length of the interior-loop is $(n + m)$ where n or m may be 0 resulting in a bulge-loop or base pair stack. The mismatching nucleotides for the closing pair (X,Y) are:
5'-mismatch: a_1
3'-mismatch: b_m
and for the enclosed base pair (V,U):
5'-mismatch: b_1
3'-mismatch: a_n

Note

Base pairs are always denoted in 5'->3' direction. Thus the enclosed base pair must be 'turned arround' when evaluating the free energy of the interior-loop

See also

scale_parameters()
vrna_param_t

Note

This function is threadsafe

Parameters

n1	The size of the 'left'-loop (number of unpaired nucleotides)
n2	The size of the 'right'-loop (number of unpaired nucleotides)
type	The pair type of the base pair closing the interior loop
type_2	The pair type of the enclosed base pair
si1	The 5'-mismatching nucleotide of the closing pair
sj1	The 3'-mismatching nucleotide of the closing pair
sp1	The 3'-mismatching nucleotide of the enclosed pair
sq1	The 5'-mismatching nucleotide of the enclosed pair

Returns

The Free energy of the Interior-loop in dcal/mol

13.5.2.16 PUBLIC FLT_OR_DBL exp_E_IntLoop (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*, vrna_exp_param_t * *P*)

```
#include <ViennaRNA/interior_loops.h>
```

Compute Boltzmann weight $e^{-\Delta G/kT}$ of interior loop

multiply by scale[u1+u2+2] for scaling

See also

get_scaled_pf_parameters()
vrna_exp_param_t
E_IntLoop()

Note

This function is threadsafe

Parameters

u1	The size of the 'left'-loop (number of unpaired nucleotides)
u2	The size of the 'right'-loop (number of unpaired nucleotides)
type	The pair type of the base pair closing the interior loop
type2	The pair type of the enclosed base pair
si1	The 5'-mismatching nucleotide of the closing pair
sj1	The 3'-mismatching nucleotide of the closing pair
sp1	The 3'-mismatching nucleotide of the enclosed pair
sq1	The 5'-mismatching nucleotide of the enclosed pair
P	The datastructure containing scaled Boltzmann weights of the energy parameters

Returns

The Boltzmann weight of the Interior-loop

13.5.2.17 int E_mb_loop_stack (int *i*, int *j*, vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/multibranch_loops.h>
```

Evaluate energy of a multi branch helices stacking onto closing pair (i,j)

Computes total free energy for coaxial stacking of (i.j) with (i+1.k) or (k+1.j-1)

13.5.2.18 int vrna_BT_mb_loop (vrna_fold_compound_t * *vc*, int * *i*, int * *j*, int * *k*, int *en*, int * *component1*, int * *component2*)

```
#include <ViennaRNA/multibranch_loops.h>
```

Backtrack the decomposition of a multi branch loop closed by (i, j).

Parameters

vc	The vrna_fold_compound_t filled with all relevant data for backtracking
i	5' position of base pair closing the loop (will be set to 5' position of leftmost decomposed block upon successful backtracking)
j	3' position of base pair closing the loop (will be set to 3' position of rightmost decomposed block upon successful backtracking)
k	Split position that delimits leftmost from rightmost block, [i,k] and [k+1, j], respectively. (Will be set upon successful backtracking)
en	The energy contribution of the substructure enclosed by (i, j)
component1	Type of leftmost block (1 = ML, 2 = C)
component2	Type of rightmost block (1 = ML, 2 = C)

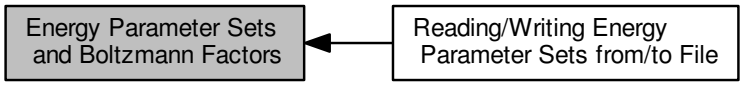
Returns

1, if backtracking succeeded, 0 otherwise.

13.6 Energy Parameter Sets and Boltzmann Factors

All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

Collaboration diagram for Energy Parameter Sets and Boltzmann Factors:



Modules

- Reading/Writing Energy Parameter Sets from/to File

  Read and Write energy parameter sets from and to text files.

Files

- file params.h

Data Structures

- struct vrna_param_s

  The datastructure that contains temperature scaled energy parameters. More...
- struct vrna_exp_param_s

  The datastructure that contains temperature scaled Boltzmann weights of the energy parameters. More...

Typedefs

- typedef struct vrna_param_s vrna_param_t

  Typename for the free energy parameter data structure vrna_params.
- typedef struct vrna_exp_param_s vrna_exp_param_t

  Typename for the Boltzmann factor data structure vrna_exp_params.
- typedef struct vrna_param_s paramT

  Old typename of vrna_param_s.
- typedef struct vrna_exp_param_s pf_paramT

  Old typename of vrna_exp_param_s.

Functions

- vrna_param_t * vrna_params (vrna_md_t *md)

  Get a data structure containing prescaled free energy parameters.
- vrna_param_t * vrna_params_copy (vrna_param_t *par)

  Get a copy of the provided free energy parameters.
- vrna_exp_param_t * vrna_exp_params (vrna_md_t *md)

  Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.
- vrna_exp_param_t * vrna_exp_params_comparative (unsigned int n_seq, vrna_md_t *md)

  Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)
- vrna_exp_param_t * vrna_exp_params_copy (vrna_exp_param_t *par)

  Get a copy of the provided free energy parameters (provided as Boltzmann factors)
- void vrna_params_subst (vrna_fold_compound_t *vc, vrna_param_t *par)

  Update/Reset energy parameters data structure within a vrna_fold_compound_t.
- void vrna_exp_params_subst (vrna_fold_compound_t *vc, vrna_exp_param_t *params)

  Update the energy parameters for subsequent partition function computations.
- void vrna_exp_params_rescale (vrna_fold_compound_t *vc, double *mfe)

  Rescale Boltzmann factors for partition function computations.
- void vrna_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)

  Reset free energy parameters within a vrna_fold_compound_t according to provided, or default model details.
- void vrna_exp_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)

  Reset Boltzmann factors for partition function computations within a vrna_fold_compound_t according to provided, or default model details.
- vrna_exp_param_t * get_scaled_pf_parameters (void)
- vrna_exp_param_t * get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.
- vrna_exp_param_t * get_boltzmann_factor_copy (vrna_exp_param_t *parameters)

  Get a copy of already precomputed Boltzmann factors.
- vrna_exp_param_t * get_scaled_alipf_parameters (unsigned int n_seq)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)
- vrna_exp_param_t * get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.
- vrna_param_t * scale_parameters (void)

  Get precomputed energy contributions for all the known loop types.
- vrna_param_t * get_scaled_parameters (double temperature, vrna_md_t md)

  Get precomputed energy contributions for all the known loop types.

13.6.1 Detailed Description

All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

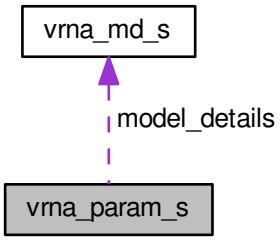
This module covers all relevant functions for precalculation of the energy parameters necessary for the folding routines provided by RNAlib. Furthermore, the energy parameter set in the RNAlib can be easily exchanged by a user-defined one. It is also possible to write the current energy parameter set into a text file.

13.6.2 Data Structure Documentation

13.6.2.1 struct vrna_param_s

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for vrna_param_s:



Data Fields

- double temperature

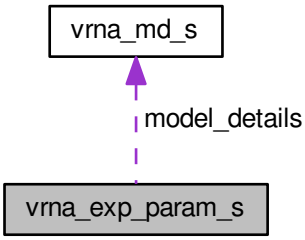
  Temperature used for loop contribution scaling.
- vrna_md_t model_details

  Model details to be used in the recursions.

13.6.2.2 struct vrna_exp_param_s

The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.

Collaboration diagram for vrna_exp_param_s:



Data Fields

- int id
  An identifier for the data structure.
- double pf_scale
  Scaling factor to avoid over-/underflows.
- double temperature
  Temperature used for loop contribution scaling.
- double alpha
  Scaling factor for the thermodynamic temperature.
- vrna_md_t model_details
  Model details to be used in the recursions.

13.6.2.2.1 Field Documentation

13.6.2.2.1.1 int vrna_exp_param_s::id

An identifier for the data structure.

Deprecated This attribute will be removed in version 3

13.6.2.2.1.2 double vrna_exp_param_s::alpha

Scaling factor for the thermodynamic temperature.

This allows for temperature scaling in Boltzmann factors independently from the energy contributions. The resulting Boltzmann factors are then computed by $e^{-E/(\alpha \cdot K \cdot T)}$

13.6.3 Typedef Documentation

13.6.3.1 typedef struct vrna_param_s paramT

```
#include <ViennaRNA/params.h>
```

Old typename of vrna_param_s.

Deprecated Use vrna_param_t instead!

13.6.3.2 typedef struct vrna_exp_param_s pf_paramT

```
#include <ViennaRNA/params.h>
```

Old typename of vrna_exp_param_s.

Deprecated Use vrna_exp_param_t instead!

13.6.4 Function Documentation

13.6.4.1 vrna_param_t* vrna_params (vrna_md_t * *md*)

```
#include <ViennaRNA/params.h>
```

Get a data structure containing prescaled free energy parameters.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested vrna_param_t structure.

See also

vrna_md_t, vrna_md_set_default(), vrna_exp_params()

Parameters

md	A pointer to the model details to store inside the structure (Maybe NULL)

Returns

A pointer to the memory location where the requested parameters are stored

13.6.4.2 vrna_param_t* vrna_params_copy (vrna_param_t * *par*)

```
#include <ViennaRNA/params.h>
```

Get a copy of the provided free energy parameters.

If NULL is passed as parameter, a default set of energy parameters is created and returned.

See also

vrna_params(), vrna_param_t

Parameters

par	The free energy parameters that are to be copied (Maybe NULL)

Returns

A copy or a default set of the (provided) parameters

13.6.4.3 vrna_exp_param_t* vrna_exp_params (vrna_md_t * *md*)

```
#include <ViennaRNA/params.h>
```

Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.

This function returns a data structure that contains all necessary precomputed energy contributions for each type of loop.

In contrast to vrna_params(), the free energies within this data structure are stored as their Boltzmann factors, i.e.

$exp(-E/kT)$

where E is the free energy.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested vrna_exp_param_t structure.

See also

vrna_md_t, vrna_md_set_default(), vrna_params(), vrna_rescale_pf_params()

Parameters

md	A pointer to the model details to store inside the structure (Maybe NULL)

Returns

A pointer to the memory location where the requested parameters are stored

13.6.4.4 vrna_exp_param_t* vrna_exp_params_comparative (unsigned int *n_seq*, vrna_md_t * *md*)

```
#include <ViennaRNA/params.h>
```

Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested vrna_exp_param_t structure.

See also

vrna_md_t, vrna_md_set_default(), vrna_exp_params(), vrna_params()

Parameters

n_seq	The number of sequences in the alignment
md	A pointer to the model details to store inside the structure (Maybe NULL)

Returns

A pointer to the memory location where the requested parameters are stored

13.6.4.5 vrna_exp_param_t* vrna_exp_params_copy (vrna_exp_param_t * *par*)

```
#include <ViennaRNA/params.h>
```

Get a copy of the provided free energy parameters (provided as Boltzmann factors)

If NULL is passed as parameter, a default set of energy parameters is created and returned.

See also

vrna_exp_params(), vrna_exp_param_t

Parameters

par	The free energy parameters that are to be copied (Maybe NULL)

Returns

A copy or a default set of the (provided) parameters

13.6.4.6 void vrna_params_subst (vrna_fold_compound_t * *vc*, vrna_param_t * *par*)

```
#include <ViennaRNA/params.h>
```

Update/Reset energy parameters data structure within a vrna_fold_compound_t.

Passing NULL as second argument leads to a reset of the energy parameters within vc to their default values. Otherwise, the energy parameters provided will be copied over into vc.

See also

vrna_params_reset(), vrna_param_t, vrna_md_t, vrna_params()

Parameters

vc	The vrna_fold_compound_t that is about to receive updated energy parameters
par	The energy parameters used to substitute those within vc (Maybe NULL)

Notes on Scripting Interface This function is attached to vrna_fc_s objects as **params_subst()** method.

13.6.4.7 void vrna_exp_params_subst (vrna_fold_compound_t * *vc*, vrna_exp_param_t * *params*)

```
#include <ViennaRNA/params.h>
```

Update the energy parameters for subsequent partition function computations.

This function can be used to properly assign new energy parameters for partition function computations to a vrna_fold_compound_t. For this purpose, the data of the provided pointer `params` will be copied into `vc` and a re-computation of the partition function scaling factor is issued, if the `pf_scale` attribute of `params` is less than `1.0`.

Passing NULL as second argument leads to a reset of the energy parameters within vc to their default values

See also

vrna_exp_params_reset(), vrna_exp_params_rescale(), vrna_exp_param_t, vrna_md_t, vrna_exp_params()

Parameters

vc	The fold compound data structure
params	A pointer to the new energy parameters

Notes on Scripting Interface This function is attached to vrna_fc_s objects as **exp_params_subst()** method.

13.6.4.8 void vrna_exp_params_rescale (vrna_fold_compound_t * *vc*, double * *mfe*)

```
#include <ViennaRNA/params.h>
```

Rescale Boltzmann factors for partition function computations.

This function may be used to (automatically) rescale the Boltzmann factors used in partition function computations. Since partition functions over subsequences can easily become extremely large, the RNAlib internally rescales them to avoid numerical over- and/or underflow. Therefore, a proper scaling factor s needs to be chosen that in turn is then used to normalize the corresponding partition functions $\hat{q}[i,j] = q[i,j]/s^{(j-i+1)}$.

This function provides two ways to automatically adjust the scaling factor.

1. Automatic guess
2. Automatic adjustment according to MFE

Passing `NULL` as second parameter activates the *automatic guess mode*. Here, the scaling factor is recomputed according to a mean free energy of `184.3*length` cal for random sequences.

Note

This recomputation only takes place if the `pf_scale` attribute of the `exp_params` datastructure contained in `vc` has a value below `1.0`.

On the other hand, if the MFE for a sequence is known, it can be used to recompute a more robust scaling factor, since it represents the lowest free energy of the entire ensemble of structures, i.e. the highest Boltzmann factor. To activate this second mode of *automatic adjustment according to MFE*, a pointer to the MFE value needs to be passed as second argument. This value is then taken to compute the scaling factor as $s = exp((sfact * MFE)/kT/length)$, where sfact is an additional scaling weight located in the vrna_md_t datastructure of `exp_params` in `vc`.

The computed scaling factor s will be stored as `pf_scale` attribute of the `exp_params` datastructure in `vc`.

See also

vrna_exp_params_subst(), vrna_md_t, vrna_exp_param_t, vrna_fold_compound_t

Parameters

vc	The fold compound data structure
mfe	A pointer to the MFE (in kcal/mol) or NULL

Notes on Scripting Interface This function is attached to vrna_fc_s objects as overloaded **exp_params_rescale()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_rescale(), i.e. default scaling of the partition function. Passing an energy in kcal/mol, e.g. as retrieved by a previous call to the *mfe()* method, instructs all subsequent calls to scale the partition function accordingly.

13.6.4.9 void vrna_params_reset (vrna_fold_compound_t * *vc*, vrna_md_t * *md_p*)

```
#include <ViennaRNA/params.h>
```

Reset free energy parameters within a vrna_fold_compound_t according to provided, or default model details.

This function allow to rescale free energy parameters for subsequent structure prediction or evaluation according to a set of model details, e.g. temperature values. To do so, the caller provides either a pointer to a set of model details to be used for rescaling, or NULL if global default setting should be used.

See also

vrna_exp_params_reset(), vrna_params_subs()

Parameters

vc	The fold compound data structure
md_p	A pointer to the new model details (or NULL for reset to defaults)

Notes on Scripting Interface This function is attached to vrna_fc_s objects as overloaded **params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

13.6.4.10 vrna_exp_params_reset (vrna_fold_compound_t * *vc*, vrna_md_t * *md_p*)

```
#include <ViennaRNA/params.h>
```

Reset Boltzmann factors for partition function computations within a vrna_fold_compound_t according to provided, or default model details.

This function allow to rescale Boltzmann factors for subsequent prartition function computations according to a set of model details, e.g. temperature values. To do so, the caller provides either a pointer to a set of model details to be used for rescaling, or NULL if global default setting should be used.

See also

vrna_params_reset(), vrna_exp_params_subst(), vrna_exp_params_rescale()

Parameters

vc	The fold compound data structure
md_p	A pointer to the new model details (or NULL for reset to defaults)

Notes on Scripting Interface This function is attached to vrna_fc_s objects as overloaded **exp_params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

13.6.4.11 vrna_exp_param_t* get_scaled_pf_parameters (void)

```
#include <ViennaRNA/params.h>
```

get a datastructure of type vrna_exp_param_t which contains the Boltzmann weights of several energy parameters scaled according to the current temperature

Deprecated Use vrna_exp_params() instead!

Returns

The datastructure containing Boltzmann weights for use in partition function calculations

13.6.4.12 vrna_exp_param_t* get_boltzmann_factors (double *temperature*, double *betaScale*, vrna_md_t *md*, double *pf_scale*)

```
#include <ViennaRNA/params.h>
```

Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.

This function returns a data structure that contains all necessary precalculated Boltzmann factors for each loop type contribution.
In contrast to get_scaled_pf_parameters(), this function enables setting of independent temperatures for both, the individual energy contributions as well as the thermodynamic temperature used in $exp(-\Delta G/kT)$

Deprecated Use vrna_exp_params() instead!

See also

get_scaled_pf_parameters(), get_boltzmann_factor_copy()

Parameters

temperature	The temperature in degrees Celcius used for (re-)scaling the energy contributions
betaScale	A scaling value that is used as a multiplication factor for the absolute temperature of the system
md	The model details to be used
pf_scale	The scaling factor for the Boltzmann factors

Returns

A set of precomputed Boltzmann factors

13.6.4.13 vrna_exp_param_t* get_boltzmann_factor_copy (vrna_exp_param_t * *parameters*)

```
#include <ViennaRNA/params.h>
```

Get a copy of already precomputed Boltzmann factors.

Deprecated Use vrna_exp_params_copy() instead!

See also

get_boltzmann_factors(), get_scaled_pf_parameters()

Parameters

parameters	The input data structure that shall be copied

Returns

A copy of the provided Boltzmann factor dataset

13.6.4.14 vrna_exp_param_t* get_scaled_alipf_parameters (unsigned int *n_seq*)

```
#include <ViennaRNA/params.h>
```

Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)

Deprecated Use vrna_exp_params_comparative() instead!

13.6.4.15 vrna_exp_param_t* get_boltzmann_factors_ali (unsigned int *n_seq*, double *temperature*, double *betaScale*, vrna_md_t *md*, double *pf_scale*)

```
#include <ViennaRNA/params.h>
```

Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.

Deprecated Use vrna_exp_params_comparative() instead!

13.6.4.16 vrna_param_t* scale_parameters (void)

```
#include <ViennaRNA/params.h>
```

Get precomputed energy contributions for all the known loop types.

Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See get_scaled_parameters() for a completely threadsafe implementation.

Deprecated Use vrna_params() instead!

Returns

A set of precomputed energy contributions

13.6.4.17 vrna_param_t* get_scaled_parameters (double *temperature*, vrna_md_t *md*)

```
#include <ViennaRNA/params.h>
```

Get precomputed energy contributions for all the known loop types.

Call this function to retrieve precomputed energy contributions, i.e. scaled according to the temperature passed. Furthermore, this function assumes a data structure that contains the model details as well, such that subsequent folding recursions are able to retrieve the correct model settings

Deprecated Use vrna_params() instead!

See also

vrna_md_t, set_model_details()

Parameters

temperature	The temperature in degrees Celcius
md	The model details

Returns

precomputed energy contributions and model settings

13.7 Manipulation of the Prediction Models

Files

- file model.h

  The model details data structure and its corresponding modifiers.

Data Structures

- struct vrna_md_s

  The data structure that contains the complete model details used throughout the calculations. More...

Macros

- #define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0

  Default temperature for structure prediction and free energy evaluation in $^\circ C$
- #define VRNA_MODEL_DEFAULT_PF_SCALE -1

  Default scaling factor for partition function computations.
- #define VRNA_MODEL_DEFAULT_BETA_SCALE 1.

  Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.
- #define VRNA_MODEL_DEFAULT_DANGLES 2

  Default dangling end model.
- #define VRNA_MODEL_DEFAULT_SPECIAL_HP 1

  Default model behavior for lookup of special tri-, tetra-, and hexa-loops.
- #define VRNA_MODEL_DEFAULT_NO_LP 0

  Default model behavior for so-called 'lonely pairs'.
- #define VRNA_MODEL_DEFAULT_NO_GU 0

  Default model behavior for G-U base pairs.
- #define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0

  Default model behavior for G-U base pairs closing a loop.
- #define VRNA_MODEL_DEFAULT_CIRC 0

  Default model behavior to treat a molecule as a circular RNA (DNA)
- #define VRNA_MODEL_DEFAULT_GQUAD 0

  Default model behavior regarding the treatment of G-Quadruplexes.
- #define VRNA_MODEL_DEFAULT_UNIQ_ML 0

  Default behavior of the model regarding unique multibranch loop decomposition.
- #define VRNA_MODEL_DEFAULT_ENERGY_SET 0

  Default model behavior on which energy set to use.
- #define VRNA_MODEL_DEFAULT_BACKTRACK 1

  Default model behavior with regards to backtracking of structures.
- #define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'

  Default model behavior on what type of backtracking to perform.
- #define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1

  Default model behavior with regards to computing base pair probabilities.
- #define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1

  Default model behavior for the allowed maximum base pair span.
- #define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1

  Default model behavior for the sliding window approach.
- #define VRNA_MODEL_DEFAULT_LOG_ML 0

Default model behavior on how to evaluate the energy contribution of multibranch loops.

- #define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0

  Default model behavior for consensus structure energy evaluation.

- #define VRNA_MODEL_DEFAULT_ALI_RIBO 0

  Default model behavior for consensus structure covariance contribution assessment.

- #define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.

  Default model behavior for weighting the covariance score in consensus structure prediction.

- #define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

  Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.

- #define MAXALPHA 20

  Maximal length of alphabet.

Typedefs

- typedef struct vrna_md_s vrna_md_t

  Typename for the model details data structure vrna_md_s.

Functions

- void vrna_md_set_default (vrna_md_t *md)

  Apply default model details to a provided vrna_md_t data structure.

- void vrna_md_update (vrna_md_t *md)

  Update the model details data structure.

- char * vrna_md_option_string (vrna_md_t *md)

  Get a corresponding commandline parameter string of the options in a vrna_md_t.

- void vrna_md_defaults_reset (vrna_md_t *md_p)

  Reset the global default model details to a specific set of parameters, or their initial values.

- void vrna_md_defaults_temperature (double T)

  Set default temperature for energy evaluation of loops.

- double vrna_md_defaults_temperature_get (void)

  Get default temperature for energy evaluation of loops.

- void vrna_md_defaults_betaScale (double b)

  Set default scaling factor of thermodynamic temperature in Boltzmann factors.

- double vrna_md_defaults_betaScale_get (void)

  Get default scaling factor of thermodynamic temperature in Boltzmann factors.

- void vrna_md_defaults_dangles (int d)

  Set default dangle model for structure prediction.

- int vrna_md_defaults_dangles_get (void)

  Get default dangle model for structure prediction.

- void vrna_md_defaults_special_hp (int flag)

  Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

- int vrna_md_defaults_special_hp_get (void)

  Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

- void vrna_md_defaults_noLP (int flag)

  Set default behavior for prediction of canonical secondary structures.

- int vrna_md_defaults_noLP_get (void)

  Get default behavior for prediction of canonical secondary structures.

- void vrna_md_defaults_noGU (int flag)

  Set default behavior for treatment of G-U wobble pairs.

- int vrna_md_defaults_noGU_get (void)

  Get default behavior for treatment of G-U wobble pairs.
- void vrna_md_defaults_noGUclosure (int flag)

  Set default behavior for G-U pairs as closing pair for loops.
- int vrna_md_defaults_noGUclosure_get (void)

  Get default behavior for G-U pairs as closing pair for loops.
- void vrna_md_defaults_logML (int flag)

  Set default behavior recomputing free energies of multibranch loops using a logarithmic model.
- int vrna_md_defaults_logML_get (void)

  Get default behavior recomputing free energies of multibranch loops using a logarithmic model.
- void vrna_md_defaults_circ (int flag)

  Set default behavior whether input sequences are circularized.
- int vrna_md_defaults_circ_get (void)

  Get default behavior whether input sequences are circularized.
- void vrna_md_defaults_gquad (int flag)

  Set default behavior for treatment of G-Quadruplexes.
- int vrna_md_defaults_gquad_get (void)

  Get default behavior for treatment of G-Quadruplexes.
- void vrna_md_defaults_uniq_ML (int flag)

  Set default behavior for creating additional matrix for unique multibranch loop prediction.
- int vrna_md_defaults_uniq_ML_get (void)

  Get default behavior for creating additional matrix for unique multibranch loop prediction.
- void vrna_md_defaults_energy_set (int e)

  Set default energy set.
- int vrna_md_defaults_energy_set_get (void)

  Get default energy set.
- void vrna_md_defaults_backtrack (int flag)

  Set default behavior for whether to backtrack secondary structures.
- int vrna_md_defaults_backtrack_get (void)

  Get default behavior for whether to backtrack secondary structures.
- void vrna_md_defaults_backtrack_type (char t)

  Set default backtrack type, i.e. which DP matrix is used.
- char vrna_md_defaults_backtrack_type_get (void)

  Get default backtrack type, i.e. which DP matrix is used.
- void vrna_md_defaults_compute_bpp (int flag)

  Set the default behavior for whether to compute base pair probabilities after partition function computation.
- int vrna_md_defaults_compute_bpp_get (void)

  Get the default behavior for whether to compute base pair probabilities after partition function computation.
- void vrna_md_defaults_max_bp_span (int span)

  Set default maximal base pair span.
- int vrna_md_defaults_max_bp_span_get (void)

  Get default maximal base pair span.
- void vrna_md_defaults_min_loop_size (int size)

  Set default minimal loop size.
- int vrna_md_defaults_min_loop_size_get (void)

  Get default minimal loop size.
- void vrna_md_defaults_window_size (int size)

  Set default window size for sliding window structure prediction approaches.
- int vrna_md_defaults_window_size_get (void)

  Get default window size for sliding window structure prediction approaches.
- void vrna_md_defaults_oldAliEn (int flag)

Set default behavior for whether to use old energy model for comparative structure prediction.

- int vrna_md_defaults_oldAliEn_get (void)

  Get default behavior for whether to use old energy model for comparative structure prediction.
- void vrna_md_defaults_ribo (int flag)

  Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.
- int vrna_md_defaults_ribo_get (void)

  Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.
- void vrna_md_defaults_cv_fact (double factor)

  Set the default covariance scaling factor used in comparative structure prediction.
- double vrna_md_defaults_cv_fact_get (void)

  Get the default covariance scaling factor used in comparative structure prediction.
- void vrna_md_defaults_nc_fact (double factor)
- double vrna_md_defaults_nc_fact_get (void)
- void vrna_md_defaults_sfact (double factor)

  Set the default scaling factor used to avoid under-/overflows in partition function computation.
- double vrna_md_defaults_sfact_get (void)

  Get the default scaling factor used to avoid under-/overflows in partition function computation.
- void set_model_details (vrna_md_t *md)

  Set default model details.

Variables

- double temperature

  Rescale energy parameters to a temperature in degC.
- double pf_scale

  A scaling factor used by pf_fold() to avoid overflows.
- int dangles

  Switch the energy model for dangling end contributions (0, 1, 2, 3)
- int tetra_loop

  Include special stabilizing energies for some tri-, tetra- and hexa-loops;.
- int noLonelyPairs

  Global switch to avoid/allow helices of length 1.
- int noGU

  Global switch to forbid/allow GU base pairs at all.
- int no_closingGU

  GU allowed only inside stacks if set to 1.
- int circ

  backward compatibility variable.. this does not effect anything
- int gquad

  Allow G-quadruplex formation.
- int canonicalBPonly
- int uniq_ML

  do ML decomposition uniquely (for subopt)
- int energy_set

  0 = BP; 1=any mit GC; 2=any mit AU-parameter
- int do_backtrack

  do backtracking, i.e. compute secondary structures or base pair probabilities
- char backtrack_type

  A backtrack array marker for inverse_fold()
- char * nonstandards

contains allowed non standard base pairs

- int max_bp_span

  Maximum allowed base pair span.

- int oldAliEn

  use old alifold energies (with gaps)

- int ribo

  use ribosum matrices

- int logML

  if nonzero use logarithmic ML energy in energy_of_struct

13.7.1 Detailed Description

13.7.2 Data Structure Documentation

13.7.2.1 struct vrna_md_s

The data structure that contains the complete model details used throughout the calculations.

For convenience reasons, we provide the type name vrna_md_t to address this data structure without the use of the struct keyword

See also

vrna_md_set_default(), set_model_details(), vrna_md_update(), vrna_md_t

Notes on Scripting Interface

This data structure is wrapped as an object **md** with multiple related functions attached as methods.

A new set of default parameters can be obtained by calling the constructor of **md:**

- *md()* – Initialize with default settings

The resulting object has a list of attached methods which directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *reset()* – vrna_md_set_default()
- *set_from_globals()* – set_model_details()
- *option_string()* – vrna_md_option_string()

Note, that default parameters can be modified by directly setting any of the following global variables. Internally, getting/setting default parameters using their global variable representative translates into calls of the following functions, therefore these wrappers for these functions do not exist in the scripting language interface(s):

global variable	*C* getter	*C* setter
temperature	vrna_md_defaults_temperature_get()	vrna_md_defaults_temperature()
dangles	vrna_md_defaults_dangles_get()	vrna_md_defaults_dangles()
betaScale	vrna_md_defaults_betaScale_get()	vrna_md_defaults_betaScale()

global variable	*C* getter	*C* setter
tetra_loop	this is an alias of *special_hp*	
special_hp	vrna_md_defaults_special_hp_get()	vrna_md_defaults_special_hp()
noLonelyPairs	this is an alias of *noLP*	
noLP	vrna_md_defaults_noLP_get()	vrna_md_defaults_noLP()
noGU	vrna_md_defaults_noGU_get()	vrna_md_defaults_noGU()
no_closingGU	this is an alias of *noGUclosure*	
noGUclosure	vrna_md_defaults_noGUclosure_get()	vrna_md_defaults_noGUclosure()
logML	vrna_md_defaults_logML_get()	vrna_md_defaults_logML()
circ	vrna_md_defaults_circ_get()	vrna_md_defaults_circ()
gquad	vrna_md_defaults_gquad_get()	vrna_md_defaults_gquad()
uniq_ML	vrna_md_defaults_uniq_ML_get()	vrna_md_defaults_uniq_ML()
energy_set	vrna_md_defaults_energy_set_get()	vrna_md_defaults_energy_set()
backtrack	vrna_md_defaults_backtrack_get()	vrna_md_defaults_backtrack()
backtrack_type	vrna_md_defaults_backtrack_type_get()	vrna_md_defaults_backtrack_type()
do_backtrack	this is an alias of *compute_bpp*	
compute_bpp	vrna_md_defaults_compute_bpp_get()	vrna_md_defaults_compute_bpp()
max_bp_span	vrna_md_defaults_max_bp_span_get()	vrna_md_defaults_max_bp_span()
min_loop_size	vrna_md_defaults_min_loop_size_get()	vrna_md_defaults_min_loop_size()
window_size	vrna_md_defaults_window_size_get()	vrna_md_defaults_window_size()
oldAliEn	vrna_md_defaults_oldAliEn_get()	vrna_md_defaults_oldAliEn()
ribo	vrna_md_defaults_ribo_get()	vrna_md_defaults_ribo()
cv_fact	vrna_md_defaults_cv_fact_get()	vrna_md_defaults_cv_fact()
nc_fact	vrna_md_defaults_nc_fact_get()	vrna_md_defaults_nc_fact()
sfact	vrna_md_defaults_sfact_get()	vrna_md_defaults_sfact()

Data Fields

- double temperature

  The temperature used to scale the thermodynamic parameters.
- double betaScale

A scaling factor for the thermodynamic temperature of the Boltzmann factors.

- int dangles

  Specifies the dangle model used in any energy evaluation (0,1,2 or 3)
- int special_hp

  Include special hairpin contributions for tri, tetra and hexaloops.
- int noLP

  Only consider canonical structures, i.e. no 'lonely' base pairs.
- int noGU

  Do not allow GU pairs.
- int noGUclosure

  Do not allow loops to be closed by GU pair.
- int logML

  Use logarithmic scaling for multi loops.
- int circ

  Assume RNA to be circular instead of linear.
- int gquad

  Include G-quadruplexes in structure prediction.
- int canonicalBPonly

  remove non-canonical bp's from constraint structures
- int uniq_ML

  Flag to ensure unique multibranch loop decomposition during folding.
- int energy_set

  Specifies the energy set that defines set of compatible base pairs.
- int backtrack

  Specifies whether or not secondary structures should be backtraced.
- char backtrack_type

  Specifies in which matrix to backtrack.
- int compute_bpp

  Specifies whether or not backward recursions for base pair probability (bpp) computation will be performed.
- char nonstandards [33]

  contains allowed non standard bases
- int max_bp_span

  maximum allowed base pair span
- int min_loop_size

  Minimum size of hairpin loops.
- int window_size

  Size of the sliding window for locally optimal structure predition.
- int oldAliEn

  Use old alifold energy model.
- int ribo

  Use ribosum scoring table in alifold energy model.
- double cv_fact

  Covariance scaling factor for consensus structure prediction.
- double nc_fact

  Scaling factor to weight covariance contributions of non-canonical pairs.
- double sfact

  Scaling factor for partition function scaling.
- int rtype [8]

  Reverse base pair type array.
- short alias [MAXALPHA+1]

  alias of an integer nucleotide representation
- int pair [MAXALPHA+1][MAXALPHA+1]

  Integer representation of a base pair.

13.7.2.1.1 Field Documentation

13.7.2.1.1.1 int vrna_md_s::dangles

Specifies the dangle model used in any energy evaluation (0,1,2 or 3)

If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (dangles = 1) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm vrna_pf() these checks are neglected. To provide comparability between free energy minimization and partition function algorithms, the default setting is 2. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.
If set to 3 co-axial stacking is explicitly included for adjacent helices in mutli-loops. The option affects only mfe folding and energy evaluation (vrna_mfe() and vrna_eval_structure()), as well as suboptimal folding (vrna_subopt()) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

Note

Some function do not implement all dangle model but only a subset of (0,1,2,3). In particular, partition function algorithms can only handle 0 and 2. Read the documentaion of the particular recurrences or energy evaluation function for information about the provided dangle model.

13.7.2.1.1.2 int vrna_md_s::min_loop_size

Minimum size of hairpin loops.

Note

The default value for this field is TURN, however, it may be 0 in cofolding context.

13.7.3 Macro Definition Documentation

13.7.3.1 #define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0

```
#include <ViennaRNA/model.h>
```

Default temperature for structure prediction and free energy evaluation in $^\circ C$

See also

vrna_md_t.temperature, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.2 #define VRNA_MODEL_DEFAULT_PF_SCALE -1

```
#include <ViennaRNA/model.h>
```

Default scaling factor for partition function computations.

See also

vrna_exp_param_t.pf_scale, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.3 #define VRNA_MODEL_DEFAULT_BETA_SCALE 1.

```
#include <ViennaRNA/model.h>
```

Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.

See also

vrna_exp_param_t.alpha, vrna_md_t.betaScale, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.4 #define VRNA_MODEL_DEFAULT_DANGLES 2

```
#include <ViennaRNA/model.h>
```

Default dangling end model.

See also

vrna_md_t.dangles, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.5 #define VRNA_MODEL_DEFAULT_SPECIAL_HP 1

```
#include <ViennaRNA/model.h>
```

Default model behavior for lookup of special tri-, tetra-, and hexa-loops.

See also

vrna_md_t.special_hp, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.6 #define VRNA_MODEL_DEFAULT_NO_LP 0

```
#include <ViennaRNA/model.h>
```

Default model behavior for so-called 'lonely pairs'.

See also

vrna_md_t.noLP, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.7 #define VRNA_MODEL_DEFAULT_NO_GU 0

```
#include <ViennaRNA/model.h>
```

Default model behavior for G-U base pairs.

See also

vrna_md_t.noGU, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.8 #define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0

```
#include <ViennaRNA/model.h>
```

Default model behavior for G-U base pairs closing a loop.

See also

vrna_md_t.noGUclosure, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.9 #define VRNA_MODEL_DEFAULT_CIRC 0

```
#include <ViennaRNA/model.h>
```

Default model behavior to treat a molecule as a circular RNA (DNA)

See also

vrna_md_t.circ, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.10 #define VRNA_MODEL_DEFAULT_GQUAD 0

```
#include <ViennaRNA/model.h>
```

Default model behavior regarding the treatment of G-Quadruplexes.

See also

vrna_md_t.gquad, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.11 #define VRNA_MODEL_DEFAULT_UNIQ_ML 0

```
#include <ViennaRNA/model.h>
```

Default behavior of the model regarding unique multibranch loop decomposition.

See also

vrna_md_t.uniq_ML, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.12 #define VRNA_MODEL_DEFAULT_ENERGY_SET 0

```
#include <ViennaRNA/model.h>
```

Default model behavior on which energy set to use.

See also

vrna_md_t.energy_set, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.13 #define VRNA_MODEL_DEFAULT_BACKTRACK 1

```
#include <ViennaRNA/model.h>
```

Default model behavior with regards to backtracking of structures.

See also

vrna_md_t.backtrack, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.14 #define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'

```
#include <ViennaRNA/model.h>
```

Default model behavior on what type of backtracking to perform.

See also

vrna_md_t.backtrack_type, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.15 #define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1

```
#include <ViennaRNA/model.h>
```

Default model behavior with regards to computing base pair probabilities.

See also

vrna_md_t.compute_bpp, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.16 #define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1

```
#include <ViennaRNA/model.h>
```

Default model behavior for the allowed maximum base pair span.

See also

vrna_md_t.max_bp_span, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.17 #define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1

```
#include <ViennaRNA/model.h>
```

Default model behavior for the sliding window approach.

See also

vrna_md_t.window_size, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.18 #define VRNA_MODEL_DEFAULT_LOG_ML 0

```
#include <ViennaRNA/model.h>
```

Default model behavior on how to evaluate the energy contribution of multibranch loops.

See also

vrna_md_t.logML, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.19 #define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0

```
#include <ViennaRNA/model.h>
```

Default model behavior for consensus structure energy evaluation.

See also

vrna_md_t.oldAliEn, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.20 #define VRNA_MODEL_DEFAULT_ALI_RIBO 0

```
#include <ViennaRNA/model.h>
```

Default model behavior for consensus structure covariance contribution assessment.

See also

vrna_md_t.ribo, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.21 #define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.

```
#include <ViennaRNA/model.h>
```

Default model behavior for weighting the covariance score in consensus structure prediction.

See also

vrna_md_t.cv_fact, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.3.22 #define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

```
#include <ViennaRNA/model.h>
```

Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.

See also

vrna_md_t.nc_fact, vrna_md_defaults_reset(), vrna_md_set_default()

13.7.4 Function Documentation

13.7.4.1 void vrna_md_set_default (vrna_md_t * *md*)

```
#include <ViennaRNA/model.h>
```

Apply default model details to a provided vrna_md_t data structure.

Use this function to initialize a vrna_md_t data structure with its default values

Parameters

md	A pointer to the data structure that is about to be initialized

13.7.4.2 void vrna_md_update (vrna_md_t * *md*)

```
#include <ViennaRNA/model.h>
```

Update the model details data structure.

This function should be called after changing the vrna_md_t.energy_set attribute since it re-initializes base pairing related arrays within the vrna_md_t data structure. In particular, vrna_md_t.pair, vrna_md_t.alias, and vrna_md_t.rtype are set to the values that correspond to the specified vrna_md_t.energy_set option

See also

vrna_md_t, vrna_md_t.energy_set, vrna_md_t.pair, vrna_md_t.rtype, vrna_md_t.alias, vrna_md_set_default()

13.7.4.3 char* vrna_md_option_string (vrna_md_t * *md*)

```
#include <ViennaRNA/model.h>
```

Get a corresponding commandline parameter string of the options in a vrna_md_t.

Note

This function is not threadsafe!

13.7.4.4 void vrna_md_defaults_reset (vrna_md_t * *md_p*)

```
#include <ViennaRNA/model.h>
```

Reset the global default model details to a specific set of parameters, or their initial values.

This function resets the global default model details to their initial values, i.e. as specified by the ViennaRNA Package release, upon passing NULL as argument. Alternatively it resets them according to a set of provided parameters.

Note

The global default parameters affect all function calls of RNAlib where model details are not explicitely provided. Hence, any change of them is not considered threadsafe

Warning

This function first resets the global default settings to factory defaults, and only then applies user provided settings (if any). User settings that do not meet specifications are skipped.

See also

vrna_md_set_default(), vrna_md_t

Parameters

md_p	A set of model details to use as global default (if NULL is passed, factory defaults are restored)

13.7.4.5 void vrna_md_defaults_temperature (double *T*)

```
#include <ViennaRNA/model.h>
```

Set default temperature for energy evaluation of loops.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_TEMPERATURE

Parameters

T	Temperature in centigrade

13.7.4.6 double vrna_md_defaults_temperature_get (void)

```
#include <ViennaRNA/model.h>
```

Get default temperature for energy evaluation of loops.

See also

vrna_md_defaults_temperature(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_TEMPERATURE

Returns

The global default settings for temperature in centigrade

13.7.4.7 void vrna_md_defaults_betaScale (double *b*)

```
#include <ViennaRNA/model.h>
```

Set default scaling factor of thermodynamic temperature in Boltzmann factors.

Bolzmann factors are then computed as $exp(-E/(b \cdot kT))$.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BETA_SCALE

Parameters

b	The scaling factor, default is 1.0

13.7.4.8 double vrna_md_defaults_betaScale_get (void)

```
#include <ViennaRNA/model.h>
```

Get default scaling factor of thermodynamic temperature in Boltzmann factors.

See also

vrna_md_defaults_betaScale(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BETA_SCALE

Returns

The global default thermodynamic temperature scaling factor

13.7.4.9 void vrna_md_defaults_dangles (int *d*)

```
#include <ViennaRNA/model.h>
```

Set default dangle model for structure prediction.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_DANGLES

Parameters

d	The dangle model

13.7.4.10 int vrna_md_defaults_dangles_get (void)

```
#include <ViennaRNA/model.h>
```

Get default dangle model for structure prediction.

See also

vrna_md_defaults_dangles(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_DANGLES

Returns

The global default settings for the dangle model

13.7.4.11 void vrna_md_defaults_special_hp (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_SPECIAL_HP

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.12 int vrna_md_defaults_special_hp_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

See also

vrna_md_defaults_special_hp(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_SPECIAL_HP

Returns

The global default settings for the treatment of special hairpin loops

13.7.4.13 void vrna_md_defaults_noLP (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for prediction of canonical secondary structures.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_LP

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.14 int vrna_md_defaults_noLP_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for prediction of canonical secondary structures.

See also

vrna_md_defaults_noLP(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_LP

Returns

The global default settings for predicting canonical secondary structures

13.7.4.15 void vrna_md_defaults_noGU (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for treatment of G-U wobble pairs.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.16 int vrna_md_defaults_noGU_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for treatment of G-U wobble pairs.

See also

vrna_md_defaults_noGU(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU

Returns

The global default settings for treatment of G-U wobble pairs

13.7.4.17 void vrna_md_defaults_noGUclosure (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for G-U pairs as closing pair for loops.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU_CLOSURE

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.18 int vrna_md_defaults_noGUclosure_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for G-U pairs as closing pair for loops.

See also

vrna_md_defaults_noGUclosure(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU_CLOSURE

Returns

The global default settings for treatment of G-U pairs closing a loop

13.7.4.19 void vrna_md_defaults_logML (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior recomputing free energies of multibranch loops using a logarithmic model.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_LOG_ML

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.20 int vrna_md_defaults_logML_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior recomputing free energies of multibranch loops using a logarithmic model.

See also

vrna_md_defaults_logML(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_LOG_ML

Returns

The global default settings for logarithmic model in multibranch loop free energy evaluation

13.7.4.21 void vrna_md_defaults_circ (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior whether input sequences are circularized.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_CIRC

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.22 int vrna_md_defaults_circ_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior whether input sequences are circularized.

See also

vrna_md_defaults_circ(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_CIRC

Returns

The global default settings for treating input sequences as circular

13.7.4.23 void vrna_md_defaults_gquad (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for treatment of G-Quadruplexes.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_GQUAD

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.24 int vrna_md_defaults_gquad_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for treatment of G-Quadruplexes.

See also

vrna_md_defaults_gquad(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_GQUAD

Returns

The global default settings for treatment of G-Quadruplexes

13.7.4.25 void vrna_md_defaults_uniq_ML (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for creating additional matrix for unique multibranch loop prediction.

Note

Activating this option usually results in higher memory consumption!

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_UNIQ_ML

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.26 int vrna_md_defaults_uniq_ML_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for creating additional matrix for unique multibranch loop prediction.

See also

vrna_md_defaults_uniq_ML(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MOD↩EL_DEFAULT_UNIQ_ML

Returns

The global default settings for creating additional matrices for unique multibranch loop prediction

13.7.4.27 void vrna_md_defaults_energy_set (int *e*)

```
#include <ViennaRNA/model.h>
```

Set default energy set.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ENERGY_SET

Parameters

e	Energy set (0, 1, 2, 3)

13.7.4.28 int vrna_md_defaults_energy_set_get (void)

```
#include <ViennaRNA/model.h>
```

Get default energy set.

See also

vrna_md_defaults_energy_set(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_M↩ODEL_DEFAULT_ENERGY_SET

Returns

The global default settings for the energy set

13.7.4.29 void vrna_md_defaults_backtrack (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for whether to backtrack secondary structures.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BACKTRACK

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.30 int vrna_md_defaults_backtrack_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to backtrack secondary structures.

See also

vrna_md_defaults_backtrack(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BACKTRACK

Returns

The global default settings for backtracking structures

13.7.4.31 void vrna_md_defaults_backtrack_type (char *t*)

```
#include <ViennaRNA/model.h>
```

Set default backtrack type, i.e. which DP matrix is used.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BACKTRACK_TYPE

Parameters

t	The type ('F', 'C', or 'M')

13.7.4.32 char vrna_md_defaults_backtrack_type_get (void)

```
#include <ViennaRNA/model.h>
```

Get default backtrack type, i.e. which DP matrix is used.

See also

vrna_md_defaults_backtrack_type(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BACKTRACK_TYPE

Returns

The global default settings that specify which DP matrix is used for backtracking

13.7.4.33 void vrna_md_defaults_compute_bpp (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set the default behavior for whether to compute base pair probabilities after partition function computation.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_COMPUTE_BPP

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.34 int vrna_md_defaults_compute_bpp_get (void)

```
#include <ViennaRNA/model.h>
```

Get the default behavior for whether to compute base pair probabilities after partition function computation.

See also

vrna_md_defaults_compute_bpp(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_↩ MODEL_DEFAULT_COMPUTE_BPP

Returns

The global default settings that specify whether base pair probabilities are computed together with partition function

13.7.4.35 void vrna_md_defaults_max_bp_span (int *span*)

```
#include <ViennaRNA/model.h>
```

Set default maximal base pair span.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_MAX_BP_SPAN

Parameters

span	Maximal base pair span

13.7.4.36 int vrna_md_defaults_max_bp_span_get (void)

```
#include <ViennaRNA/model.h>
```

Get default maximal base pair span.

See also

vrna_md_defaults_max_bp_span(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_↩ MODEL_DEFAULT_MAX_BP_SPAN

Returns

The global default settings for maximum base pair span

13.7.4.37 void vrna_md_defaults_min_loop_size (int *size*)

```
#include <ViennaRNA/model.h>
```

Set default minimal loop size.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, TURN

Parameters

size	Minimal size, i.e. number of unpaired nucleotides for a hairpin loop

13.7.4.38 int vrna_md_defaults_min_loop_size_get (void)

```
#include <ViennaRNA/model.h>
```

Get default minimal loop size.

See also

vrna_md_defaults_min_loop_size(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, TURN

Returns

The global default settings for minimal size of hairpin loops

13.7.4.39 void vrna_md_defaults_window_size (int *size*)

```
#include <ViennaRNA/model.h>
```

Set default window size for sliding window structure prediction approaches.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_WINDOW_SIZE

Parameters

size	The size of the sliding window

13.7.4.40 int vrna_md_defaults_window_size_get (void)

```
#include <ViennaRNA/model.h>
```

Get default window size for sliding window structure prediction approaches.

See also

vrna_md_defaults_window_size(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_WINDOW_SIZE

Returns

The global default settings for the size of the sliding window

13.7.4.41 void vrna_md_defaults_oldAliEn (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for whether to use old energy model for comparative structure prediction.

Note

This option is outdated. Activating the old energy model usually results in worse consensus structure predictions.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_OLD_EN

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.42 int vrna_md_defaults_oldAliEn_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to use old energy model for comparative structure prediction.

See also

vrna_md_defaults_oldAliEn(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MOD↩EL_DEFAULT_ALI_OLD_EN

Returns

The global default settings for using old energy model for comparative structure prediction

13.7.4.43 void vrna_md_defaults_ribo (int *flag*)

```
#include <ViennaRNA/model.h>
```

Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_RIBO

Parameters

flag	On/Off switch (0 = OFF, else = ON)

13.7.4.44 int vrna_md_defaults_ribo_get (void)

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.

See also

vrna_md_defaults_ribo(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_↩DEFAULT_ALI_RIBO

Returns

The global default settings for using Ribosum scoring in comparative structure prediction

13.7.4.45 void vrna_md_defaults_cv_fact (double *factor*)

```
#include <ViennaRNA/model.h>
```

Set the default covariance scaling factor used in comparative structure prediction.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_CV_FACT

Parameters

factor	The covariance factor

13.7.4.46 double vrna_md_defaults_cv_fact_get (void)

```
#include <ViennaRNA/model.h>
```

Get the default covariance scaling factor used in comparative structure prediction.

See also

vrna_md_defaults_cv_fact(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_CV_FACT

Returns

The global default settings for the covariance factor

13.7.4.47 void vrna_md_defaults_nc_fact (double *factor*)

```
#include <ViennaRNA/model.h>
```

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_NC_FACT

Parameters

factor	

13.7.4.48 double vrna_md_defaults_nc_fact_get (void)

```
#include <ViennaRNA/model.h>
```

See also

vrna_md_defaults_nc_fact(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_NC_FACT

Returns

13.7.4.49 void vrna_md_defaults_sfact (double *factor*)

```
#include <ViennaRNA/model.h>
```

Set the default scaling factor used to avoid under-/overflows in partition function computation.

See also

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t

Parameters

factor	The scaling factor (default: 1.07)

13.7.4.50 double vrna_md_defaults_sfact_get (void)

```
#include <ViennaRNA/model.h>
```

Get the default scaling factor used to avoid under-/overflows in partition function computation.

See also

vrna_md_defaults_sfact(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t

Returns

The global default settings of the scaling factor

13.7.4.51 void set_model_details (vrna_md_t * *md*)

```
#include <ViennaRNA/model.h>
```

Set default model details.

Use this function if you wish to initialize a vrna_md_t data structure with its default values, i.e. the global model settings as provided by the deprecated global variables.

Deprecated This function will vanish as soon as backward compatibility of RNAlib is dropped (expected in version 3). Use vrna_md_set_default() instead!

Parameters

md	A pointer to the data structure that is about to be initialized

13.7.5 Variable Documentation

13.7.5.1 double temperature

```
#include <ViennaRNA/model.h>
```

Rescale energy parameters to a temperature in degC.

Default is 37C. You have to call the update_..._params() functions after changing this parameter.

Deprecated Use vrna_md_defaults_temperature(), and vrna_md_defaults_temperature_get() to change, and read the global default temperature settings

See also

vrna_md_defaults_temperature(), vrna_md_defaults_temperature_get(), vrna_md_defaults_reset()

13.7.5.2 double pf_scale

```
#include <ViennaRNA/model.h>
```

A scaling factor used by pf_fold() to avoid overflows.

Should be set to approximately $exp((-F/kT)/length)$, where F is an estimate for the ensemble free energy, for example the minimum free energy. You must call update_pf_params() after changing this parameter.
If pf_scale is -1 (the default) , an estimate will be provided automatically when computing partition functions, e.g. pf_fold() The automatic estimate is usually insufficient for sequences more than a few hundred bases long.

13.7.5.3 int dangles

```
#include <ViennaRNA/model.h>
```

Switch the energy model for dangling end contributions (0, 1, 2, 3)

If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (dangles = 1) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm pf_fold() these checks are neglected. If dangles is set to 2, all folding routines will follow this convention. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.
If dangles = 3 co-axial stacking is explicitly included for adjacent helices in mutli-loops. The option affects only mfe folding and energy evaluation (fold() and energy_of_structure()), as well as suboptimal folding (subopt()) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

Default is 2 in most algorithms, partition function algorithms can only handle 0 and 2

13.7.5.4 int tetra_loop

```
#include <ViennaRNA/model.h>
```

Include special stabilizing energies for some tri-, tetra- and hexa-loops;.

default is 1.

13.7.5.5 int noLonelyPairs

```
#include <ViennaRNA/model.h>
```

Global switch to avoid/allow helices of length 1.

Disallow all pairs which can only occur as lonely pairs (i.e. as helix of length 1). This avoids lonely base pairs in the predicted structures in most cases.

13.7.5.6 int canonicalBPonly

```
#include <ViennaRNA/model.h>
```

Do not use this variable, it will eventually be removed in one of the next versions

13.7.5.7 int energy_set

```
#include <ViennaRNA/model.h>
```

0 = BP; 1=any mit GC; 2=any mit AU-parameter

If set to 1 or 2: fold sequences from an artificial alphabet ABCD..., where A pairs B, C pairs D, etc. using either GC (1) or AU parameters (2); default is 0, you probably don't want to change it.

13.7.5.8 int do_backtrack

```
#include <ViennaRNA/model.h>
```

do backtracking, i.e. compute secondary structures or base pair probabilities

If 0, do not calculate pair probabilities in pf_fold(); this is about twice as fast. Default is 1.

13.7.5.9 char backtrack_type

```
#include <ViennaRNA/model.h>
```

A backtrack array marker for inverse_fold()

If set to 'C': force (1,N) to be paired, 'M' fold as if the sequence were inside a multi-loop. Otherwise ('F') the usual mfe structure is computed.

13.7.5.10 char* nonstandards

```
#include <ViennaRNA/model.h>
```

contains allowed non standard base pairs

Lists additional base pairs that will be allowed to form in addition to GC, CG, AU, UA, GU and UG. Nonstandard base pairs are given a stacking energy of 0.

13.7.5.11 int max_bp_span

```
#include <ViennaRNA/model.h>
```

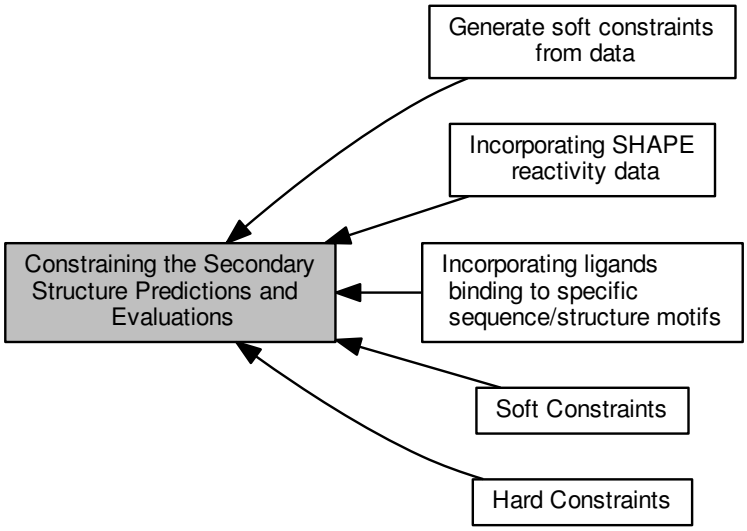
Maximum allowed base pair span.

A value of -1 indicates no restriction for distant base pairs.

13.8 Constraining the Secondary Structure Predictions and Evaluations

This module covers all functions and variables related to the problem of incorporating secondary structure constraints into the folding recursions.

Collaboration diagram for Constraining the Secondary Structure Predictions and Evaluations:



Modules

- Hard Constraints

  This module covers all functionality for hard constraints in secondary structure prediction.
- Soft Constraints

  Functions and data structures for secondary structure soft constraints.
- Incorporating SHAPE reactivity data

  Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.
- Incorporating ligands binding to specific sequence/structure motifs

  This module covers functions that enable the incorporation of ligand binding free energies to specific hairpin/interior loop motifs by means of generic soft constraints.
- Generate soft constraints from data

  Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Files

- file constraints.h

  Functions and data structures for constraining secondary structure predictions and evaluation.

Macros

- #define VRNA_CONSTRAINT_FILE 0

  Flag for vrna_constraints_add() to indicate that constraints are present in a text file.
- #define VRNA_CONSTRAINT_SOFT_MFE 0

  Indicate generation of constraints for MFE folding.
- #define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF

  Indicate generation of constraints for partition function computation.
- #define VRNA_DECOMP_PAIR_HP 1

  Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.
- #define VRNA_DECOMP_PAIR_IL 2

  Indicator for interior loop decomposition step.
- #define VRNA_DECOMP_PAIR_ML 3

  Indicator for multibranch loop decomposition step.
- #define VRNA_DECOMP_ML_ML_ML 5

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_STEM 4

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_ML 6

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_UP 11

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_ML_STEM 20

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_COAXIAL 13

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_EXT_EXT 9

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_UP 8

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_STEM 14

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_EXT_EXT 15

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_STEM_EXT 16

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_STEM_OUTSIDE 17

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_EXT_STEM 18

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_EXT_STEM1 19

  Indicator for decomposition of exterior loop part.

Functions

- void vrna_constraints_add (vrna_fold_compound_t *vc, const char *constraint, unsigned int options)

  Add constraints to a vrna_fold_compound_t data structure.
- void vrna_message_constraint_options (unsigned int option)

  Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)
- void vrna_message_constraint_options_all (void)

  Print structure constraint characters to stdout (full constraint support)

13.8.1 Detailed Description

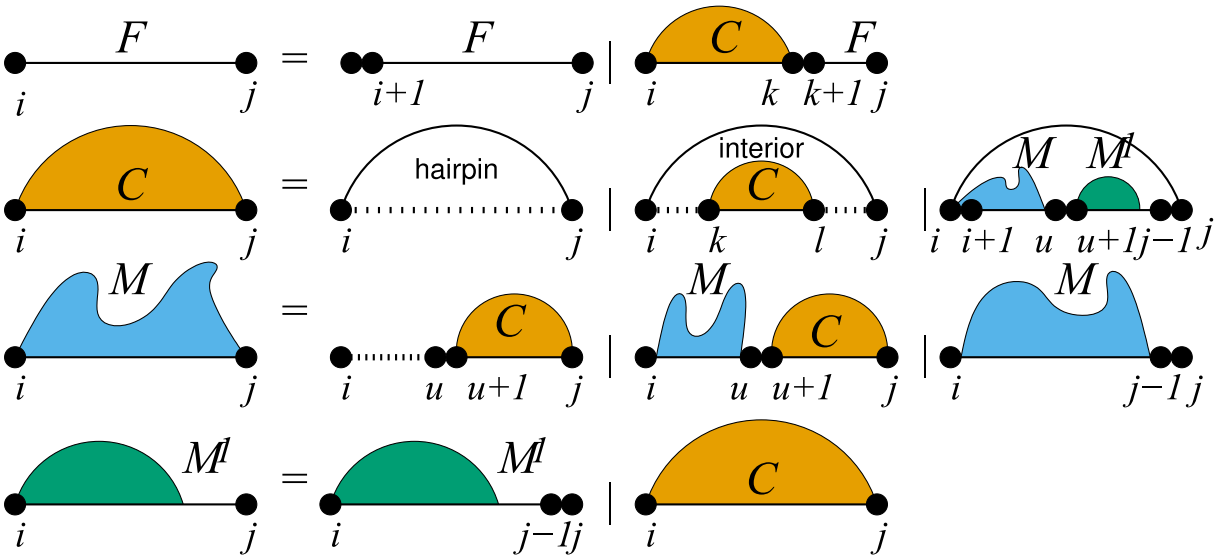
This module covers all functions and variables related to the problem of incorporating secondary structure constraints into the folding recursions.

This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation. Secondary Structure constraints can be subdivided into two groups:

- Hard Constraints, and
- Soft Constraints.

While Hard-Constraints directly influence the production rules used in the folding recursions by allowing, disallowing, or enforcing certain decomposition steps, Soft-constraints on the other hand are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations.

Secondary structure constraints are always applied at decomposition level, i.e. in each step of the recursive structure decomposition, for instance during MFE prediction. Below is a visualization of the decomposition scheme



For Hard Constraints the following option flags may be used to constrain the pairing behavior of single, or pairs of nucleotides:

- VRNA_CONSTRAINT_CONTEXT_EXT_LOOP - Hard constraints flag, base pair in the exterior loop.
- VRNA_CONSTRAINT_CONTEXT_HP_LOOP - Hard constraints flag, base pair encloses hairpin loop.
- VRNA_CONSTRAINT_CONTEXT_INT_LOOP - Hard constraints flag, base pair encloses an interior loop.
- VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC - Hard constraints flag, base pair encloses a multi branch loop.
- VRNA_CONSTRAINT_CONTEXT_MB_LOOP - Hard constraints flag, base pair is enclosed in an interior loop.
- VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC - Hard constraints flag, base pair is enclosed in a multi branch loop.

- VRNA_CONSTRAINT_CONTEXT_ENFORCE - Hard constraint flag to indicate enforcement of constraints.
- VRNA_CONSTRAINT_CONTEXT_NO_REMOVE - Hard constraint flag to indicate not to remove base pairs that conflict with a given constraint.
- VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS - Hard constraints flag, shortcut for all base pairs.

However, for Soft Constraints we do not allow for simple loop type dependent constraining. But soft constraints are equipped with generic constraint support. This enables the user to pass arbitrary callback functions that return auxiliary energy contributions for evaluation the avaluation of any decomposition.

The callback will then always be notified about the type of decomposition that is happening, and the corresponding delimiting sequence positions. The following decomposition steps are distinguished, and should be captured by the user's implementation of the callback:

- VRNA_DECOMP_PAIR_HP - Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.
- VRNA_DECOMP_PAIR_IL - Indicator for interior loop decomposition step.
- VRNA_DECOMP_PAIR_ML - Indicator for multibranch loop decomposition step.
- VRNA_DECOMP_ML_ML_ML - Indicator for decomposition of multibranch loop part.
- VRNA_DECOMP_ML_STEM - Indicator for decomposition of multibranch loop part.
- VRNA_DECOMP_ML_ML - Indicator for decomposition of multibranch loop part.
- VRNA_DECOMP_ML_UP - Indicator for decomposition of multibranch loop part.
- VRNA_DECOMP_ML_ML_STEM - Indicator for decomposition of multibranch loop part.
- VRNA_DECOMP_ML_COAXIAL - Indicator for decomposition of multibranch loop part.
- VRNA_DECOMP_EXT_EXT - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_UP - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_STEM - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_EXT_EXT - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_STEM_EXT - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_STEM_OUTSIDE - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_EXT_STEM - Indicator for decomposition of exterior loop part.
- VRNA_DECOMP_EXT_EXT_STEM1 - Indicator for decomposition of exterior loop part.

Simplified interfaces to the soft constraints framework can be obtained by the implementations in the submodules

- Incorporating SHAPE reactivity data and
- Incorporating ligands binding to specific sequence/structure motifs.

An implementation that generates soft constraints for unpaired nucleotides by minimizing the discrepancy between their predicted and expected pairing probability is available in submodule Generate soft constraints from data.

13.8.2 Macro Definition Documentation

13.8.2.1 #define VRNA_CONSTRAINT_FILE 0

```
#include <ViennaRNA/constraints.h>
```

Flag for vrna_constraints_add() to indicate that constraints are present in a text file.

See also

vrna_constraints_add()

Deprecated Use 0 instead!

13.8.2.2 #define VRNA_CONSTRAINT_SOFT_MFE 0

```
#include <ViennaRNA/constraints.h>
```

Indicate generation of constraints for MFE folding.

Deprecated This flag has no meaning anymore, since constraints are now always stored!

13.8.2.3 #define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF

```
#include <ViennaRNA/constraints.h>
```

Indicate generation of constraints for partition function computation.

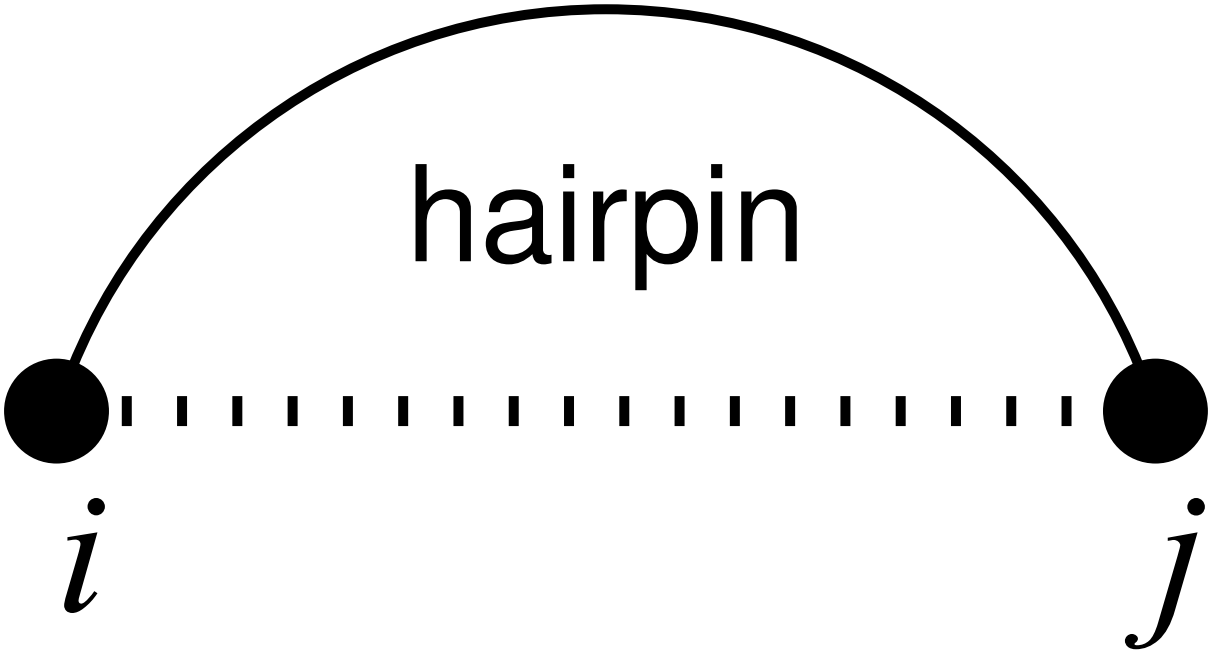
Deprecated Use VRNA_OPTION_PF instead!

13.8.2.4 #define VRNA_DECOMP_PAIR_HP 1

```
#include <ViennaRNA/constraints.h>
```

Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a hairpin loop enclosed by the base pair (i, j).

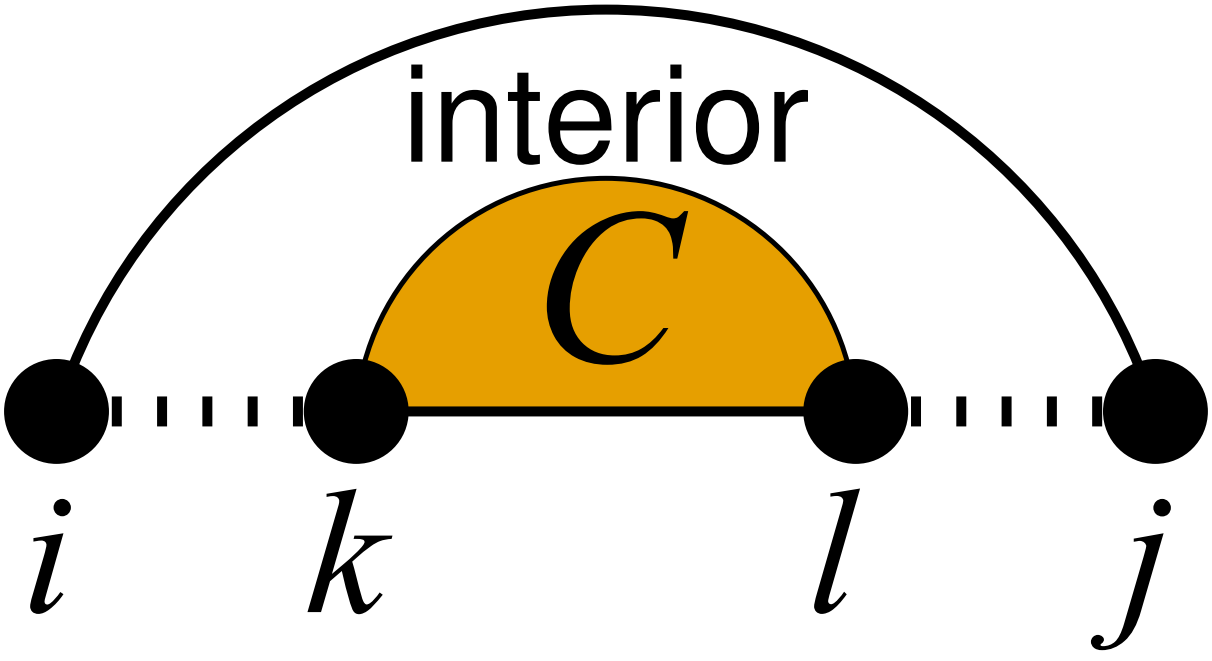


13.8.2.5 #define VRNA_DECOMP_PAIR_IL 2

```
#include <ViennaRNA/constraints.h>
```

Indicator for interior loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an interior loop enclosed by the base pair (i, j), and enclosing the base pair (k, l).

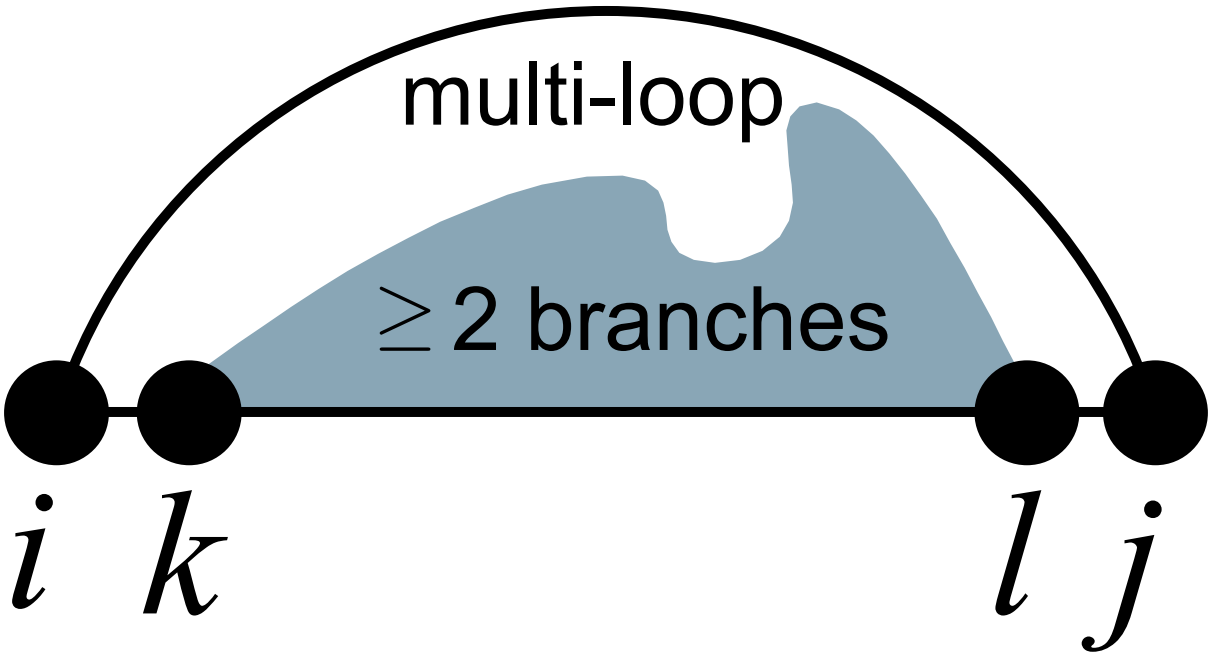


13.8.2.6 #define VRNA_DECOMP_PAIR_ML 3

```
#include <ViennaRNA/constraints.h>
```

Indicator for multibranch loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop enclosed by the base pair (i, j), and consisting of some enclosed multi loop content from k to l.

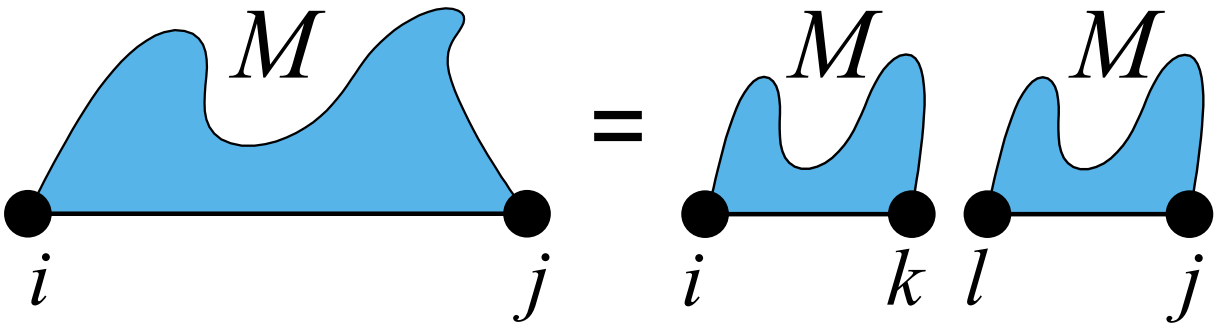


13.8.2.7 #define VRNA_DECOMP_ML_ML_ML 5

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, which will be decomposed into two multibranch loop parts $[i : k]$, and $[l : j]$.

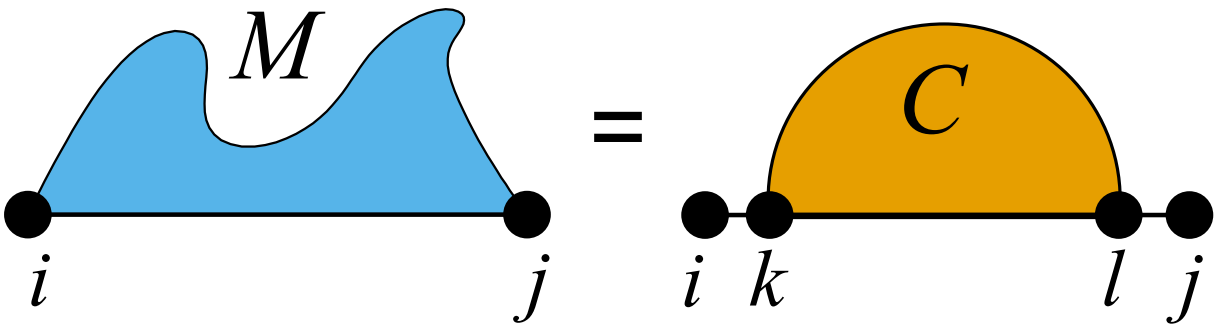


13.8.2.8 #define VRNA_DECOMP_ML_STEM 4

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, which will be considered a single stem branching off with base pair (k, l).

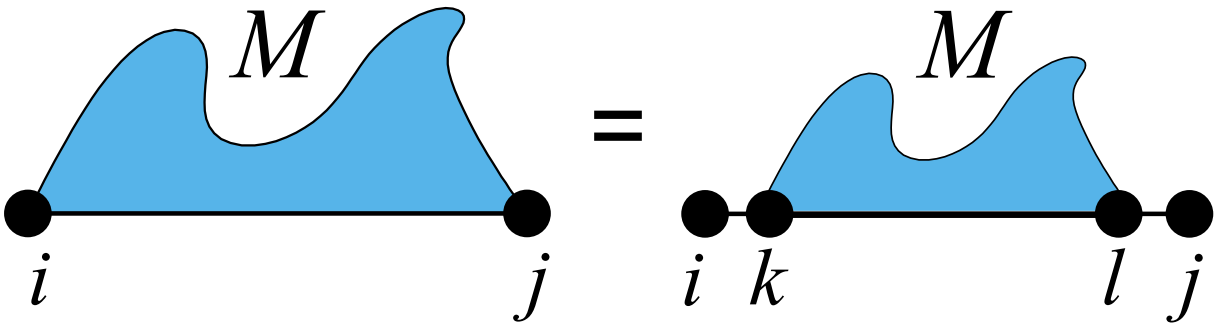


13.8.2.9 #define VRNA_DECOMP_ML_ML 6

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, which will be decomposed into a (usually) smaller multibranch loop part $[k : l]$.

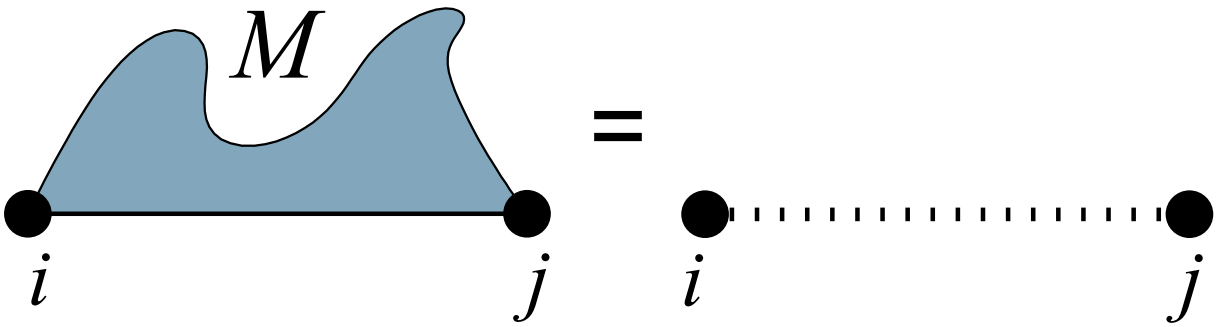


13.8.2.10 #define VRNA_DECOMP_ML_UP 11

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, which will be considered a multibranch loop part that only consists of unpaired nucleotides.

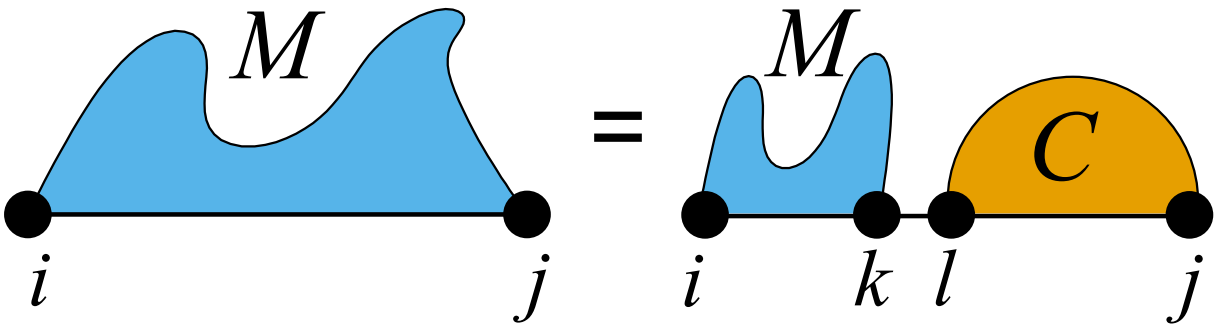


13.8.2.11 #define VRNA_DECOMP_ML_ML_STEM 20

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, which will decomposed into a multibranch loop part $[i : k]$, and a stem with enclosing base pair (l, j).

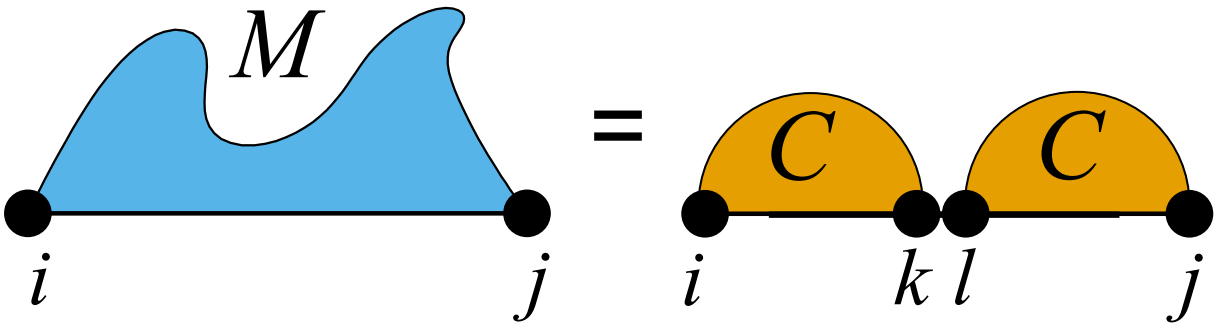


13.8.2.12 #define VRNA_DECOMP_ML_COAXIAL 13

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, where two stems with enclosing pairs (i, k) and (l, j) are coaxially stacking onto each other.

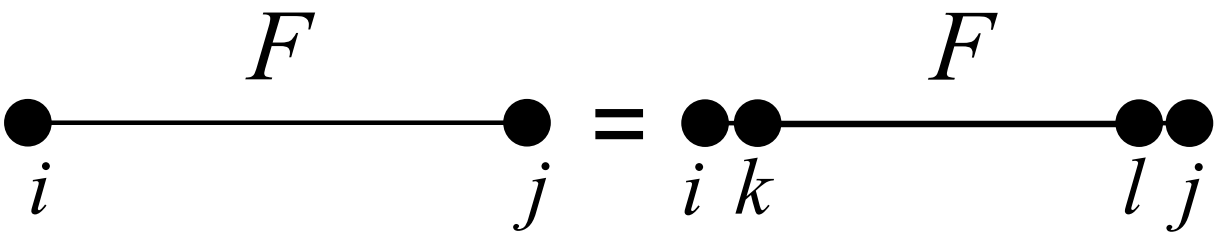


13.8.2.13 #define VRNA_DECOMP_EXT_EXT 9

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into a (usually) smaller exterior loop part $[k : l]$.

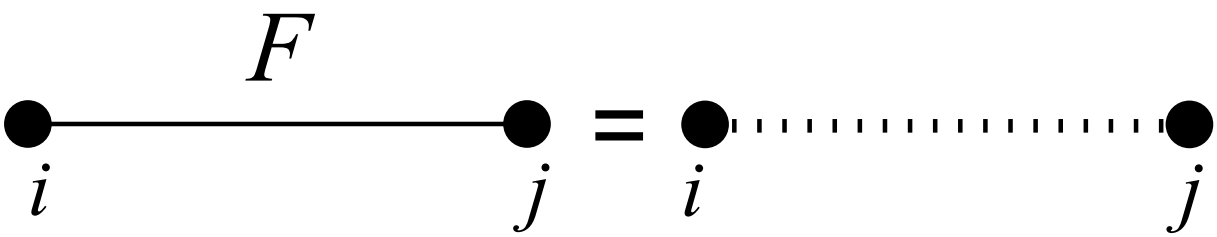


13.8.2.14 #define VRNA_DECOMP_EXT_UP 8

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be considered as an exterior loop component consisting of only unpaired nucleotides.

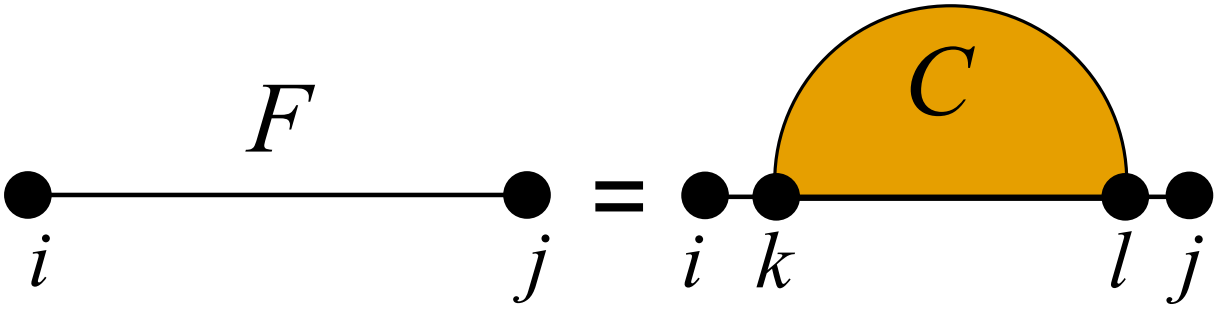


13.8.2.15 #define VRNA_DECOMP_EXT_STEM 14

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be considered a stem with enclosing pair (k, l).

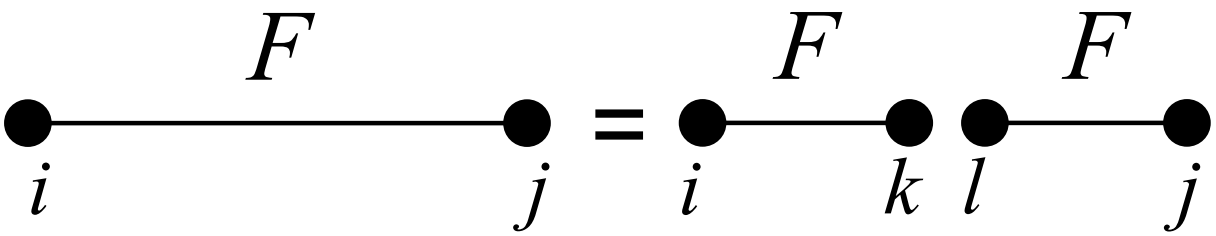


13.8.2.16 #define VRNA_DECOMP_EXT_EXT_EXT 15

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into two exterior loop parts $[i : k]$ and $[l : j]$.

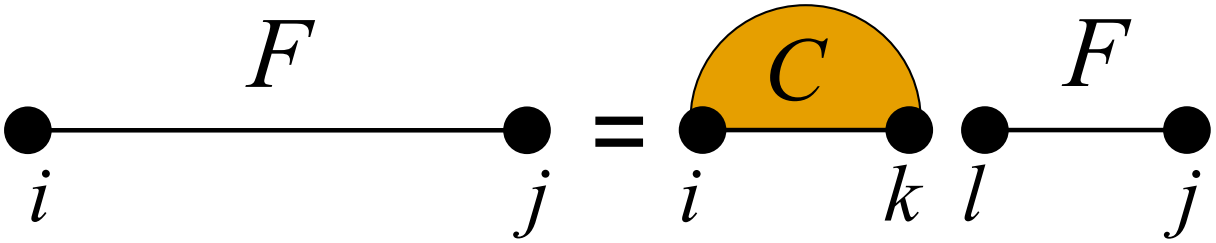


13.8.2.17 #define VRNA_DECOMP_EXT_STEM_EXT 16

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into a stem branching off with base pair (i, k), and an exterior loop part $[l : j]$.

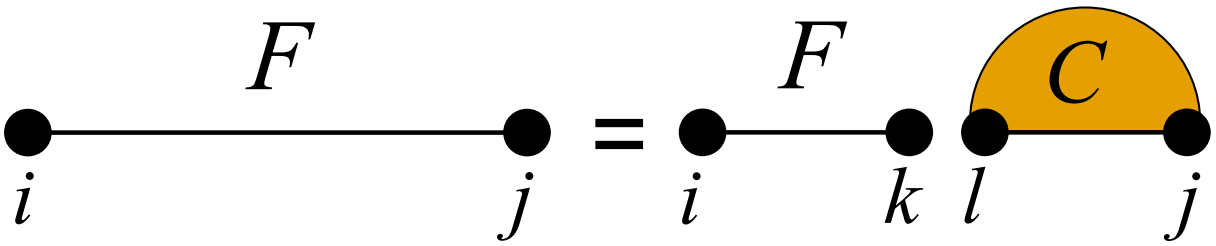


13.8.2.18 #define VRNA_DECOMP_EXT_EXT_STEM 18

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into an exterior loop part $[i : k]$, and a stem branching off with base pair (l, j).

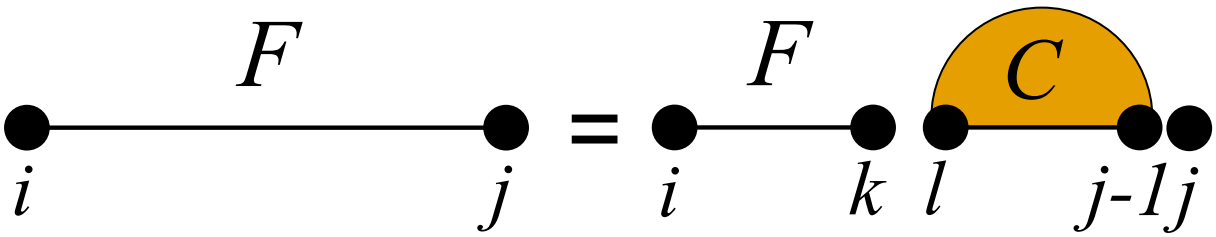


13.8.2.19 #define VRNA_DECOMP_EXT_EXT_STEM1 19

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into an exterior loop part $[i : k]$, and a stem branching off with base pair $(l, j - 1)$.



13.8.3 Function Documentation

13.8.3.1 void vrna_constraints_add (vrna_fold_compound_t * *vc*, const char * *constraint*, unsigned int *options*)

```
#include <ViennaRNA/constraints.h>
```

Add constraints to a vrna_fold_compound_t data structure.

Use this function to add/update the hard/soft constraints The function allows for passing a string 'constraint' that can either be a filename that points to a constraints definition file or it may be a pseudo dot-bracket notation indicating hard constraints. For the latter, the user has to pass the VRNA_CONSTRAINT_DB option. Also, the user has to specify, which characters are allowed to be interpreted as constraints by passing the corresponding options via the third parameter.

See also

vrna_hc_init(), vrna_hc_add_up(), vrna_hc_add_up_batch(), vrna_hc_add_bp(), vrna_sc_init(), vrna_sc_add_up(), vrna_sc_add_bp(), vrna_sc_add_SHAPE_deigan(), vrna_sc_add_SHAPE_zarringhalam(), vrna_hc_free(), vrna_sc_free(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_DEFAULT, VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTRAMOL, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_GQUAD

The following is an example for adding hard constraints given in pseudo dot-bracket notation. Here, `vc` is the vrna_fold_compound_t object, `structure` is a char array with the hard constraint in dot-bracket notation, and `enforceConstraints` is a flag indicating whether or not constraints for base pairs should be enforced instead of just doing a removal of base pair that conflict with the constraint.

```
unsigned int constraint_options = VRNA_CONSTRAINT_DB_DEFAULT;
if(enforceConstraints)
  constraint_options |= VRNA_CONSTRAINT_DB_ENFORCE_BP;
vrna_constraints_add(vc, (const char *)structure, constraint_options);
```

In constrat to the above, constraints may also be read from file:

```
  vrna_constraints_add(vc, constraints_file,
VRNA_OPTION_MFE | ((pf) ? VRNA_OPTION_PF : 0));
```

See also

vrna_hc_add_from_db(), vrna_hc_add_up(), vrna_hc_add_up_batch() vrna_hc_add_bp_unspecific(), vrna_hc_add_bp()

Parameters

vc	The fold compound
constraint	A string with either the filename of the constraint definitions or a pseudo dot-bracket notation of the hard constraint. May be NULL.
options	The option flags

13.8.3.2 void vrna_message_constraint_options (unsigned int *option*)

```
#include <ViennaRNA/constraints_hard.h>
```

Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)

Currently available options are:
VRNA_CONSTRAINT_DB_PIPE (paired with another base)
VRNA_CONSTRAINT_DB_DOT (no constraint at all)
VRNA_CONSTRAINT_DB_X (base must not pair)
VRNA_CONSTRAINT_DB_ANG_BRACK (paired downstream/upstream)
VRNA_CONSTRAINT_DB_RND_BRACK (base i pairs base j)
pass a collection of options as one value like this:

```
vrna_message_constraints(option_1 | option_2 | option_n)
```

See also

vrna_message_constraint_options_all(), vrna_constraints_add(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_INTRAMOL

Parameters

option	Option switch that tells which constraint help will be printed

13.8.3.3 void vrna_message_constraint_options_all (void)

```
#include <ViennaRNA/constraints_hard.h>
```

Print structure constraint characters to stdout (full constraint support)

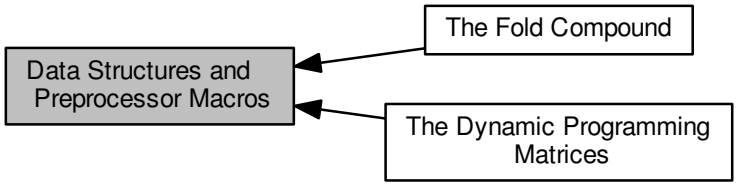
See also

vrna_message_constraint_options(), vrna_constraints_add(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_INTRAMOL

13.9 Data Structures and Preprocessor Macros

Most datastructures and typedefs shared among the ViennaRNA Package can be found here.

Collaboration diagram for Data Structures and Preprocessor Macros:



Modules

- The Fold Compound

  This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.
- The Dynamic Programming Matrices

  This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

Data Structures

- struct vrna_basepair_s

  Base pair data structure used in subopt.c. More...
- struct vrna_plist_s

  this datastructure is used as input parameter in functions of PS_dot.h and others More...
- struct vrna_cpair_s

  this datastructure is used as input parameter in functions of PS_dot.c More...
- struct vrna_sect_s

  Stack of partial structures for backtracking. More...
- struct vrna_bp_stack_s

  Base pair stack element. More...
- struct pu_contrib

  contributions to p_u More...
- struct interact

  interaction data structure for RNAup More...
- struct pu_out

  Collection of all free_energy of beeing unpaired values for output. More...
- struct constrain

  constraints for cofolding More...
- struct duplexT

  Data structure for RNAduplex. More...

- struct node

  Data structure for RNAsnoop (fold energy list) More...
- struct snoopT

  Data structure for RNAsnoop. More...
- struct dupVar

  Data structure used in RNApkplex. More...

Typedefs

- typedef struct vrna_basepair_s vrna_basepair_t

  Typename for the base pair repesenting data structure vrna_basepair_s.
- typedef struct vrna_plist_s vrna_plist_t

  Typename for the base pair list repesenting data structure vrna_plist_s.
- typedef struct vrna_bp_stack_s vrna_bp_stack_t

  Typename for the base pair stack repesenting data structure vrna_bp_stack_s.
- typedef struct vrna_cpair_s vrna_cpair_t

  Typename for data structure vrna_cpair_s.
- typedef struct vrna_sect_s vrna_sect_t

  Typename for stack of partial structures vrna_sect_s.
- typedef double FLT_OR_DBL

  Typename for floating point number in partition function computations.
- typedef struct vrna_basepair_s PAIR

  Old typename of vrna_basepair_s.
- typedef struct vrna_plist_s plist

  Old typename of vrna_plist_s.
- typedef struct vrna_cpair_s cpair

  Old typename of vrna_cpair_s.
- typedef struct vrna_sect_s sect

  Old typename of vrna_sect_s.
- typedef struct vrna_bp_stack_s bondT

  Old typename of vrna_bp_stack_s.
- typedef struct pu_contrib pu_contrib

  contributions to p_u
- typedef struct interact interact

  interaction data structure for RNAup
- typedef struct pu_out pu_out

  Collection of all free_energy of beeing unpaired values for output.
- typedef struct constrain constrain

  constraints for cofolding
- typedef struct node folden

  Data structure for RNAsnoop (fold energy list)
- typedef struct dupVar dupVar

  Data structure used in RNApkplex.

13.9.1 Detailed Description

Most datastructures and typedefs shared among the ViennaRNA Package can be found here.

13.9.2 Data Structure Documentation

13.9.2.1 struct vrna_basepair_s

Base pair data structure used in subopt.c.

13.9.2.2 struct vrna_plist_s

this datastructure is used as input parameter in functions of PS_dot.h and others

13.9.2.3 struct vrna_cpair_s

this datastructure is used as input parameter in functions of PS_dot.c

13.9.2.4 struct vrna_sect_s

Stack of partial structures for backtracking.

13.9.2.5 struct vrna_bp_stack_s

Base pair stack element.

13.9.2.6 struct pu_contrib

contributions to p_u

Data Fields

- double ** H
  hairpin loops
- double ** I
  interior loops
- double ** M
  multi loops
- double ** E
  exterior loop
- int length
  length of the input sequence
- int w
  longest unpaired region

13.9.2.7 struct interact

interaction data structure for RNAup

Data Fields

- double * Pi
  probabilities of interaction
- double * Gi
  free energies of interaction
- double Gikjl
  full free energy for interaction between [k,i] k<i in longer seq and [j,l] j<l in shorter seq
- double Gikjl_wo
  Gikjl without contributions for prob_unpaired.
- int i
  k<i in longer seq
- int k
  k<i in longer seq
- int j
  j<l in shorter seq
- int l
  j<l in shorter seq
- int length
  length of longer sequence

13.9.2.8 struct pu_out

Collection of all free_energy of beeing unpaired values for output.

Data Fields

- int len
  sequence length
- int u_vals
  number of different -u values
- int contribs
  [-c "SHIME"]
- char ** header
  header line
- double ** u_values
  *(the -u values * [-c "SHIME"]) * seq len*

13.9.2.9 struct constrain

constraints for cofolding

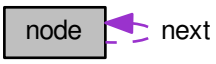
13.9.2.10 struct duplexT

Data structure for RNAduplex.

13.9.2.11 struct node

Data structure for RNAsnoop (fold energy list)

Collaboration diagram for node:



13.9.2.12 struct snoopT

Data structure for RNAsnoop.

13.9.2.13 struct dupVar

Data structure used in RNApkplex.

13.9.3 Typedef Documentation

13.9.3.1 typedef struct vrna_basepair_s PAIR

```
#include <ViennaRNA/data_structures.h>
```

Old typename of vrna_basepair_s.

Deprecated Use vrna_basepair_t instead!

13.9.3.2 typedef struct vrna_plist_s plist

```
#include <ViennaRNA/data_structures.h>
```

Old typename of vrna_plist_s.

Deprecated Use vrna_plist_t instead!

13.9.3.3 typedef struct vrna_cpair_s cpair

```
#include <ViennaRNA/data_structures.h>
```

Old typename of vrna_cpair_s.

Deprecated Use vrna_cpair_t instead!

13.9.3.4 typedef struct vrna_sect_s sect

```
#include <ViennaRNA/data_structures.h>
```

Old typename of vrna_sect_s.

Deprecated Use vrna_sect_t instead!

13.9.3.5 typedef struct vrna_bp_stack_s bondT

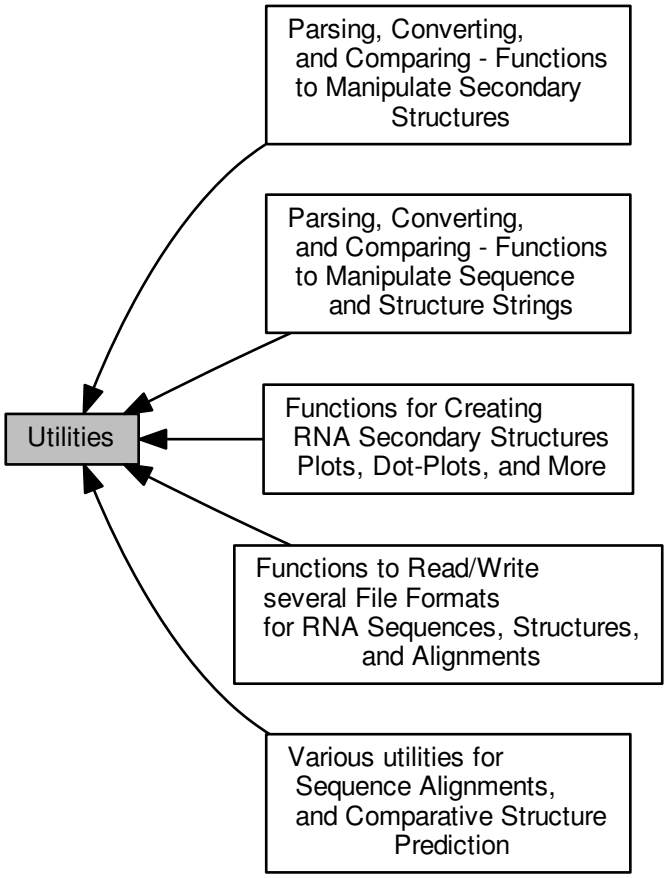
```
#include <ViennaRNA/data_structures.h>
```

Old typename of vrna_bp_stack_s.

Deprecated Use vrna_bp_stack_t instead!

13.10 Utilities

Collaboration diagram for Utilities:



Modules

- Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings
- Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures
- Various utilities for Sequence Alignments, and Comparative Structure Prediction
- Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments
- Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More

Files

- file alphabet.h

  Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.
- file utils.h

  General utility- and helper-functions used throughout the ViennaRNA Package.

Macros

- #define VRNA_INPUT_ERROR 1U

  Output flag of get_input_line(): "An ERROR has occured, maybe EOF".
- #define VRNA_INPUT_QUIT 2U

  Output flag of get_input_line(): "the user requested quitting the program".
- #define VRNA_INPUT_MISC 4U

  Output flag of get_input_line(): "something was read".
- #define VRNA_INPUT_FASTA_HEADER 8U

  Input/Output flag of get_input_line():
  if used as input option this tells get_input_line() that the data to be read should comply with the FASTA format.
- #define VRNA_INPUT_CONSTRAINT 32U

  Input flag for get_input_line():
  Tell get_input_line() that we assume to read a structure constraint.
- #define VRNA_INPUT_NO_TRUNCATION 256U

  Input switch for get_input_line(): "do not trunkate the line by eliminating white spaces at end of line".
- #define VRNA_INPUT_NO_REST 512U

  Input switch for vrna_file_fasta_read_record(): "do fill rest array".
- #define VRNA_INPUT_NO_SPAN 1024U

  Input switch for vrna_file_fasta_read_record(): "never allow data to span more than one line".
- #define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U

  Input switch for vrna_file_fasta_read_record(): "do not skip empty lines".
- #define VRNA_INPUT_BLANK_LINE 4096U

  Output flag for vrna_file_fasta_read_record(): "read an empty line".
- #define VRNA_INPUT_NOSKIP_COMMENTS 128U

  Input switch for get_input_line(): "do not skip comment lines".
- #define VRNA_INPUT_COMMENT 8192U

  Output flag for vrna_file_fasta_read_record(): "read a comment".
- #define MIN2(A, B) ((A) < (B) ? (A) : (B))

  Get the minimum of two comparable values.
- #define MAX2(A, B) ((A) > (B) ? (A) : (B))

  Get the maximum of two comparable values.
- #define MIN3(A, B, C) (MIN2((MIN2((A),(B))) ,(C)))

  Get the minimum of three comparable values.
- #define MAX3(A, B, C) (MAX2((MAX2((A),(B))) ,(C)))

  Get the maximum of three comparable values.

Functions

- void * vrna_alloc (unsigned size)

  Allocate space safely.
- void * vrna_realloc (void *p, unsigned size)

  Reallocate space safely.
- void vrna_message_error (const char message[ ])

  Die with an error message.
- void vrna_message_warning (const char message[ ])

  Print a warning message.
- void vrna_init_rand (void)

  Initialize seed for random number generator.
- double vrna_urn (void)

  get a random number from [0..1]

- int vrna_int_urn (int from, int to)

  Generates a pseudo random integer in a specified range.
- void vrna_file_copy (FILE *from, FILE *to)

  Inefficient 'cp'.
- char * vrna_time_stamp (void)

  Get a timestamp.
- char * get_line (FILE *fp)

  Read a line of arbitrary length from a stream.
- unsigned int get_input_line (char **string, unsigned int options)
- void vrna_message_input_seq_simple (void)

  Print a line to stdout that asks for an input sequence.
- void vrna_message_input_seq (const char *s)

  Print a line with a user defined string and a ruler to stdout.
- int * vrna_idx_row_wise (unsigned int length)

  Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.
- int * vrna_idx_col_wise (unsigned int length)

  Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.

Variables

- unsigned short xsubi [3]

  Current 48 bit random number.

13.10.1 Detailed Description

13.10.2 Macro Definition Documentation

13.10.2.1 #define VRNA_INPUT_FASTA_HEADER 8U

```
#include <ViennaRNA/utils.h>
```

Input/Output flag of get_input_line():
if used as input option this tells get_input_line() that the data to be read should comply with the FASTA format.

the function will return this flag if a fasta header was read

13.10.2.2 #define VRNA_INPUT_CONSTRAINT 32U

```
#include <ViennaRNA/utils.h>
```

Input flag for get_input_line():
Tell get_input_line() that we assume to read a structure constraint.

13.10.3 Function Documentation

13.10.3.1 void* vrna_alloc (unsigned *size*)

```
#include <ViennaRNA/utils.h>
```

Allocate space safely.

Parameters

size	The size of the memory to be allocated in bytes

Returns

A pointer to the allocated memory

13.10.3.2 void* vrna_realloc (void * *p*, unsigned *size*)

```
#include <ViennaRNA/utils.h>
```

Reallocate space safely.

Parameters

p	A pointer to the memory region to be reallocated
size	The size of the memory to be allocated in bytes

Returns

A pointer to the newly allocated memory

13.10.3.3 void vrna_message_error (const char *message[ ]*)

```
#include <ViennaRNA/utils.h>
```

Die with an error message.

See also

vrna_message_warning()

Parameters

message	The error message to be printed before exiting with 'FAILURE'

13.10.3.4 void vrna_message_warning (const char *message[ ]*)

```
#include <ViennaRNA/utils.h>
```

Print a warning message.

Print a warning message to *stderr*

Parameters

message	The warning message

13.10.3.5 double vrna_urn (void)

```
#include <ViennaRNA/utils.h>
```

get a random number from [0..1]

See also

vrna_int_urn(), vrna_init_rand()

Note

Usually implemented by calling *erand48()*.

Returns

A random number in range [0..1]

13.10.3.6 int vrna_int_urn (int *from*, int *to*)

```
#include <ViennaRNA/utils.h>
```

Generates a pseudo random integer in a specified range.

See also

vrna_urn(), vrna_init_rand()

Parameters

from	The first number in range
to	The last number in range

Returns

A pseudo random number in range [from, to]

13.10.3.7 char* vrna_time_stamp (void)

```
#include <ViennaRNA/utils.h>
```

Get a timestamp.

Returns a string containing the current date in the format

```
Fri Mar 19 21:10:57 1993
```

Returns

A string containing the timestamp

13.10.3.8 char* get_line (FILE * *fp*)

```
#include <ViennaRNA/utils.h>
```

Read a line of arbitrary length from a stream.

Returns a pointer to the resulting string. The necessary memory is allocated and should be released using *free()* when the string is no longer needed.

Parameters

fp	A file pointer to the stream where the function should read from

Returns

A pointer to the resulting string

13.10.3.9 unsigned int get_input_line (char ** *string,* unsigned int *options*)

```
#include <ViennaRNA/utils.h>
```

Retrieve a line from 'stdin' savely while skipping comment characters and other features This function returns the type of input it has read if recognized. An option argument allows to switch between different reading modes. Currently available options are:
#VRNA_INPUT_NOPRINT_COMMENTS, VRNA_INPUT_NOSKIP_COMMENTS, #VRNA_INPUT_NOELIM_W↩S_SUFFIX

pass a collection of options as one value like this:

```
get_input_line(string, option_1 | option_2 | option_n)
```

If the function recognizes the type of input, it will report it in the return value. It also reports if a user defined 'quit' command (-sign on 'stdin') was given. Possible return values are:
VRNA_INPUT_FASTA_HEADER, VRNA_INPUT_ERROR, VRNA_INPUT_MISC, VRNA_INPUT_QUIT

Parameters

string	A pointer to the character array that contains the line read
options	A collection of options for switching the functions behavior

Returns

A flag with information about what has been read

13.10.3.10 void vrna_message_input_seq_simple (void)

```
#include <ViennaRNA/utils.h>
```

Print a line to *stdout* that asks for an input sequence.

There will also be a ruler (scale line) printed that helps orientation of the sequence positions

13.10.3.11 void vrna_message_input_seq (const char * *s*)

```
#include <ViennaRNA/utils.h>
```

Print a line with a user defined string and a ruler to stdout.

(usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

Parameters

s	A user defined string that will be printed to stdout

13.10.3.12 int* vrna_idx_row_wise (unsigned int *length*)

```
#include <ViennaRNA/utils.h>
```

Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.

Access of a position "(i,j)" is then accomplished by using

```
(i,j) ~ iindx[i]-j
```

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNA Package

Consult the implemented code to find out about the mapping formula ;)

See also

vrna_idx_col_wise()

Parameters

length	The length of the RNA sequence

Returns

The mapper array

13.10.3.13 int* vrna_idx_col_wise (unsigned int *length*)

```
#include <ViennaRNA/utils.h>
```

Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.

Access of a position "(i,j)" is then accomplished by using

```
(i,j) ~ indx[j]+i
```

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNAPackage

Consult the implemented code to find out about the mapping formula ;)

See also

vrna_idx_row_wise()

Parameters

length	The length of the RNA sequence

Returns

The mapper array

13.10.4 Variable Documentation

13.10.4.1 unsigned short xsubi[3]

```
#include <ViennaRNA/utils.h>
```

Current 48 bit random number.

This variable is used by vrna_urn(). These should be set to some random number seeds before the first call to vrna_urn().

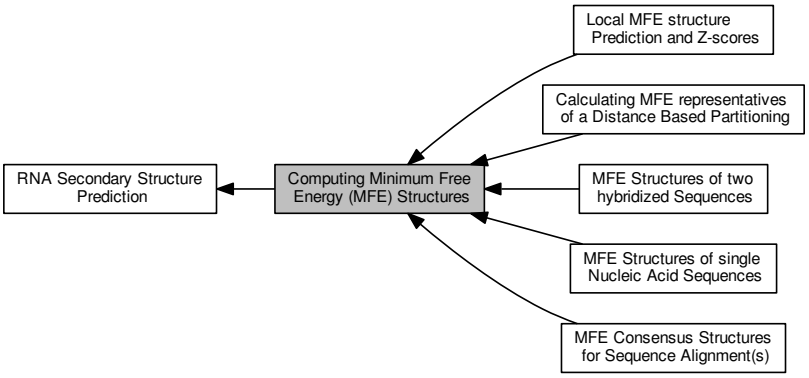
See also

vrna_urn()

13.11 Computing Minimum Free Energy (MFE) Structures

This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

Collaboration diagram for Computing Minimum Free Energy (MFE) Structures:



Modules

- MFE Structures of single Nucleic Acid Sequences

  This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.
- MFE Structures of two hybridized Sequences
- MFE Consensus Structures for Sequence Alignment(s)
- Local MFE structure Prediction and Z-scores
- Calculating MFE representatives of a Distance Based Partitioning

  Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.

Functions

- float vrna_mfe (vrna_fold_compound_t *vc, char *structure)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.

13.11.1 Detailed Description

This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

The library provides a fast dynamic programming minimum free energy folding algorithm as described in [21]. All relevant parts that directly implement the "Zuker & Stiegler" algorithm for single sequences are described in this section.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

Nevertheless, the RNAlib also provides interfaces for the prediction of consensus MFE structures of sequence alignments, MFE structure for two hybridized sequences, local optimal structures and many more. For those more specialized variants of MFE folding routines, please consult the appropriate subsections (Modules) as listed above.

13.11.2 Function Documentation

13.11.2.1 float vrna_mfe (vrna_fold_compound_t * *vc*, char * *structure*)

```
#include <ViennaRNA/mfe.h>
```

Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.

Depending on the type of the provided vrna_fold_compound_t, this function predicts the MFE for a single sequence, or a corresponding averaged MFE for a sequence alignment. If backtracking is activated, it also constructs the corresponding secondary structure, or consensus structure. Therefore, the second parameter, *structure*, has to point to an allocated block of memory with a size of at least $\mathrm{strlen}(\mathrm{sequence}) + 1$ to store the backtracked MFE structure. (For consensus structures, this is the length of the alignment + 1. If NULL is passed, no backtracking will be performed.

Note

This function is polymorphic. It accepts vrna_fold_compound_t of type VRNA_VC_TYPE_SINGLE, and VRNA_VC_TYPE_ALIGNMENT.

See also

vrna_fold_compound_t, vrna_fold_compound(), vrna_fold(), vrna_circfold(), vrna_fold_compound_comparative(), vrna_alifold(), vrna_circalifold()

Parameters

vc	fold compound
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to (Maybe NULL)

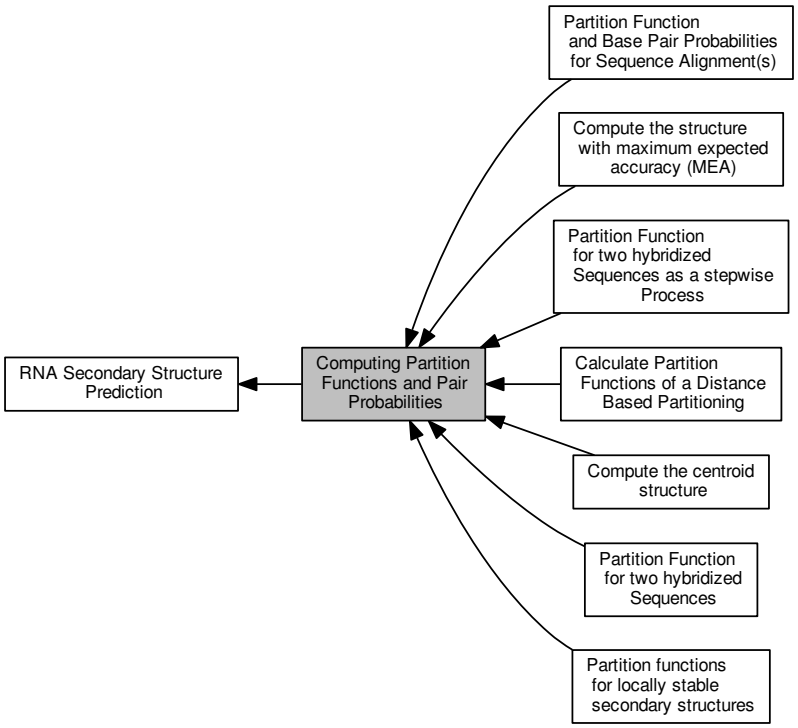
Returns

the minimum free energy (MFE) in kcal/mol

13.12 Computing Partition Functions and Pair Probabilities

This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.

Collaboration diagram for Computing Partition Functions and Pair Probabilities:



Modules

- Compute the structure with maximum expected accuracy (MEA)
- Compute the centroid structure
- Partition Function for two hybridized Sequences

  Partition Function Cofolding.
- Partition Function for two hybridized Sequences as a stepwise Process

  Partition Function Cofolding as a stepwise process.
- Partition Function and Base Pair Probabilities for Sequence Alignment(s)
- Partition functions for locally stable secondary structures
- Calculate Partition Functions of a Distance Based Partitioning

  Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.

Files

- file boltzmann_sampling.h

  Boltzmann Sampling of secondary structures from the ensemble.
- file equilibrium_probs.h

  Equilibrium Probability implementations.
- file part_func.h

  Partition function of single RNA sequences.

Functions

- double vrna_mean_bp_distance_pr (int length, FLT_OR_DBL *pr)

  Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.
- double vrna_mean_bp_distance (vrna_fold_compound_t *vc)

  Get the mean base pair distance in the thermodynamic ensemble.
- vrna_plist_t * vrna_stack_prob (vrna_fold_compound_t *vc, double cutoff)

  Compute stacking probabilities.
- float vrna_pf (vrna_fold_compound_t *vc, char *structure)

  Compute the partition function Q for a given RNA sequence, or sequence alignment.
- float vrna_pf_fold (const char *sequence, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for an RNA sequence using a comparative method.
- float vrna_pf_circfold (const char *sequence, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for a circular RNA sequences using a comparative method.
- float pf_fold_par (const char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_↩ bppm, int is_constrained, int is_circular)

  Compute the partition function Q for a given RNA sequence.
- float pf_fold (const char *sequence, char *structure)

  Compute the partition function Q of an RNA sequence.
- float pf_circ_fold (const char *sequence, char *structure)

  Compute the partition function of a circular RNA sequence.
- void free_pf_arrays (void)

  Free arrays for the partition function recursions.
- void update_pf_params (int length)

  Recalculate energy parameters.
- void update_pf_params_par (int length, vrna_exp_param_t *parameters)

  Recalculate energy parameters.
- FLT_OR_DBL * export_bppm (void)

  Get a pointer to the base pair probability array
  Accessing the base pair probabilities for a pair (i,j) is achieved by.
- int get_pf_arrays (short **S_p, short **S1_p, char **ptype_p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm↩ _p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p)

  Get the pointers to (almost) all relavant computation arrays used in partition function computation.
- double mean_bp_distance (int length)

  Get the mean base pair distance of the last partition function computation.
- double mean_bp_distance_pr (int length, FLT_OR_DBL *pr)

  Get the mean base pair distance in the thermodynamic ensemble.
- vrna_plist_t * vrna_plist_from_probs (vrna_fold_compound_t *vc, double cut_off)

  Create a vrna_plist_t from base pair probability matrix.
- void assign_plist_from_pr (vrna_plist_t **pl, FLT_OR_DBL *probs, int length, double cutoff)

  Create a vrna_plist_t from a probability matrix.

13.12.1 Detailed Description

This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.

Instead of the minimum free energy structure the partition function of all possible structures and from that the pairing probability for every possible pair can be calculated, using a dynamic programming algorithm as described in [13].

13.12.2 Function Documentation

13.12.2.1 double vrna_mean_bp_distance_pr (int *length*, FLT_OR_DBL * *pr*)

```
#include <ViennaRNA/equilibrium_probs.h>
```

Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.

$< d >= \sum_{a,b} p_a p_b d(S_a, S_b)$
this can be computed from the pair probs $p_i j$ as
$< d >= \sum_{ij} p_{ij}(1 - p_{ij})$

Parameters

length	The length of the sequence
pr	The matrix containing the base pair probabilities

Returns

The mean pair distance of the structure ensemble

13.12.2.2 double vrna_mean_bp_distance (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/equilibrium_probs.h>
```

Get the mean base pair distance in the thermodynamic ensemble.

$< d >= \sum_{a,b} p_a p_b d(S_a, S_b)$
this can be computed from the pair probs $p_i j$ as
$< d >= \sum_{ij} p_{ij}(1 - p_{ij})$

Parameters

vc	The fold compound data structure

Returns

The mean pair distance of the structure ensemble

13.12.2.3 vrna_plist_t* vrna_stack_prob (vrna_fold_compound_t * *vc*, double *cutoff*)

```
#include <ViennaRNA/equilibrium_probs.h>
```

Compute stacking probabilities.

For each possible base pair (i, j), compute the probability of a stack (i, j), $(i + 1, j - 1)$.

Parameters

vc	The fold compound data structure with precomputed base pair probabilities
cutoff	A cutoff value that limits the output to stacks with $p > \text{cutoff}$.

Returns

A list of stacks with enclosing base pair (i, j) and probabiltiy p

13.12.2.4 float vrna_pf (vrna_fold_compound_t * *vc*, char * *structure*)

```
#include <ViennaRNA/part_func.h>
```

Compute the partition function Q for a given RNA sequence, or sequence alignment.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters " . , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If the parameter calculate_bppm is set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise after calculations took place pr will contain the probability that bases *i* and *j* pair.

Note

This function is polymorphic. It accepts vrna_fold_compound_t of type VRNA_VC_TYPE_SINGLE, and VRNA_VC_TYPE_ALIGNMENT.

See also

vrna_fold_compound_t, vrna_fold_compound(), vrna_pf_fold(), vrna_pf_circfold(), vrna_fold_compound_comparative(), vrna_pf_alifold(), vrna_pf_circalifold(), vrna_db_from_probs(), vrna_exp_params(), vrna_aln_pinfo()

Parameters

in,out	*vc*	The fold compound data structure
in,out	*structure*	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.12.2.5 float vrna_pf_fold (const char * *sequence*, char * *structure*, vrna_plist_t ** *pl*)

```
#include <ViennaRNA/part_func.h>
```

Compute Partition function Q (and base pair probabilities) for an RNA sequence using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for an RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

See also

vrna_pf_circfold(), vrna_pf(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

sequence	RNA sequence
structure	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
pl	A pointer to a list of vrna_plist_t to store pairing probabilities (Maybe NULL)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.12.2.6 float vrna_pf_circfold (const char * *sequence*, char * *structure*, vrna_plist_t ** *pl*)

```
#include <ViennaRNA/part_func.h>
```

Compute Partition function Q (and base pair probabilities) for a circular RNA sequences using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for a circular RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

See also

vrna_pf_fold(), vrna_pf(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

sequence	A circular RNA sequence
structure	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
pl	A pointer to a list of vrna_plist_t to store pairing probabilities (Maybe NULL)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.12.2.7 float pf_fold_par (const char * *sequence*, char * *structure*, vrna_exp_param_t * *parameters*, int *calculate_bppm*, int *is_constrained*, int *is_circular*)

```
#include <ViennaRNA/part_func.h>
```

Compute the partition function Q for a given RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters " . , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If fold_constrained is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If the parameter calculate_bppm is set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise after calculations took place pr will contain the probability that bases *i* and *j* pair.

Deprecated Use vrna_pf() instead

Note

The global array pr is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function export_bppm()

Postcondition

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable do_backtrack was set the base pair probabilities are already computed and may be accessed for further usage via the export_bppm() function. A call of free_pf_arrays() will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

See also

vrna_pf(), bppm_to_structure(), export_bppm(), vrna_exp_params(), free_pf_arrays()

Parameters

in	*sequence*	The RNA sequence input
in,out	*structure*	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)
in	*parameters*	Data structure containing the precalculated Boltzmann factors
in	*calculate_bppm*	Switch to Base pair probability calculations on/off (0==off)
in	*is_constrained*	Switch to indicate that a structure contraint is passed via the structure argument (0==off)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.12.2.8 float pf_fold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/part_func.h>
```

Compute the partition function Q of an RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters " . , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If fold_constrained is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If do_backtrack has been set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise pr will contain the probability that bases *i* and *j* pair.

Note

The global array pr is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function export_bppm().
OpenMP: This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using pf_fold_par() for a really threadsafe implementation.

Precondition

This function takes its model details from the global variables provided in *RNAlib*

Postcondition

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable do_backtrack was set the base pair probabilities are already computed and may be accessed for further usage via the export_bppm() function. A call of free_pf_arrays() will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

See also

pf_fold_par(), pf_circ_fold(), bppm_to_structure(), export_bppm()

Parameters

sequence	The RNA sequence input
structure	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.12.2.9 float pf_circ_fold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/part_func.h>
```

Compute the partition function of a circular RNA sequence.

Note

The global array pr is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function export_bppm().
OpenMP: This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using pf_fold_par() for a really threadsafe implementation.

Precondition

This function takes its model details from the global variables provided in *RNAlib*

Postcondition

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable do_backtrack was set the base pair probabilities are already computed and may be accessed for further usage via the export_bppm() function. A call of free_pf_arrays() will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

See also

vrna_pf()

Deprecated Use vrna_pf() instead!

Parameters

in	*sequence*	The RNA sequence input
in,out	*structure*	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.12.2.10 void free_pf_arrays (void)

```
#include <ViennaRNA/part_func.h>
```

Free arrays for the partition function recursions.

Call this function if you want to free all allocated memory associated with the partition function forward recursion.

Note

Successive calls of pf_fold(), pf_circ_fold() already check if they should free any memory from a previous run.
OpenMP notice:
This function should be called before leaving a thread in order to avoid leaking memory

Deprecated See vrna_fold_compound_t and its related functions for how to free memory occupied by the dynamic programming matrices

Postcondition

All memory allocated by pf_fold_par(), pf_fold() or pf_circ_fold() will be free'd

See also

pf_fold_par(), pf_fold(), pf_circ_fold()

13.12.2.11 void update_pf_params (int *length*)

```
#include <ViennaRNA/part_func.h>
```

Recalculate energy parameters.

Call this function to recalculate the pair matrix and energy parameters after a change in folding parameters like temperature

Deprecated Use vrna_exp_params_subst() instead

13.12.2.12 void update_pf_params_par (int *length*, vrna_exp_param_t * *parameters*)

```
#include <ViennaRNA/part_func.h>
```

Recalculate energy parameters.

Deprecated Use vrna_exp_params_subst() instead

13.12.2.13 FLT_OR_DBL* export_bppm (void)

```
#include <ViennaRNA/part_func.h>
```

Get a pointer to the base pair probability array

Accessing the base pair probabilities for a pair (i,j) is achieved by.

```
00001 FLT_OR_DBL *pr  = export_bppm();
00002 pr_ij           = pr[iindx[i]-j];
```

Precondition

Call pf_fold_par(), pf_fold() or pf_circ_fold() first to fill the base pair probability array

See also

pf_fold(), pf_circ_fold(), vrna_idx_row_wise()

Returns

A pointer to the base pair probability array

13.12.2.14 int get_pf_arrays (short ** *S_p*, short ** *S1_p*, char ** *ptype_p*, FLT_OR_DBL ** *qb_p*, FLT_OR_DBL ** *qm_p*, FLT_OR_DBL ** *q1k_p*, FLT_OR_DBL ** *qln_p*)

```
#include <ViennaRNA/part_func.h>
```

Get the pointers to (almost) all relavant computation arrays used in partition function computation.

Precondition

In order to assign meaningful pointers, you have to call pf_fold_par() or pf_fold() first!

See also

pf_fold_par(), pf_fold(), pf_circ_fold()

Parameters

out	*S_p*	A pointer to the 'S' array (integer representation of nucleotides)
out	*S1_p*	A pointer to the 'S1' array (2nd integer representation of nucleotides)
out	*ptype_p*	A pointer to the pair type matrix
out	*qb_p*	A pointer to the Q^B matrix
out	*qm_p*	A pointer to the Q^M matrix
out	*q1k_p*	A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1,k)$)
out	*qln_p*	A pointer to the 3' slice of the Q matrix ($qln(l) = Q(l,n)$)

Returns

Non Zero if everything went fine, 0 otherwise

13.12.2.15 double mean_bp_distance (int *length*)

```
#include <ViennaRNA/part_func.h>
```

Get the mean base pair distance of the last partition function computation.

Deprecated Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

See also

vrna_mean_bp_distance(), vrna_mean_bp_distance_pr()

Parameters

length	

Returns

mean base pair distance in thermodynamic ensemble

13.12.2.16 double mean_bp_distance_pr (int *length*, FLT_OR_DBL * *pr*)

```
#include <ViennaRNA/part_func.h>
```

Get the mean base pair distance in the thermodynamic ensemble.

This is a threadsafe implementation of mean_bp_dist() !

$< d >= \sum_{a,b} p_a p_b d(S_a, S_b)$
this can be computed from the pair probs $p_i j$ as
$< d >= \sum_{ij} p_{ij}(1 - p_{ij})$

Deprecated Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

Parameters

length	The length of the sequence
pr	The matrix containing the base pair probabilities

Returns

The mean pair distance of the structure ensemble

13.12.2.17 vrna_plist_t* vrna_plist_from_probs (vrna_fold_compound_t * *vc*, double *cut_off*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a vrna_plist_t from base pair probability matrix.

The probability matrix provided via the vrna_fold_compound_t is parsed and all pair probabilities above the given threshold are used to create an entry in the plist

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

Parameters

in	*vc*	The fold compound
in	*cut_off*	The cutoff value

Returns

A pointer to the plist that is to be created

13.12.2.18 void assign_plist_from_pr (vrna_plist_t ** *pl*, FLT_OR_DBL * *probs*, int *length*, double *cutoff*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a vrna_plist_t from a probability matrix.

The probability matrix given is parsed and all pair probabilities above the given threshold are used to create an entry in the plist

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

Note

This function is threadsafe

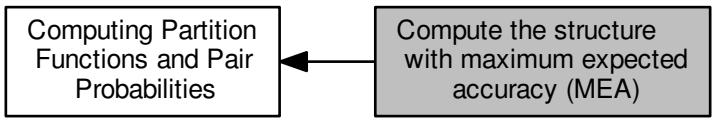
Deprecated Use vrna_plist_from_probs() instead!

Parameters

out	*pl*	A pointer to the vrna_plist_t that is to be created
in	*probs*	The probability matrix used for creating the plist
in	*length*	The length of the RNA sequence
in	*cutoff*	The cutoff value

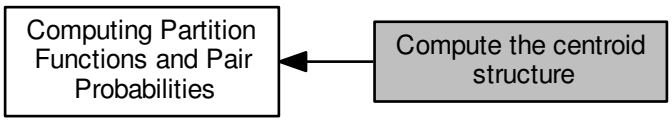
13.13 Compute the structure with maximum expected accuracy (MEA)

Collaboration diagram for Compute the structure with maximum expected accuracy (MEA):



13.14 Compute the centroid structure

Collaboration diagram for Compute the centroid structure:



Functions

- char * vrna_centroid (vrna_fold_compound_t *vc, double *dist)

  Get the centroid structure of the ensemble.
- char * vrna_centroid_from_plist (int length, double *dist, vrna_plist_t *pl)

  Get the centroid structure of the ensemble.
- char * vrna_centroid_from_probs (int length, double *dist, FLT_OR_DBL *probs)

  Get the centroid structure of the ensemble.

13.14.1 Detailed Description

13.14.2 Function Documentation

13.14.2.1 char* vrna_centroid (vrna_fold_compound_t * *vc*, double * *dist*)

```
#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

The centroid is the structure with the minimal average distance to all other structures
$< d(S) >= \sum_{(i,j)\in S}(1-p_{ij}) + \sum_{(i,j)\notin S} p_{ij}$
Thus, the centroid is simply the structure containing all pairs with $p_ij > 0.5$ The distance of the centroid to the ensemble is written to the memory adressed by *dist*.

Parameters

in	*vc*	The fold compound data structure
out	*dist*	A pointer to the distance variable where the centroid distance will be written to

Returns

The centroid structure of the ensemble in dot-bracket notation

13.14.2.2 char* vrna_centroid_from_plist (int *length*, double * *dist*, vrna_plist_t * *pl*)

```
#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for centroid() with a vrna_plist_t input

The centroid is the structure with the minimal average distance to all other structures
$< d(S) >= \sum_{(i,j)\in S}(1-p_{ij}) + \sum_{(i,j)\notin S} p_{ij}$
Thus, the centroid is simply the structure containing all pairs with $p_ij > 0.5$ The distance of the centroid to the ensemble is written to the memory adressed by *dist*.

Parameters

in	*length*	The length of the sequence
out	*dist*	A pointer to the distance variable where the centroid distance will be written to
in	*pl*	A pair list containing base pair probability information about the ensemble

Returns

The centroid structure of the ensemble in dot-bracket notation

13.14.2.3 char* vrna_centroid_from_probs (int *length*, double * *dist*, FLT_OR_DBL * *probs*)

```
#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for centroid() with a probability array input

The centroid is the structure with the minimal average distance to all other structures
$< d(S) >= \sum_{(i,j)\in S}(1-p_{ij}) + \sum_{(i,j)\notin S} p_{ij}$
Thus, the centroid is simply the structure containing all pairs with $p_ij > 0.5$ The distance of the centroid to the ensemble is written to the memory adressed by *dist*.

Parameters

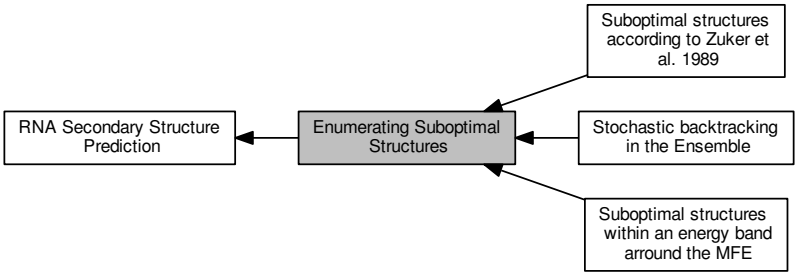
in	*length*	The length of the sequence
out	*dist*	A pointer to the distance variable where the centroid distance will be written to
in	*probs*	An upper triangular matrix containing base pair probabilities (access via iindx vrna_idx_row_wise())

Returns

The centroid structure of the ensemble in dot-bracket notation

13.15 Enumerating Suboptimal Structures

Collaboration diagram for Enumerating Suboptimal Structures:



Modules

- Suboptimal structures according to Zuker et al. 1989
- Suboptimal structures within an energy band arround the MFE
- Stochastic backtracking in the Ensemble

Files

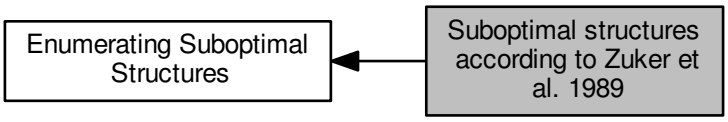
- file subopt.h

  RNAsubopt and density of states declarations.

13.15.1 Detailed Description

13.16 Suboptimal structures according to Zuker et al. 1989

Collaboration diagram for Suboptimal structures according to Zuker et al. 1989:



Functions

- vrna_subopt_solution_t * vrna_subopt_zuker (vrna_fold_compound_t *vc)
  Compute Zuker type suboptimal structures.
- SOLUTION * zukersubopt (const char *string)
  Compute Zuker type suboptimal structures.
- SOLUTION * zukersubopt_par (const char *string, vrna_param_t *parameters)
  Compute Zuker type suboptimal structures.

13.16.1 Detailed Description

13.16.2 Function Documentation

13.16.2.1 vrna_subopt_solution_t* vrna_subopt_zuker (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker [20] , i.e. for every possible base pair the minimum energy structure containing the resp. base pair. Returns a list of these structures and their energies.

Note

This function internally uses the cofold implementation to compute the suboptimal structures. For that purpose, the function doubles the sequence and enlarges the DP matrices, which in fact will grow by a factor of 4 during the computation! At the end of the structure prediction, everything will be re-set to its original requriements, i.e. normal sequence, normal (empty) DP matrices.

Bug Due to resizing, any pre-existing constraints will be lost!

See also

vrna_subopt(), zukersubopt(), zukersubopt_par()

Parameters

vc	fold compound

Returns

List of zuker suboptimal structures

13.16.2.2 SOLUTION* zukersubopt (const char * *string*)

```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker, i.e. for every possible base pair the minimum energy structure containing the resp. base pair. Returns a list of these structures and their energies.

Deprecated use vrna_zukersubopt() instead

Parameters

string	RNA sequence

Returns

List of zuker suboptimal structures

13.16.2.3 SOLUTION* zukersubopt_par (const char * *string*, vrna_param_t * *parameters*)

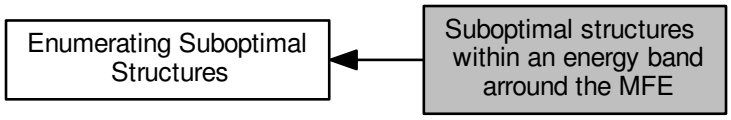
```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

Deprecated use vrna_zukersubopt() instead

13.17 Suboptimal structures within an energy band arround the MFE

Collaboration diagram for Suboptimal structures within an energy band arround the MFE:



Functions

- vrna_subopt_solution_t * vrna_subopt (vrna_fold_compound_t *vc, int delta, int sorted, FILE *fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION * subopt (char *seq, char *structure, int delta, FILE *fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION * subopt_par (char *seq, char *structure, vrna_param_t *parameters, int delta, int is_↩ constrained, int is_circular, FILE *fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION * subopt_circ (char *seq, char *sequence, int delta, FILE *fp)

  Returns list of circular subopt structures or writes to fp.

Variables

- double print_energy

  printing threshold for use with logML
- int subopt_sorted

  Sort output by energy.

13.17.1 Detailed Description

13.17.2 Function Documentation

13.17.2.1 vrna_subopt_solution_t* vrna_subopt (vrna_fold_compound_t * *vc*, int *delta*, int *sorted*, FILE * *fp*)

```
#include <ViennaRNA/subopt.h>
```

Returns list of subopt structures or writes to fp.

This function produces **all** suboptimal secondary structures within 'delta' * 0.01 kcal/mol of the optimum, see [18]. The results are either directly written to a 'fp' (if 'fp' is not NULL), or (fp==NULL) returned in a vrna_subopt_↩ solution_t * list terminated by an entry were the 'structure' member is NULL.

See also

vrna_subopt_zuker()

Parameters

vc	
delta	
sorted	Sort results by energy in ascending order
fp	

Returns

13.17.2.2 SOLUTION* subopt (char * *seq*, char * *structure*, int *delta*, FILE * *fp*)

```
#include <ViennaRNA/subopt.h>
```

Returns list of subopt structures or writes to fp.

This function produces **all** suboptimal secondary structures within 'delta' * 0.01 kcal/mol of the optimum. The results are either directly written to a 'fp' (if 'fp' is not NULL), or (fp==NULL) returned in a SOLUTION * list terminated by an entry were the 'structure' pointer is NULL.

Parameters

seq	
structure	
delta	
fp	

Returns

13.17.2.3 SOLUTION* subopt_circ (char * *seq*, char * *sequence*, int *delta*, FILE * *fp*)

```
#include <ViennaRNA/subopt.h>
```

Returns list of circular subopt structures or writes to fp.

This function is similar to subopt() but calculates secondary structures assuming the RNA sequence to be circular instead of linear

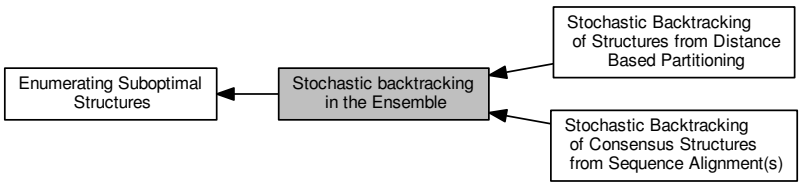
Parameters

seq	
sequence	
delta	
fp	

Returns

13.18 Stochastic backtracking in the Ensemble

Collaboration diagram for Stochastic backtracking in the Ensemble:



Modules

- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)
- Stochastic Backtracking of Structures from Distance Based Partitioning

  Contains functions related to stochastic backtracking from a specified distance class.

Functions

- char * vrna_pbacktrack5 (vrna_fold_compound_t *vc, int length)

  Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.
- char * vrna_pbacktrack (vrna_fold_compound_t *vc)

  Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.
- char * pbacktrack (char *sequence)

  Sample a secondary structure from the Boltzmann ensemble according its probability.
- char * pbacktrack_circ (char *sequence)

  Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

Variables

- int st_back

  Flag indicating that auxilary arrays are needed throughout the computations. This is essential for stochastic backtracking.

13.18.1 Detailed Description

13.18.2 Function Documentation

13.18.2.1 char* vrna_pbacktrack5 (vrna_fold_compound_t * *vc*, int *length*)

```
#include <ViennaRNA/boltzmann_sampling.h>
```

Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.

Precondition

The fold compound has to be obtained using the VRNA_OPTION_HYBRID option in vrna_fold_compound() vrna_pf() has to be called first to fill the partition function matrices

Parameters

vc	The fold compound data structure
length	The length of the subsequence to consider (starting with 5' end)

Returns

A sampled secondary structure in dot-bracket notation

13.18.2.2 char* vrna_pbacktrack (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/boltzmann_sampling.h>
```

Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.

Precondition

The dynamic programming (DP) matrices have to allow for unique multibranch loop decomposition, i.e. the vrna_md_t.uniq_ML flag has to be non-zero before calling vrna_fold_compound()
vrna_pf() has to be called first to fill the partition function matrices

Note

This function is polymorphic. It accepts vrna_fold_compound_t of type VRNA_VC_TYPE_SINGLE, and VRNA_VC_TYPE_ALIGNMENT.
The function will automagically detect cicular RNAs based on the model_details in exp_params as provided via the vrna_fold_compound_t

Parameters

vc	The fold compound data structure

Returns

A sampled secondary structure in dot-bracket notation

13.18.2.3 char* pbacktrack (char * *sequence*)

```
#include <ViennaRNA/part_func.h>
```

Sample a secondary structure from the Boltzmann ensemble according its probability.

Precondition

st_back has to be set to 1 before calling pf_fold() or pf_fold_par()
pf_fold_par() or pf_fold() have to be called first to fill the partition function matrices

Parameters

sequence	The RNA sequence

Returns

A sampled secondary structure in dot-bracket notation

13.18.2.4 char* pbacktrack_circ (char * *sequence*)

```
#include <ViennaRNA/part_func.h>
```

Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

This function does the same as pbacktrack() but assumes the RNA molecule to be circular

Precondition

st_back has to be set to 1 before calling pf_fold() or pf_fold_par()
pf_fold_par() or pf_circ_fold() have to be called first to fill the partition function matrices

Deprecated Use vrna_pbacktrack() instead.

Parameters

sequence	The RNA sequence

Returns

A sampled secondary structure in dot-bracket notation

13.18.3 Variable Documentation

13.18.3.1 int st_back

```
#include <ViennaRNA/part_func.h>
```

Flag indicating that auxilary arrays are needed throughout the computations. This is essential for stochastic backtracking.

Set this variable to 1 prior to a call of pf_fold() to ensure that all matrices needed for stochastic backtracking are filled in the forward recursions

Deprecated set the *uniq_ML* flag in vrna_md_t before passing it to vrna_fold_compound().

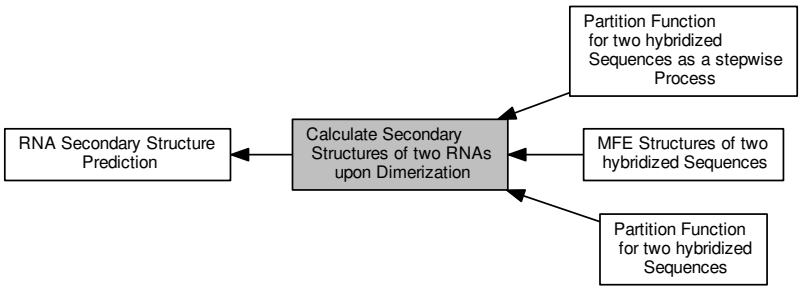
See also

pbacktrack(), pbacktrack_circ

13.19 Calculate Secondary Structures of two RNAs upon Dimerization

Predict structures formed by two molecules upon hybridization.

Collaboration diagram for Calculate Secondary Structures of two RNAs upon Dimerization:



Modules

- MFE Structures of two hybridized Sequences
- Partition Function for two hybridized Sequences

  Partition Function Cofolding.
- Partition Function for two hybridized Sequences as a stepwise Process

  Partition Function Cofolding as a stepwise process.

13.19.1 Detailed Description

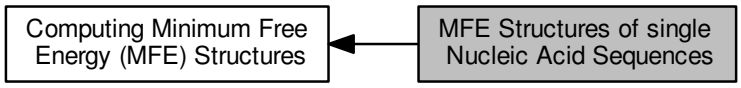
Predict structures formed by two molecules upon hybridization.

The function of an RNA molecule often depends on its interaction with other RNAs. The following routines therefore allow to predict structures formed by two RNA molecules upon hybridization.
One approach to co-folding two RNAs consists of concatenating the two sequences and keeping track of the concatenation point in all energy evaluations. Correspondingly, many of the cofold() and co_pf_fold() routines below take one sequence string as argument and use the the global variable cut_point to mark the concatenation point. Note that while the *RNAcofold* program uses the '&' character to mark the chain break in its input, you should not use an '&' when using the library routines (set cut_point instead).
In a second approach to co-folding two RNAs, cofolding is seen as a stepwise process. In the first step the probability of an unpaired region is calculated and in a second step this probability of an unpaired region is multiplied with the probability of an interaction between the two RNAs. This approach is implemented for the interaction between a long target sequence and a short ligand RNA. Function pf_unstru() calculates the partition function over all unpaired regions in the input sequence. Function pf_interact(), which calculates the partition function over all possible interactions between two sequences, needs both sequence as separate strings as input.

13.20 MFE Structures of single Nucleic Acid Sequences

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

Collaboration diagram for MFE Structures of single Nucleic Acid Sequences:



Functions

- float vrna_fold (const char *sequence, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.
- float vrna_circfold (const char *sequence, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.
- float fold_par (const char *sequence, char *structure, vrna_param_t *parameters, int is_constrained, int is↩ _circular)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.
- float fold (const char *sequence, char *structure)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.
- float circfold (const char *sequence, char *structure)

  Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.
- void free_arrays (void)

  Free arrays for mfe folding.
- void update_fold_params (void)

  Recalculate energy parameters.
- void update_fold_params_par (vrna_param_t *parameters)

  Recalculate energy parameters.
- void export_fold_arrays (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)
- void export_fold_arrays_par (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p)
- void export_circfold_arrays (int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)
- void export_circfold_arrays_par (int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p)
- int LoopEnergy (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1)
- int HairpinE (int size, int type, int si1, int sj1, const char *string)
- void initialize_fold (int length)

13.20.1 Detailed Description

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

The library provides a fast dynamic programming minimum free energy folding algorithm as described by "Zuker & Stiegler (1981)" [21].

13.20.2 Function Documentation

13.20.2.1 float vrna_fold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/fold.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.

This simplified interface to vrna_mfe() computes the MFE and, if required, a secondary structure for an RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

See also

vrna_circfold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

sequence	RNA sequence
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.20.2.2 float vrna_circfold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/fold.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.

This simplified interface to vrna_mfe() computes the MFE and, if required, a secondary structure for a circular RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

See also

vrna_fold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

sequence	RNA sequence
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.20.2.3 float fold_par (const char * *sequence*, char * *structure*, vrna_param_t * *parameters*, int *is_constrained*, int *is_circular*)

```
#include <ViennaRNA/fold.h>
```

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

The first parameter given, the RNA sequence, must be *uppercase* and should only contain an alphabet Σ that is understood by the RNAlib
(e.g. $\Sigma = \{A, U, C, G\}$)
The second parameter, *structure*, must always point to an allocated block of memory with a size of at least $\text{strlen}(\text{sequence}) + 1$

If the third parameter is NULL, global model detail settings are assumed for the folding recursions. Otherwise, the provided parameters are used.

The fourth parameter indicates whether a secondary structure constraint in enhanced dot-bracket notation is passed through the structure parameter or not. If so, the characters " | x < > " are recognized to mark bases that are paired, unpaired, paired upstream, or downstream, respectively. Matching brackets " ( ) " denote base pairs, dots "." are used for unconstrained bases.

To indicate that the RNA sequence is circular and thus has to be post-processed, set the last parameter to non-zero

After a successful call of fold_par(), a backtracked secondary structure (in dot-bracket notation) that exhibits the minimum of free energy will be written to the memory *structure* is pointing to. The function returns the minimum of free energy for any fold of the sequence given.

Note

OpenMP: Passing NULL to the 'parameters' argument involves access to several global model detail variables and thus is not to be considered threadsafe

Deprecated use vrna_mfe() instead!

See also

vrna_mfe(), fold(), circfold(), vrna_md_t, set_energy_model(), get_scaled_parameters()

Parameters

sequence	RNA sequence
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to
parameters	A data structure containing the prescaled energy contributions and the model details. (NULL may be passed, see OpenMP notes above)
is_constrained	Switch to indicate that a structure contraint is passed via the structure argument (0==off)
is_circular	Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

Returns

the minimum free energy (MFE) in kcal/mol

13.20.2.4 float fold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/fold.h>
```

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

This function essentially does the same thing as fold_par(). However, it takes its model details, i.e. temperature, dangles, tetra_loop, noGU, no_closingGU, fold_constrained, noLonelyPairs from the current global settings within the library

Deprecated use vrna_fold(), or vrna_mfe() instead!

See also

fold_par(), circfold()

Parameters

sequence	RNA sequence
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.20.2.5 float circfold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/fold.h>
```

Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.

This function essentially does the same thing as fold_par(). However, it takes its model details, i.e. temperature, dangles, tetra_loop, noGU, no_closingGU, fold_constrained, noLonelyPairs from the current global settings within the library

Deprecated Use vrna_circfold(), or vrna_mfe() instead!

See also

fold_par(), circfold()

Parameters

sequence	RNA sequence
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.20.2.6 void free_arrays (void)

```
#include <ViennaRNA/fold.h>
```

Free arrays for mfe folding.

Deprecated See vrna_fold(), vrna_circfold(), or vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

13.20.2.7 void update_fold_params (void)

```
#include <ViennaRNA/fold.h>
```

Recalculate energy parameters.

Deprecated For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

13.20.2.8 void update_fold_params_par (vrna_param_t * *parameters*)

```
#include <ViennaRNA/fold.h>
```

Recalculate energy parameters.

Deprecated For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

13.20.2.9 void export_fold_arrays (int ** *f5_p*, int ** *c_p*, int ** *fML_p*, int ** *fM1_p*, int ** *indx_p*, char ** *ptype_p*)

```
#include <ViennaRNA/fold.h>
```

Deprecated See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

13.20.2.10 void export_fold_arrays_par (int ** *f5_p*, int ** *c_p*, int ** *fML_p*, int ** *fM1_p*, int ** *indx_p*, char ** *ptype_p*, vrna_param_t ** *P_p*)

```
#include <ViennaRNA/fold.h>
```

Deprecated See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

13.20.2.11 void export_circfold_arrays (int * *Fc_p*, int * *FcH_p*, int * *FcI_p*, int * *FcM_p*, int ** *fM2_p*, int ** *f5_p*, int ** *c_p*, int ** *fML_p*, int ** *fM1_p*, int ** *indx_p*, char ** *ptype_p*)

```
#include <ViennaRNA/fold.h>
```

Deprecated See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

13.20.2.12 void export_circfold_arrays_par (int * *Fc_p*, int * *FcH_p*, int * *FcI_p*, int * *FcM_p*, int ** *fM2_p*, int ** *f5_p*, int ** *c_p*, int ** *fML_p*, int ** *fM1_p*, int ** *indx_p*, char ** *ptype_p*, vrna_param_t ** *P_p*)

```
#include <ViennaRNA/fold.h>
```

Deprecated See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

13.20.2.13 int LoopEnergy (int *n1*, int *n2*, int *type*, int *type_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*)

```
#include <ViennaRNA/fold.h>
```

Deprecated {This function is deprecated and will be removed soon. Use E_IntLoop() instead!}

13.20.2.14 int HairpinE (int *size*, int *type*, int *si1*, int *sj1*, const char * *string*)

```
#include <ViennaRNA/fold.h>
```

Deprecated {This function is deprecated and will be removed soon. Use E_Hairpin() instead!}

13.20.2.15 void initialize_fold (int *length*)

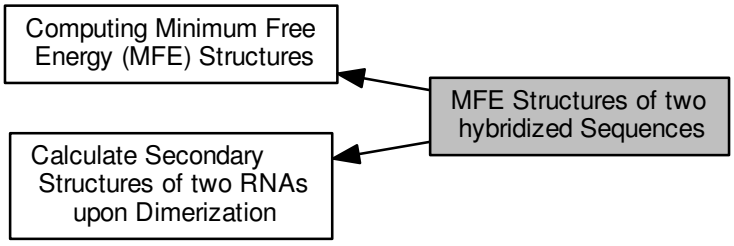
```
#include <ViennaRNA/fold.h>
```

Allocate arrays for folding

Deprecated See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

13.21 MFE Structures of two hybridized Sequences

Collaboration diagram for MFE Structures of two hybridized Sequences:



Files

- file cofold.h

  MFE version of cofolding routines.

Functions

- float vrna_cofold (const char *sequence, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.
- float cofold (const char *sequence, char *structure)

  Compute the minimum free energy of two interacting RNA molecules.
- float cofold_par (const char *string, char *structure, vrna_param_t *parameters, int is_constrained)

  Compute the minimum free energy of two interacting RNA molecules.
- void free_co_arrays (void)

  Free memory occupied by cofold()
- void update_cofold_params (void)

  Recalculate parameters.
- void update_cofold_params_par (vrna_param_t *parameters)

  Recalculate parameters.
- void export_cofold_arrays_gq (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **ggg_p, int **indx_p, char **ptype_p)

  Export the arrays of partition function cofold (with gquadruplex support)
- void export_cofold_arrays (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **indx_p, char **ptype_p)

  Export the arrays of partition function cofold.
- void get_monomere_mfes (float *e1, float *e2)

  get_monomer_free_energies
- void initialize_cofold (int length)
- float vrna_mfe_dimer (vrna_fold_compound_t *vc, char *structure)

  Compute the minimum free energy of two interacting RNA molecules.

13.21.1 Detailed Description

13.21.2 Function Documentation

13.21.2.1 float vrna_cofold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/cofold.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.

This simplified interface to vrna_mfe() computes the MFE and, if required, a secondary structure for two RNA sequences upon dimerization using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

See also

vrna_mfe_dimer(), vrna_fold_compound(), vrna_fold_compound_t, vrna_cut_point_insert()

Parameters

sequence	two RNA sequences separated by the '&' character
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.21.2.2 float cofold (const char * *sequence*, char * *structure*)

```
#include <ViennaRNA/cofold.h>
```

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the fold() function. If cut_point ==-1 results should be the same as with fold().

Deprecated use vrna_mfe_dimer() instead

Parameters

sequence	The two sequences concatenated
structure	Will hold the barcket dot structure of the dimer molecule

Returns

minimum free energy of the structure

13.21.2.3 float cofold_par (const char * *string*, char * *structure*, vrna_param_t * *parameters*, int *is_constrained*)

```
#include <ViennaRNA/cofold.h>
```

Compute the minimum free energy of two interacting RNA molecules.

Deprecated use vrna_mfe_dimer() instead

13.21.2.4 void free_co_arrays (void)

```
#include <ViennaRNA/cofold.h>
```

Free memory occupied by cofold()

Deprecated This function will only free memory allocated by a prior call of cofold() or cofold_par(). See vrna_↩ mfe_dimer() for how to use the new API

Note

folding matrices now reside in the fold compound, and should be free'd there

See also

vrna_fc_destroy(), vrna_mfe_dimer()

13.21.2.5 void update_cofold_params (void)

```
#include <ViennaRNA/cofold.h>
```

Recalculate parameters.

Deprecated See vrna_params_subst() for an alternative using the new API

13.21.2.6 void update_cofold_params_par (vrna_param_t * *parameters*)

```
#include <ViennaRNA/cofold.h>
```

Recalculate parameters.

Deprecated See vrna_params_subst() for an alternative using the new API

13.21.2.7 void export_cofold_arrays_gq (int ** *f5_p*, int ** *c_p*, int ** *fML_p*, int ** *fM1_p*, int ** *fc_p*, int ** *ggg_p*, int ** *indx_p*, char ** *ptype_p*)

```
#include <ViennaRNA/cofold.h>
```

Export the arrays of partition function cofold (with gquadruplex support)

Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

Deprecated folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to cofold() or cofold_par()

See also

vrna_mfe_dimer() for the new API

Parameters

f5_p	A pointer to the 'f5' array, i.e. array conatining best free energy in interval [1,j]
c_p	A pointer to the 'c' array, i.e. array containing best free energy in interval [i,j] given that i pairs with j
fML_p	A pointer to the 'M' array, i.e. array containing best free energy in interval [i,j] for any multiloop segment with at least one stem
fM1_p	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i,j] for multiloop segment with exactly one stem
fc_p	A pointer to the 'fc' array, i.e. array ...
ggg_p	A pointer to the 'ggg' array, i.e. array containing best free energy of a gquadruplex delimited by [i,j]
indx_p	A pointer to the indexing array used for accessing the energy matrices
ptype_p	A pointer to the ptype array containing the base pair types for each possibility (i,j)

13.21.2.8 void export_cofold_arrays (int ** *f5_p*, int ** *c_p*, int ** *fML_p*, int ** *fM1_p*, int ** *fc_p*, int ** *indx_p*, char ** *ptype_p*)

```
#include <ViennaRNA/cofold.h>
```

Export the arrays of partition function cofold.

Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

Deprecated folding matrices now reside within the vrna_fold_compound_t. Thus, this function will only work in conjunction with a prior call to the deprecated functions cofold() or cofold_par()

See also

vrna_mfe_dimer() for the new API

Parameters

f5_p	A pointer to the 'f5' array, i.e. array conatining best free energy in interval [1,j]
c_p	A pointer to the 'c' array, i.e. array containing best free energy in interval [i,j] given that i pairs with j
fML_p	A pointer to the 'M' array, i.e. array containing best free energy in interval [i,j] for any multiloop segment with at least one stem
fM1_p	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i,j] for multiloop segment with exactly one stem
fc_p	A pointer to the 'fc' array, i.e. array ...
indx_p	A pointer to the indexing array used for accessing the energy matrices
ptype↩ _p	A pointer to the ptype array containing the base pair types for each possibility (i,j)

13.21.2.9 void get_monomere_mfes (float * *e1*, float * *e2*)

```
#include <ViennaRNA/cofold.h>
```

get_monomer_free_energies

Export monomer free energies out of cofold arrays

Deprecated {This function is obsolete and will be removed soon!}

Parameters

e1	A pointer to a variable where the energy of molecule A will be written to
e2	A pointer to a variable where the energy of molecule B will be written to

13.21.2.10 void initialize_cofold (int *length*)

```
#include <ViennaRNA/cofold.h>
```

allocate arrays for folding

Deprecated {This function is obsolete and will be removed soon!}

13.21.2.11 float vrna_mfe_dimer (vrna_fold_compound_t * *vc*, char * *structure*)

```
#include <ViennaRNA/mfe.h>
```

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the vrna_mfe() function.

Parameters

vc	fold compound
structure	Will hold the barcket dot structure of the dimer molecule

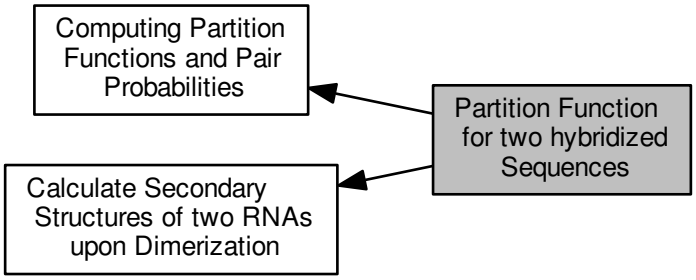
Returns

minimum free energy of the structure

13.22 Partition Function for two hybridized Sequences

Partition Function Cofolding.

Collaboration diagram for Partition Function for two hybridized Sequences:



Files

- file part_func_co.h

  Partition function for two RNA sequences.

Data Structures

- struct vrna_dimer_pf_s

  Data structure returned by vrna_pf_dimer() More...
- struct vrna_dimer_conc_s

  Data structure for concentration dependency computations. More...

Typedefs

- typedef struct vrna_dimer_pf_s vrna_dimer_pf_t

  Typename for the data structure that stores the dimer partition functions, vrna_dimer_pf_s, as returned by vrna_pf_dimer()
- typedef struct vrna_dimer_conc_s vrna_dimer_conc_t

  Typename for the data structure that stores the dimer concentrations, vrna_dimer_conc_s, as required by vrna_pf_dimer_concentration()
- typedef struct vrna_dimer_pf_s cofoldF

  Backward compatibility typedef for vrna_dimer_pf_s.
- typedef struct vrna_dimer_conc_s ConcEnt

  Backward compatibility typedef for vrna_dimer_conc_s.

Functions

- vrna_dimer_pf_t vrna_pf_dimer (vrna_fold_compound_t *vc, char *structure)

  Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.
- void vrna_pf_dimer_probs (double FAB, double FA, double FB, vrna_plist_t *prAB, const vrna_plist_t *prA, const vrna_plist_t *prB, int Alength, const vrna_exp_param_t *exp_params)

  Compute Boltzmann probabilities of dimerization without homodimers.
- vrna_dimer_conc_t * vrna_pf_dimer_concentrations (double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double *startconc, const vrna_exp_param_t *exp_params)

  Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

Variables

- int mirnatog

  Toggles no intrabp in 2nd mol.
- double F_monomer [2]

  Free energies of the two monomers.

13.22.1 Detailed Description

Partition Function Cofolding.

To simplify the implementation the partition function computation is done internally in a null model that does not include the duplex initiation energy, i.e. the entropic penalty for producing a dimer from two monomers). The resulting free energies and pair probabilities are initially relative to that null model. In a second step the free energies can be corrected to include the dimerization penalty, and the pair probabilities can be divided into the conditional pair probabilities given that a re dimer is formed or not formed. See [2] for further details.

13.22.2 Data Structure Documentation

13.22.2.1 struct vrna_dimer_pf_s

Data structure returned by vrna_pf_dimer()

Data Fields

- double F0AB

  Null model without DuplexInit.
- double FAB

  all states with DuplexInit correction
- double FcAB

  true hybrid states only
- double FA

  monomer A
- double FB

  monomer B

13.22.2.2 struct vrna_dimer_conc_s

Data structure for concentration dependency computations.

Data Fields

- double A0
  start concentration A
- double B0
  start concentration B
- double ABc
  End concentration AB.

13.22.3 Function Documentation

13.22.3.1 vrna_dimer_pf_t vrna_pf_dimer (vrna_fold_compound_t * *vc*, char * *structure*)

```
#include <ViennaRNA/part_func_co.h>
```

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This is the cofold partition function folding.

See also

vrna_fold_compound() for how to retrieve the necessary data structure

Parameters

vc	the fold compound data structure
structure	Will hold the structure or constraints

Returns

vrna_dimer_pf_t structure containing a set of energies needed for concentration computations.

13.22.3.2 void vrna_pf_dimer_probs (double *FAB*, double *FA*, double *FB*, vrna_plist_t * *prAB*, const vrna_plist_t * *prA*, const vrna_plist_t * *prB*, int *Alength*, const vrna_exp_param_t * *exp_params*)

```
#include <ViennaRNA/part_func_co.h>
```

Compute Boltzmann probabilities of dimerization without homodimers.

Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by vrna_plist_from_probs(), the dimer probabilities 'prAB' are modified in place.

Parameters

FAB	free energy of dimer AB
FA	free energy of monomer A
FB	free energy of monomer B
prAB	pair probabilities for dimer
prA	pair probabilities monomer
prB	pair probabilities monomer
Alength	Length of molecule A
exp_params	The precomputed Boltzmann factors

13.22.3.3 vrna_dimer_conc_t* vrna_pf_dimer_concentrations (double *FcAB*, double *FcAA*, double *FcBB*, double *FEA*, double *FEB*, const double * *startconc*, const vrna_exp_param_t * *exp_params*)

```
#include <ViennaRNA/part_func_co.h>
```

Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

This function takes an array 'startconc' of input concentrations with alternating entries for the initial concentrations of molecules A and B (terminated by two zeroes), then computes the resulting equilibrium concentrations from the free energies for the dimers. Dimer free energies should be the dimer-only free energies, i.e. the FcAB entries from the vrna_dimer_pf_t struct.

Parameters

FcAB	Free energy of AB dimer (FcAB entry)
FcAA	Free energy of AA dimer (FcAB entry)
FcBB	Free energy of BB dimer (FcAB entry)
FEA	Free energy of monomer A
FEB	Free energy of monomer B
startconc	List of start concentrations [a0],[b0],[a1],[b1],...,[an][bn],[0],[0]
exp_params	The precomputed Boltzmann factors

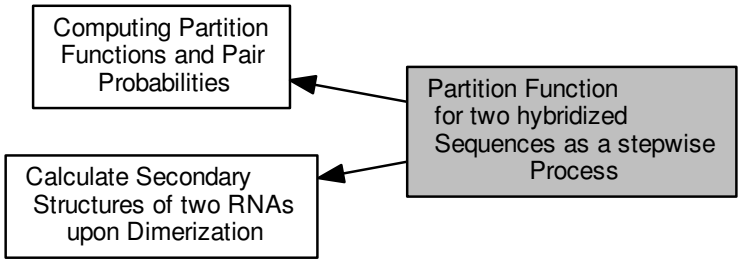
Returns

vrna_dimer_conc_t array containing the equilibrium energies and start concentrations

13.23 Partition Function for two hybridized Sequences as a stepwise Process

Partition Function Cofolding as a stepwise process.

Collaboration diagram for Partition Function for two hybridized Sequences as a stepwise Process:



Files

- file part_func_up.h

  Partition Function Cofolding as stepwise process.

Functions

- pu_contrib * pf_unstru (char *sequence, int max_w)

  Calculate the partition function over all unpaired regions of a maximal length.
- interact * pf_interact (const char *s1, const char *s2, pu_contrib *p_c, pu_contrib *p_c2, int max_w, char *cstruc, int incr3, int incr5)

  Calculates the probability of a local interaction between two sequences.
- void free_interact (interact *pin)

  Frees the output of function pf_interact().
- void free_pu_contrib_struct (pu_contrib *pu)

  Frees the output of function pf_unstru().

13.23.1 Detailed Description

Partition Function Cofolding as a stepwise process.

13.23.2 Function Documentation

13.23.2.1 pu_contrib* pf_unstru (char * *sequence*, int *max_w*)

```
#include <ViennaRNA/part_func_up.h>
```

Calculate the partition function over all unpaired regions of a maximal length.

You have to call function pf_fold() providing the same sequence before calling pf_unstru(). If you want to calculate unpaired regions for a constrained structure, set variable 'structure' in function 'pf_fold()' to the constrain string. It returns a pu_contrib struct containing four arrays of dimension [i = 1 to length(sequence)][j = 0 to u-1] containing all possible contributions to the probabilities of unpaired regions of maximum length u. Each array in pu_contrib contains one of the contributions to the total probability of being unpaired: The probability of being unpaired within an exterior loop is in array pu_contrib->E, the probability of being unpaired within a hairpin loop is in array pu_contrib->H, the probability of being unpaired within an interior loop is in array pu_contrib->I and probability of being unpaired within a multi-loop is in array pu_contrib->M. The total probability of being unpaired is the sum of the four arrays of pu_contrib.

This function frees everything allocated automatically. To free the output structure call free_pu_contrib().

Parameters

sequence	
max_w	

Returns

13.23.2.2 interact* pf_interact (const char * *s1*, const char * *s2*, pu_contrib * *p_c*, pu_contrib * *p_c2*, int *max_w*, char * *cstruc*, int *incr3*, int *incr5*)

```
#include <ViennaRNA/part_func_up.h>
```

Calculates the probability of a local interaction between two sequences.

The function considers the probability that the region of interaction is unpaired within 's1' and 's2'. The longer sequence has to be given as 's1'. The shorter sequence has to be given as 's2'. Function pf_unstru() has to be called for 's1' and 's2', where the probabilities of being unpaired have to be given in 'p_c' and 'p_c2', respectively. If you do not want to include the probabilities of being unpaired for 's2' set 'p_c2' to NULL. If variable 'cstruc' is not NULL, constrained folding is done: The available constrains for intermolecular interaction are: '.' (no constrain), 'x' (the base has no intermolecular interaction) and '|' (the corresponding base has to be paired intermolecularily).
The parameter 'w' determines the maximal length of the interaction. The parameters 'incr5' and 'incr3' allows inclusion of unpaired residues left ('incr5') and right ('incr3') of the region of interaction in 's1'. If the 'incr' options are used, function pf_unstru() has to be called with w=w+incr5+incr3 for the longer sequence 's1'.

It returns a structure of type interact which contains the probability of the best local interaction including residue i in Pi and the minimum free energy in Gi, where i is the position in sequence 's1'. The member Gikjl of structure interact is the best interaction between region [k,i] k<i in longer sequence 's1' and region [j,l] j<l in 's2'. Gikjl_wo is Gikjl without the probability of beeing unpaired.
Use free_interact() to free the returned structure, all other stuff is freed inside pf_interact().

Parameters

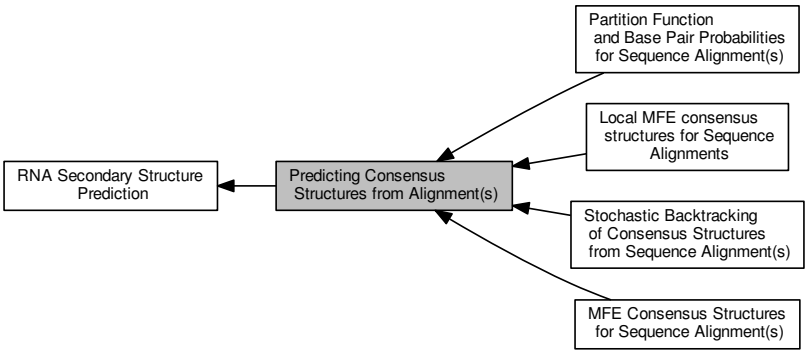
s1	
s2	
p_c	
p_c2	
max_w	
cstruc	
incr3	
incr5	

Returns

13.24 Predicting Consensus Structures from Alignment(s)

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

Collaboration diagram for Predicting Consensus Structures from Alignment(s):



Modules

- MFE Consensus Structures for Sequence Alignment(s)
- Partition Function and Base Pair Probabilities for Sequence Alignment(s)
- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)
- Local MFE consensus structures for Sequence Alignments

Files

- file alifold.h

  compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

Functions

- float energy_of_alistruct (const char **sequences, const char *structure, int n_seq, float *energy)

  Calculate the free energy of a consensus structure given a set of aligned sequences.
- int get_alipf_arrays (short ***S_p, short ***S5_p, short ***S3_p, unsigned short ***a2s_p, char ***Ss↩ _p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm_p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p, short **pscore)

  Get pointers to (almost) all relavant arrays used in alifold's partition function computation.
- void update_alifold_params (void)

  Update the energy parameters for alifold function.
- int vrna_aln_mpi (char *Alseq[ ], int n_seq, int length, int *mini)

  Get the mean pairwise identity in steps from ?to?(ident)
- int get_mpi (char *Alseq[ ], int n_seq, int length, int *mini)

  Get the mean pairwise identity in steps from ?to?(ident)

- void encode_ali_sequence (const char *sequence, short *S, short *s5, short *s3, char *ss, unsigned short *as, int circ)

  Get arrays with encoded sequence of the alignment.
- void alloc_sequence_arrays (const char **sequences, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss, int circ)

  Allocate memory for sequence array used to deal with aligned sequences.
- void free_sequence_arrays (unsigned int n_seq, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss)

  Free the memory of the sequence arrays used to deal with aligned sequences.
- float ** get_ribosum (const char **Alseq, int n_seq, int length)

  Retrieve a RiboSum Scoring Matrix for a given Alignment.

Variables

- double cv_fact

  This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.
- double nc_fact

  This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

13.24.1 Detailed Description

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

Consensus structures can be predicted by a modified version of the fold() algorithm that takes a set of aligned sequences instead of a single sequence. The energy function consists of the mean energy averaged over the sequences, plus a covariance term that favors pairs with consistent and compensatory mutations and penalizes pairs that cannot be formed by all structures. For details see [7] and [1].

13.24.2 Function Documentation

13.24.2.1 float energy_of_alistruct (const char ** *sequences*, const char * *structure*, int *n_seq*, float * *energy*)

```
#include <ViennaRNA/alifold.h>
```

Calculate the free energy of a consensus structure given a set of aligned sequences.

Deprecated Usage of this function is discouraged! Use vrna_eval_structure(), and vrna_eval_covar_structure() instead!

Parameters

sequences	The NULL terminated array of sequences
structure	The consensus structure
n_seq	The number of sequences in the alignment
energy	A pointer to an array of at least two floats that will hold the free energies (energy[0] will contain the free energy, energy[1] will be filled with the covariance energy term)

Returns

free energy in kcal/mol

13.24.2.2 int get_alipf_arrays (short *** *S_p*, short *** *S5_p*, short *** *S3_p*, unsigned short *** *a2s_p*, char *** *Ss_p*, FLT_OR_DBL ** *qb_p*, FLT_OR_DBL ** *qm_p*, FLT_OR_DBL ** *q1k_p*, FLT_OR_DBL ** *qln_p*, short ** *pscore*)

```
#include <ViennaRNA/alifold.h>
```

Get pointers to (almost) all relavant arrays used in alifold's partition function computation.

Note

To obtain meaningful pointers, call alipf_fold first!

See also

pf_alifold(), alipf_circ_fold()

Deprecated It is discouraged to use this function! The new vrna_fold_compound_t allows direct access to all necessary consensus structure prediction related variables!

See also

vrna_fold_compound_t, vrna_fold_compound_comparative(), vrna_pf()

Parameters

S_p	A pointer to the 'S' array (integer representation of nucleotides)
S5_p	A pointer to the 'S5' array
S3_p	A pointer to the 'S3' array
a2s_p	A pointer to the pair type matrix
Ss_p	A pointer to the 'Ss' array
qb_p	A pointer to the Q^B matrix
qm_p	A pointer to the Q^M matrix
q1k_p	A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1,k)$)
qln_p	A pointer to the 3' slice of the Q matrix ($qln(l) = Q(l,n)$)
pscore	A pointer to the start of a pscore list

Returns

Non Zero if everything went fine, 0 otherwise

13.24.2.3 void update_alifold_params (void)

```
#include <ViennaRNA/alifold.h>
```

Update the energy parameters for alifold function.

Call this to recalculate the pair matrix and energy parameters after a change in folding parameters like temperature

Deprecated Usage of this function is discouraged! The new API uses vrna_fold_compound_t to lump all folding related necessities together, including the energy parameters. Use vrna_update_fold_params() to update the energy parameters within a vrna_fold_compound_t.

13.24.2.4 int vrna_aln_mpi (char * *Alseq[ ],* int *n_seq,* int *length,* int * *mini*)

```
#include <ViennaRNA/aln_util.h>
```

Get the mean pairwise identity in steps from ?to?(ident)

Parameters

Alseq	
n_seq	The number of sequences in the alignment
length	The length of the alignment
mini	

Returns

The mean pairwise identity

13.24.2.5 int get_mpi (char * *Alseq[ ],* int *n_seq,* int *length,* int * *mini*)

```
#include <ViennaRNA/aln_util.h>
```

Get the mean pairwise identity in steps from ?to?(ident)

Deprecated Use vrna_aln_mpi() as a replacement

Parameters

Alseq	
n_seq	The number of sequences in the alignment
length	The length of the alignment
mini	

Returns

The mean pairwise identity

13.24.2.6 void encode_ali_sequence (const char * *sequence*, short * *S*, short * *s5*, short * *s3*, char * *ss*, unsigned short * *as*, int *circ*)

```
#include <ViennaRNA/aln_util.h>
```

Get arrays with encoded sequence of the alignment.

this function assumes that in S, S5, s3, ss and as enough space is already allocated (size must be at least sequence length+2)

Parameters

sequence	The gapped sequence from the alignment
S	pointer to an array that holds encoded sequence
s5	pointer to an array that holds the next base 5' of alignment position i
s3	pointer to an array that holds the next base 3' of alignment position i
ss	
as	
circ	assume the molecules to be circular instead of linear (circ=0)

13.24.2.7 void alloc_sequence_arrays (const char ** *sequences*, short *** *S*, short *** *S5*, short *** *S3*, unsigned short *** *a2s*, char *** *Ss*, int *circ*)

```
#include <ViennaRNA/aln_util.h>
```

Allocate memory for sequence array used to deal with aligned sequences.

Note that these arrays will also be initialized according to the sequence alignment given

See also

free_sequence_arrays()

Parameters

sequences	The aligned sequences
S	A pointer to the array of encoded sequences
S5	A pointer to the array that contains the next 5' nucleotide of a sequence position
S3	A pointer to the array that contains the next 3' nucleotide of a sequence position
a2s	A pointer to the array that contains the alignment to sequence position mapping
Ss	A pointer to the array that contains the ungapped sequence
circ	assume the molecules to be circular instead of linear (circ=0)

13.24.2.8 void free_sequence_arrays (unsigned int *n_seq*, short *** *S*, short *** *S5*, short *** *S3*, unsigned short *** *a2s*, char *** *Ss*)

```
#include <ViennaRNA/aln_util.h>
```

Free the memory of the sequence arrays used to deal with aligned sequences.

This function frees the memory previously allocated with alloc_sequence_arrays()

See also

alloc_sequence_arrays()

Parameters

n_seq	The number of aligned sequences
S	A pointer to the array of encoded sequences
S5	A pointer to the array that contains the next 5' nucleotide of a sequence position
S3	A pointer to the array that contains the next 3' nucleotide of a sequence position
a2s	A pointer to the array that contains the alignment to sequence position mapping
Ss	A pointer to the array that contains the ungapped sequence

13.24.3 Variable Documentation

13.24.3.1 double cv_fact

```
#include <ViennaRNA/alifold.h>
```

This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.

Deprecated See vrna_md_t.cv_fact, and vrna_mfe() to avoid using global variables

Default is 1.

13.24.3.2 double nc_fact

```
#include <ViennaRNA/alifold.h>
```

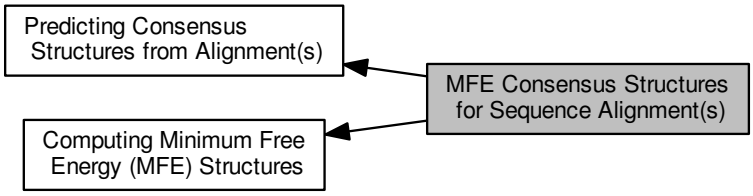
This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

Deprecated See vrna_md_t.nc_fact, and vrna_mfe() to avoid using global variables

Default is 1.

13.25 MFE Consensus Structures for Sequence Alignment(s)

Collaboration diagram for MFE Consensus Structures for Sequence Alignment(s):



Functions

- float vrna_alifold (const char **sequences, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.
- float vrna_circalifold (const char **sequences, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.
- float alifold (const char **strings, char *structure)

  Compute MFE and according consensus structure of an alignment of sequences.
- float circalifold (const char **strings, char *structure)

  Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.
- void free_alifold_arrays (void)

  Free the memory occupied by MFE alifold functions.

13.25.1 Detailed Description

13.25.2 Function Documentation

13.25.2.1 float vrna_alifold (const char ** *sequences*, char * *structure*)

```
#include <ViennaRNA/alifold.h>
```

Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.

This simplified interface to vrna_mfe() computes the MFE and, if required, a consensus secondary structure for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

See also

vrna_circalifold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

sequences	RNA sequence alignment
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.25.2.2 float vrna_circalifold (const char ** *sequences*, char * *structure*)

```
#include <ViennaRNA/alifold.h>
```

Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.

This simplified interface to vrna_mfe() computes the MFE and, if required, a consensus secondary structure for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

See also

vrna_alifold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

sequences	Sequence alignment of circular RNAs
structure	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

13.25.2.3 float alifold (const char ** *strings*, char * *structure*)

```
#include <ViennaRNA/alifold.h>
```

Compute MFE and according consensus structure of an alignment of sequences.

This function predicts the consensus structure for the aligned 'sequences' and returns the minimum free energy; the mfe structure in bracket notation is returned in 'structure'.

Sufficient space must be allocated for 'structure' before calling alifold().

Deprecated Usage of this function is discouraged! Use vrna_alifold(), or vrna_mfe() instead!

See also

vrna_alifold(), vrna_mfe()

Parameters

strings	A pointer to a NULL terminated array of character arrays
structure	A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

Returns

The free energy score in kcal/mol

13.25.2.4 float circalifold (const char ** *strings*, char * *structure*)

```
#include <ViennaRNA/alifold.h>
```

Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.

Deprecated Usage of this function is discouraged! Use vrna_alicircfold(), and vrna_mfe() instead!

See also

vrna_alicircfold(), vrna_alifold(), vrna_mfe()

Parameters

strings	A pointer to a NULL terminated array of character arrays
structure	A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

Returns

The free energy score in kcal/mol

13.25.2.5 void free_alifold_arrays (void)

```
#include <ViennaRNA/alifold.h>
```

Free the memory occupied by MFE alifold functions.

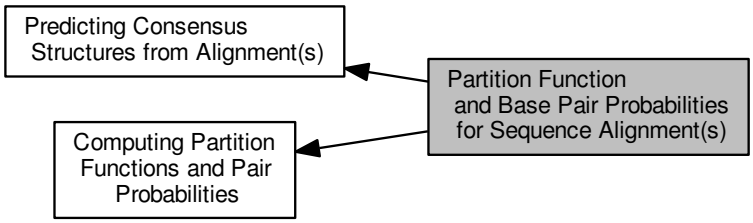
Deprecated Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced vrna_fold_compound_t is handled by vrna_vrna_fold_compound_free()

See also

vrna_vrna_fold_compound_free()

13.26 Partition Function and Base Pair Probabilities for Sequence Alignment(s)

Collaboration diagram for Partition Function and Base Pair Probabilities for Sequence Alignment(s):



Functions

- float vrna_pf_alifold (const char **sequences, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for an RNA sequence alignment using a comparative method.
- float vrna_pf_circalifold (const char **sequences, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.
- float alipf_fold_par (const char **sequences, char *structure, vrna_plist_t **pl, vrna_exp_param_↩ t *parameters, int calculate_bppm, int is_constrained, int is_circular)
- float alipf_fold (const char **sequences, char *structure, vrna_plist_t **pl)

  The partition function version of alifold() works in analogy to pf_fold(). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of vrna_pinfo_t structs. The list is terminated by the first entry with pi.i = 0.
- float alipf_circ_fold (const char **sequences, char *structure, vrna_plist_t **pl)
- FLT_OR_DBL * export_ali_bppm (void)

  Get a pointer to the base pair probability array.
- void free_alipf_arrays (void)

  Free the memory occupied by folding matrices allocated by alipf_fold, alipf_circ_fold, etc.

13.26.1 Detailed Description

13.26.2 Function Documentation

13.26.2.1 float vrna_pf_alifold (const char ** *sequences*, char * *structure*, vrna_plist_t ** *pl*)

```
#include <ViennaRNA/alifold.h>
```

Compute Partition function Q (and base pair probabilities) for an RNA sequence alignment using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

See also

vrna_pf_circalifold(), vrna_pf(), vrna_fold_compound_comparative(), vrna_fold_compound_t

Parameters

sequences	RNA sequence alignment
structure	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
pl	A pointer to a list of vrna_plist_t to store pairing probabilities (Maybe NULL)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.26.2.2 float vrna_pf_circalifold (const char ** *sequences*, char * *structure*, vrna_plist_t ** *pl*)

```
#include <ViennaRNA/alifold.h>
```

Compute Partition function Q (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

See also

vrna_pf_alifold(), vrna_pf(), vrna_fold_compound_comparative(), vrna_fold_compound_t

Parameters

sequences	Sequence alignment of circular RNAs
structure	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
pl	A pointer to a list of vrna_plist_t to store pairing probabilities (Maybe NULL)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

13.26.2.3 float alipf_fold_par (const char ** *sequences*, char * *structure*, vrna_plist_t ** *pl*, vrna_exp_param_t * *parameters*, int *calculate_bppm*, int *is_constrained*, int *is_circular*)

```
#include <ViennaRNA/alifold.h>
```

Deprecated Use vrna_pf() instead

Parameters

sequences	
structure	
pl	
parameters	
calculate_bppm	
is_constrained	
is_circular	

Returns

13.26.2.4 float alipf_fold (const char ** *sequences*, char * *structure*, vrna_plist_t ** *pl*)

```
#include <ViennaRNA/alifold.h>
```

The partition function version of alifold() works in analogy to pf_fold(). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of vrna_pinfo_t structs. The list is terminated by the first entry with pi.i = 0.

Deprecated Use vrna_pf() instead

Parameters

sequences	
structure	
pl	

Returns

13.26.2.5 float alipf_circ_fold (const char ** *sequences*, char * *structure*, vrna_plist_t ** *pl*)

```
#include <ViennaRNA/alifold.h>
```

Deprecated Use vrna_pf() instead

Parameters

sequences	
structure	
pl	

Returns

13.26.2.6 FLT_OR_DBL* export_ali_bppm (void)

```
#include <ViennaRNA/alifold.h>
```

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

Deprecated Usage of this function is discouraged! The new vrna_fold_compound_t allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to vrna_pf(), or any of the old API calls for consensus structure partition function folding.

See also

vrna_fold_compound_t, vrna_fold_compound_comparative(), and vrna_pf()

Returns

A pointer to the base pair probability array

13.26.2.7 void free_alipf_arrays (void)

```
#include <ViennaRNA/alifold.h>
```

Free the memory occupied by folding matrices allocated by alipf_fold, alipf_circ_fold, etc.

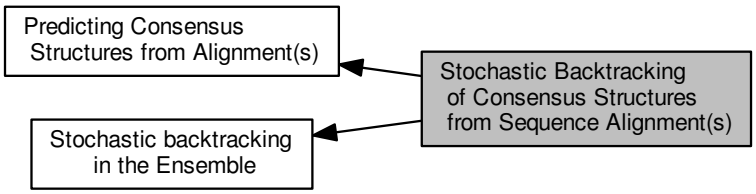
Deprecated Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with vrna_) will be not affected!

See also

vrna_fold_compound_t, vrna_vrna_fold_compound_free()

13.27 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)

Collaboration diagram for Stochastic Backtracking of Consensus Structures from Sequence Alignment(s):



Functions

- char * alipbacktrack (double *prob)

  Sample a consensus secondary structure from the Boltzmann ensemble according its probability.

13.27.1 Detailed Description

13.27.2 Function Documentation

13.27.2.1 char* alipbacktrack (double * *prob*)

```
#include <ViennaRNA/alifold.h>
```

Sample a consensus secondary structure from the Boltzmann ensemble according its probability.

Deprecated Use vrna_pbacktrack() instead!

Parameters

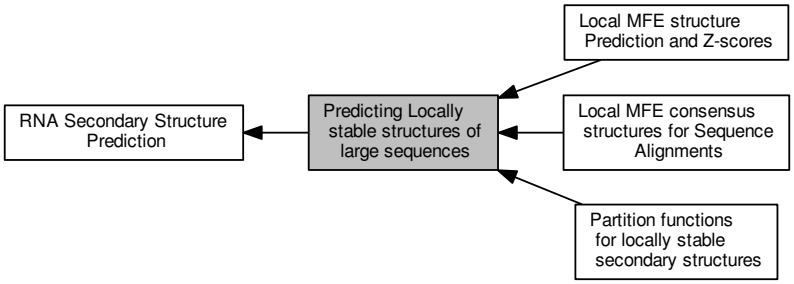
prob	to be described (berni)

Returns

A sampled consensus secondary structure in dot-bracket notation

13.28 Predicting Locally stable structures of large sequences

Collaboration diagram for Predicting Locally stable structures of large sequences:



Modules

- Local MFE structure Prediction and Z-scores
- Partition functions for locally stable secondary structures
- Local MFE consensus structures for Sequence Alignments

Files

- file Lfold.h

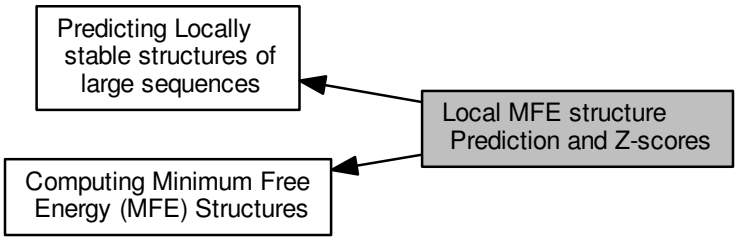
  Predicting local MFE structures of large sequences.

13.28.1 Detailed Description

Local structures can be predicted by a modified version of the fold() algorithm that restricts the span of all base pairs.

13.29 Local MFE structure Prediction and Z-scores

Collaboration diagram for Local MFE structure Prediction and Z-scores:



Functions

- float vrna_Lfold (const char *string, int window_size, FILE *file)

  Local MFE prediction using a sliding window approach (simplified interface)
- float vrna_Lfoldz (const char *string, int window_size, double min_z, FILE *file)

  Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)
- float Lfold (const char *string, char *structure, int maxdist)

  The local analog to fold().
- float Lfoldz (const char *string, char *structure, int maxdist, int zsc, double min_z)
- float vrna_mfe_window (vrna_fold_compound_t *vc, FILE *file)

  Local MFE prediction using a sliding window approach.
- float vrna_mfe_window_zscore (vrna_fold_compound_t *vc, double min_z, FILE *file)

  Local MFE prediction using a sliding window approach (with z-score cut-off)

13.29.1 Detailed Description

13.29.2 Function Documentation

13.29.2.1 float vrna_Lfold (const char * *string*, int *window_size*, FILE * *file*)

```
#include <ViennaRNA/Lfold.h>
```

Local MFE prediction using a sliding window approach (simplified interface)

This simplified interface to vrna_mfe_window() computes the MFE and locally optimal secondary structure using default options. Structures are predicted using a sliding window approach, where base pairs may not span outside the window. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe_window(), and the data structure vrna_fold_compound_t instead.

See also

vrna_mfe_window(), vrna_Lfoldz(), vrna_mfe_window_zscore(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

string	The nucleic acid sequence
window_size	The window size for locally optimal structures
file	The output file handle where predictions are written to (if NULL, output is written to stdout)

13.29.2.2 float vrna_Lfoldz (const char * *string*, int *window_size*, double *min_z*, FILE * *file*)

```
#include <ViennaRNA/Lfold.h>
```

Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)

This simplified interface to vrna_mfe_window_zscore() computes the MFE and locally optimal secondary structure using default options. Structures are predicted using a sliding window approach, where base pairs may not span outside the window. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing. This function is the z-score version of vrna_Lfold(), i.e. only predictions above a certain z-score cut-off value are printed.

Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe_window(), and the data structure vrna_fold_compound_t instead.

See also

vrna_mfe_window_zscore(), vrna_Lfold(), vrna_mfe_window(), vrna_fold_compound(), vrna_fold_compound_t

Parameters

string	The nucleic acid sequence
window_size	The window size for locally optimal structures
min_z	The minimal z-score for a predicted structure to appear in the output
file	The output file handle where predictions are written to (if NULL, output is written to stdout)

13.29.2.3 float Lfold (const char * *string*, char * *structure*, int *maxdist*)

```
#include <ViennaRNA/Lfold.h>
```

The local analog to fold().

Computes the minimum free energy structure including only base pairs with a span smaller than 'maxdist'

Deprecated Use vrna_mfe_window() instead!

13.29.2.4 float Lfoldz (const char * *string*, char * *structure*, int *maxdist*, int *zsc*, double *min_z*)

```
#include <ViennaRNA/Lfold.h>
```

Deprecated Use vrna_mfe_window_zscore() instead!

13.29.2.5 float vrna_mfe_window (vrna_fold_compound_t * *vc*, FILE * *file*)

```
#include <ViennaRNA/mfe.h>
```

Local MFE prediction using a sliding window approach.

Computes minimum free energy structures using a sliding window approach, where base pairs may not span outside the window. In contrast to vrna_mfe(), where a maximum base pair span may be set using the vrna_md_t.max_bp_span attribute and one globally optimal structure is predicted, this function uses a sliding window to retrieve all locally optimal structures within each window. The size of the sliding window is set in the vrna_md_t.window_size attribute, prior to the retrieval of the vrna_fold_compound_t using vrna_fold_compound() with option VRNA_OPTION_WINDOW

The predicted structures are written on-the-fly, either to stdout, if a NULL pointer is passed as file parameter, or to the corresponding filehandle.

See also

vrna_fold_compound(), vrna_mfe_window_zscore(), vrna_mfe(), vrna_Lfold(), vrna_Lfoldz(), VRNA_OPTION_WINDOW, vrna_md_t.max_bp_span, vrna_md_t.window_size

Parameters

vc	The vrna_fold_compound_t with preallocated memory for the DP matrices
file	The output file handle where predictions are written to (maybe NULL)

13.29.2.6 float vrna_mfe_window_zscore (vrna_fold_compound_t * *vc*, double *min_z*, FILE * *file*)

```
#include <ViennaRNA/mfe.h>
```

Local MFE prediction using a sliding window approach (with z-score cut-off)

Computes minimum free energy structures using a sliding window approach, where base pairs may not span outside the window. This function is the z-score version of vrna_mfe_window(), i.e. only predictions above a certain z-score cut-off value are printed. As for vrna_mfe_window(), the size of the sliding window is set in the vrna_md_t.window_size attribute, prior to the retrieval of the vrna_fold_compound_t using vrna_fold_compound() with option VRNA_OPTION_WINDOW.

The predicted structures are written on-the-fly, either to stdout, if a NULL pointer is passed as file parameter, or to the corresponding filehandle.

See also

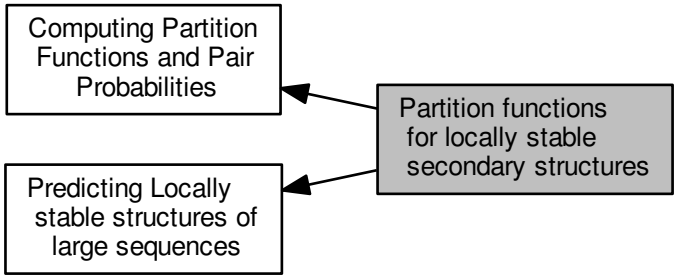
vrna_fold_compound(), vrna_mfe_window_zscore(), vrna_mfe(), vrna_Lfold(), vrna_Lfoldz(), VRNA_OPTION_WINDOW, vrna_md_t.max_bp_span, vrna_md_t.window_size

Parameters

vc	The vrna_fold_compound_t with preallocated memory for the DP matrices
min_z	The minimal z-score for a predicted structure to appear in the output
file	The output file handle where predictions are written to (maybe NULL)

13.30 Partition functions for locally stable secondary structures

Collaboration diagram for Partition functions for locally stable secondary structures:



Files

- file LPfold.h

  Function declarations of partition function variants of the Lfold algorithm.

Functions

- void update_pf_paramsLP (int length)
- plist * pfl_fold (char *sequence, int winSize, int pairSize, float cutoffb, double **pU, plist **dpp2, FILE *pUfp, FILE *spup)

  Compute partition functions for locally stable secondary structures.
- plist * pfl_fold_par (char *sequence, int winSize, int pairSize, float cutoffb, double **pU, plist **dpp2, FILE *pUfp, FILE *spup, vrna_exp_param_t *parameters)

  Compute partition functions for locally stable secondary structures.
- void putoutpU_prob (double **pU, int length, int ulength, FILE *fp, int energies)

  Writes the unpaired probabilities (pU) or opening energies into a file.
- void putoutpU_prob_bin (double **pU, int length, int ulength, FILE *fp, int energies)

  Writes the unpaired probabilities (pU) or opening energies into a binary file.

13.30.1 Detailed Description

13.30.2 Function Documentation

13.30.2.1 void update_pf_paramsLP (int *length*)

```
#include <ViennaRNA/LPfold.h>
```

Parameters

length	

13.30.2.2 plist* pfl_fold (char * *sequence*, int *winSize*, int *pairSize*, float *cutoffb*, double ** *pU*, plist ** *dpp2*, FILE * *pUfp*, FILE * *spup*)

```
#include <ViennaRNA/LPfold.h>
```

Compute partition functions for locally stable secondary structures.

pfl_fold computes partition functions for every window of size 'winSize' possible in a RNA molecule, allowing only pairs with a span smaller than 'pairSize'. It returns the mean pair probabilities averaged over all windows containing the pair in 'pl'. 'winSize' should always be >= 'pairSize'. Note that in contrast to Lfold(), bases outside of the window do not influence the structure at all. Only probabilities higher than 'cutoffb' are kept.

If 'pU' is supplied (i.e is not the NULL pointer), pfl_fold() will also compute the mean probability that regions of length 'u' and smaller are unpaired. The parameter 'u' is supplied in 'pup[0][0]'. On return the 'pup' array will contain these probabilities, with the entry on 'pup[x][y]' containing the mean probability that x and the y-1 preceding bases are unpaired. The 'pU' array needs to be large enough to hold n+1 float* entries, where n is the sequence length.

If an array dpp2 is supplied, the probability of base pair (i,j) given that there already exists a base pair (i+1,j-1) is also computed and saved in this array. If pUfp is given (i.e. not NULL), pU is not saved but put out imediately. If spup is given (i.e. is not NULL), the pair probabilities in pl are not saved but put out imediately.

Parameters

sequence	RNA sequence
winSize	size of the window
pairSize	maximum size of base pair
cutoffb	cutoffb for base pairs
pU	array holding all unpaired probabilities
dpp2	array of dependent pair probabilities
pUfp	file pointer for pU
spup	file pointer for pair probabilities

Returns

list of pair probabilities

13.30.2.3 void putoutpU_prob (double ** *pU*, int *length*, int *ulength*, FILE * *fp*, int *energies*)

```
#include <ViennaRNA/LPfold.h>
```

Writes the unpaired probabilities (pU) or opening energies into a file.

Can write either the unpaired probabilities (accessibilities) pU or the opening energies -log(pU)kT into a file

Parameters

pU	pair probabilities
length	length of RNA sequence
ulength	maximum length of unpaired stretch
fp	file pointer of destination file
energies	switch to put out as opening energies

13.30.2.4 void putoutpU_prob_bin (double ** *pU*, int *length*, int *ulength*, FILE * *fp*, int *energies*)

```
#include <ViennaRNA/LPfold.h>
```

Writes the unpaired probabilities (pU) or opening energies into a binary file.

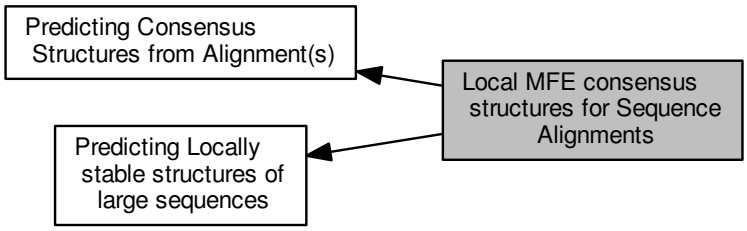
Can write either the unpaired probabilities (accessibilities) pU or the opening energies -log(pU)kT into a file

Parameters

pU	pair probabilities
length	length of RNA sequence
ulength	maximum length of unpaired stretch
fp	file pointer of destination file
energies	switch to put out as opening energies

13.31 Local MFE consensus structures for Sequence Alignments

Collaboration diagram for Local MFE consensus structures for Sequence Alignments:



Functions

- float aliLfold (const char **strings, char *structure, int maxdist)

13.31.1 Detailed Description

13.31.2 Function Documentation

13.31.2.1 float aliLfold (const char ** *strings*, char * *structure*, int *maxdist*)

```
#include <ViennaRNA/Lfold.h>
```

Parameters

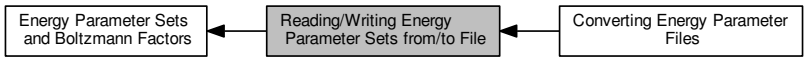
strings	
structure	
maxdist	

Returns

13.32 Reading/Writing Energy Parameter Sets from/to File

Read and Write energy parameter sets from and to text files.

Collaboration diagram for Reading/Writing Energy Parameter Sets from/to File:



Modules

- Converting Energy Parameter Files

  Convert energy parameter files into the latest format.

Files

- file read_epars.h

Functions

- void read_parameter_file (const char fname[ ])

  Read energy parameters from a file.
- void write_parameter_file (const char fname[ ])

  Write energy parameters to a file.

13.32.1 Detailed Description

Read and Write energy parameter sets from and to text files.

A default set of parameters, identical to the one described in [12] and [16], is compiled into the library.

13.32.2 Function Documentation

13.32.2.1 void read_parameter_file (const char *fname[ ]*)

```
#include <ViennaRNA/read_epars.h>
```

Read energy parameters from a file.

Parameters

fname	The path to the file containing the energy parameters

13.32.2.2 void write_parameter_file (const char *fname[ ]*)

```
#include <ViennaRNA/read_epars.h>
```

Write energy parameters to a file.

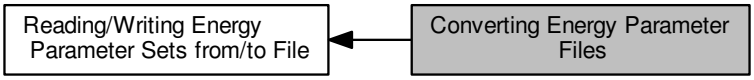
Parameters

fname	A filename (path) for the file where the current energy parameters will be written to

13.33 Converting Energy Parameter Files

Convert energy parameter files into the latest format.

Collaboration diagram for Converting Energy Parameter Files:



Files

- file convert_epars.h

  Functions and definitions for energy parameter file format conversion.

Macros

- #define VRNA_CONVERT_OUTPUT_ALL 1U
- #define VRNA_CONVERT_OUTPUT_HP 2U
- #define VRNA_CONVERT_OUTPUT_STACK 4U
- #define VRNA_CONVERT_OUTPUT_MM_HP 8U
- #define VRNA_CONVERT_OUTPUT_MM_INT 16U
- #define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U
- #define VRNA_CONVERT_OUTPUT_MM_INT_23 64U
- #define VRNA_CONVERT_OUTPUT_MM_MULTI 128U
- #define VRNA_CONVERT_OUTPUT_MM_EXT 256U
- #define VRNA_CONVERT_OUTPUT_DANGLE5 512U
- #define VRNA_CONVERT_OUTPUT_DANGLE3 1024U
- #define VRNA_CONVERT_OUTPUT_INT_11 2048U
- #define VRNA_CONVERT_OUTPUT_INT_21 4096U
- #define VRNA_CONVERT_OUTPUT_INT_22 8192U
- #define VRNA_CONVERT_OUTPUT_BULGE 16384U
- #define VRNA_CONVERT_OUTPUT_INT 32768U
- #define VRNA_CONVERT_OUTPUT_ML 65536U
- #define VRNA_CONVERT_OUTPUT_MISC 131072U
- #define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U
- #define VRNA_CONVERT_OUTPUT_VANILLA 524288U
- #define VRNA_CONVERT_OUTPUT_NINIO 1048576U
- #define VRNA_CONVERT_OUTPUT_DUMP 2097152U

Functions

- void convert_parameter_file (const char *iname, const char *oname, unsigned int options)

13.33.1 Detailed Description

Convert energy parameter files into the latest format.

To preserve some backward compatibility the RNAlib also provides functions to convert energy parameter files from the format used in version 1.4-1.8 into the new format used since version 2.0

13.33.2 Macro Definition Documentation

13.33.2.1 #define VRNA_CONVERT_OUTPUT_ALL 1U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of a complete parameter set

13.33.2.2 #define VRNA_CONVERT_OUTPUT_HP 2U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of hairpin contributions

13.33.2.3 #define VRNA_CONVERT_OUTPUT_STACK 4U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of base pair stack contributions

13.33.2.4 #define VRNA_CONVERT_OUTPUT_MM_HP 8U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of hairpin mismatch contribution

13.33.2.5 #define VRNA_CONVERT_OUTPUT_MM_INT 16U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of interior loop mismatch contribution

13.33.2.6 #define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 1:n interior loop mismatch contribution

13.33.2.7 #define VRNA_CONVERT_OUTPUT_MM_INT_23 64U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 2:3 interior loop mismatch contribution

13.33.2.8 #define VRNA_CONVERT_OUTPUT_MM_MULTI 128U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of multi loop mismatch contribution

13.33.2.9 #define VRNA_CONVERT_OUTPUT_MM_EXT 256U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of exterior loop mismatch contribution

13.33.2.10 #define VRNA_CONVERT_OUTPUT_DANGLE5 512U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 5' dangle conctribution

13.33.2.11 #define VRNA_CONVERT_OUTPUT_DANGLE3 1024U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 3' dangle contribution

13.33.2.12 #define VRNA_CONVERT_OUTPUT_INT_11 2048U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 1:1 interior loop contribution

13.33.2.13 #define VRNA_CONVERT_OUTPUT_INT_21 4096U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 2:1 interior loop contribution

13.33.2.14 #define VRNA_CONVERT_OUTPUT_INT_22 8192U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of 2:2 interior loop contribution

13.33.2.15 #define VRNA_CONVERT_OUTPUT_BULGE 16384U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of bulge loop contribution

13.33.2.16 #define VRNA_CONVERT_OUTPUT_INT 32768U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of interior loop contribution

13.33.2.17 #define VRNA_CONVERT_OUTPUT_ML 65536U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of multi loop contribution

13.33.2.18 #define VRNA_CONVERT_OUTPUT_MISC 131072U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of misc contributions (such as terminalAU)

13.33.2.19 #define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of special hairpin contributions (tri-, tetra-, hexa-loops)

13.33.2.20 #define VRNA_CONVERT_OUTPUT_VANILLA 524288U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of given parameters only

Note

This option overrides all other output options, except VRNA_CONVERT_OUTPUT_DUMP !

13.33.2.21 #define VRNA_CONVERT_OUTPUT_NINIO 1048576U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate printing of interior loop asymmetry contribution

13.33.2.22 #define VRNA_CONVERT_OUTPUT_DUMP 2097152U

```
#include <ViennaRNA/convert_epars.h>
```

Flag to indicate dumping the energy contributions from the library instead of an input file

13.33.3 Function Documentation

13.33.3.1 void convert_parameter_file (const char * *iname*, const char * *oname*, unsigned int *options*)

```
#include <ViennaRNA/convert_epars.h>
```

Convert/dump a Vienna 1.8.4 formatted energy parameter file

The options argument allows to control the different output modes.
Currently available options are:
VRNA_CONVERT_OUTPUT_ALL, VRNA_CONVERT_OUTPUT_HP, VRNA_CONVERT_OUTPUT_STACK
VRNA_CONVERT_OUTPUT_MM_HP, VRNA_CONVERT_OUTPUT_MM_INT, VRNA_CONVERT_OUTPUT_↩
MM_INT_1N
VRNA_CONVERT_OUTPUT_MM_INT_23, VRNA_CONVERT_OUTPUT_MM_MULTI, VRNA_CONVERT_OUT↩
PUT_MM_EXT
VRNA_CONVERT_OUTPUT_DANGLE5, VRNA_CONVERT_OUTPUT_DANGLE3, VRNA_CONVERT_OUTPU↩
T_INT_11
VRNA_CONVERT_OUTPUT_INT_21, VRNA_CONVERT_OUTPUT_INT_22, VRNA_CONVERT_OUTPUT_BU↩
LGE
VRNA_CONVERT_OUTPUT_INT, VRNA_CONVERT_OUTPUT_ML, VRNA_CONVERT_OUTPUT_MISC
VRNA_CONVERT_OUTPUT_SPECIAL_HP, VRNA_CONVERT_OUTPUT_VANILLA, VRNA_CONVERT_OUT↩
PUT_NINIO
VRNA_CONVERT_OUTPUT_DUMP

The defined options are fine for bitwise compare- and assignment-operations, e. g.: pass a collection of options as a single value like this:

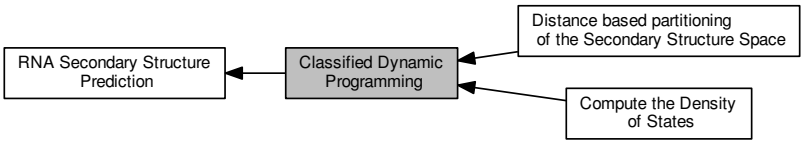
```
convert_parameter_file(ifile, ofile, option_1 | option_2 | option_n)
```

Parameters

iname	The input file name (If NULL input is read from stdin)
oname	The output file name (If NULL output is written to stdout)
options	The options (as described above)

13.34 Classified Dynamic Programming

Collaboration diagram for Classified Dynamic Programming:



Modules

- Distance based partitioning of the Secondary Structure Space

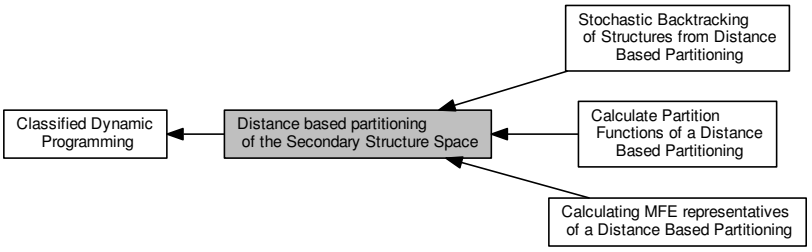
  Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.
- Compute the Density of States

13.34.1 Detailed Description

13.35 Distance based partitioning of the Secondary Structure Space

Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

Collaboration diagram for Distance based partitioning of the Secondary Structure Space:



Modules

- Calculating MFE representatives of a Distance Based Partitioning

  Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.
- Calculate Partition Functions of a Distance Based Partitioning

  Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.
- Stochastic Backtracking of Structures from Distance Based Partitioning

  Contains functions related to stochastic backtracking from a specified distance class.

13.35.1 Detailed Description

Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

All functions related to this group implement the basic recursions for MFE folding, partition function computation and stochastic backtracking with a *classified dynamic programming* approach. The secondary structure space is divided into partitions according to the base pair distance to two given reference structures and all relevant properties are calculated for each of the resulting partitions

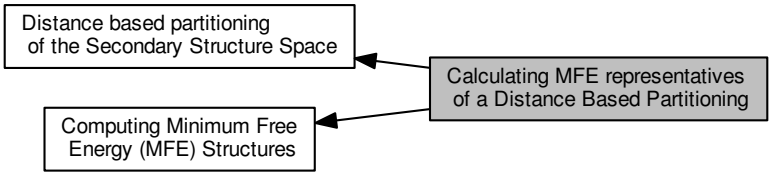
See also

For further details, we refer to Lorenz et al. 2009 [11]

13.36 Calculating MFE representatives of a Distance Based Partitioning

Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.

Collaboration diagram for Calculating MFE representatives of a Distance Based Partitioning:



Files

- file 2Dfold.h

Data Structures

- struct vrna_sol_TwoD_t

  Solution element returned from vrna_mfe_TwoD() More...
- struct TwoDfold_vars

  Variables compound for 2Dfold MFE folding. More...

Typedefs

- typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t

  Solution element returned from vrna_mfe_TwoD()
- typedef struct TwoDfold_vars TwoDfold_vars

  Variables compound for 2Dfold MFE folding.

Functions

- vrna_sol_TwoD_t * vrna_mfe_TwoD (vrna_fold_compound_t *vc, int distance1, int distance2)

  Compute MFE's and representative for distance partitioning.
- char * vrna_backtrack5_TwoD (vrna_fold_compound_t *vc, int k, int l, unsigned int j)

  Backtrack a minimum free energy structure from a 5' section of specified length.
- TwoDfold_vars * get_TwoDfold_variables (const char *seq, const char *structure1, const char *structure2, int circ)

  Get a structure of type TwoDfold_vars prefilled with current global settings.
- void destroy_TwoDfold_variables (TwoDfold_vars *our_variables)

  Destroy a TwoDfold_vars datastructure without memory loss.
- vrna_sol_TwoD_t * TwoDfoldList (TwoDfold_vars *vars, int distance1, int distance2)

  Compute MFE's and representative for distance partitioning.
- char * TwoDfold_backtrack_f5 (unsigned int j, int k, int l, TwoDfold_vars *vars)

  Backtrack a minimum free energy structure from a 5' section of specified length.

13.36.1 Detailed Description

Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.

See also

For further details, we refer to Lorenz et al. 2009 [11]

13.36.2 Data Structure Documentation

13.36.2.1 struct vrna_sol_TwoD_t

Solution element returned from vrna_mfe_TwoD()

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'en' of type float representing the free energy in kcal/mol and an attribute 's' of type char* containg the secondary structure representative,

A value of INF in k denotes the end of a list

See also

vrna_mfe_TwoD()

Data Fields

- int k

  Distance to first reference.
- int l

  Distance to second reference.
- float en

  Free energy in kcal/mol.
- char * s

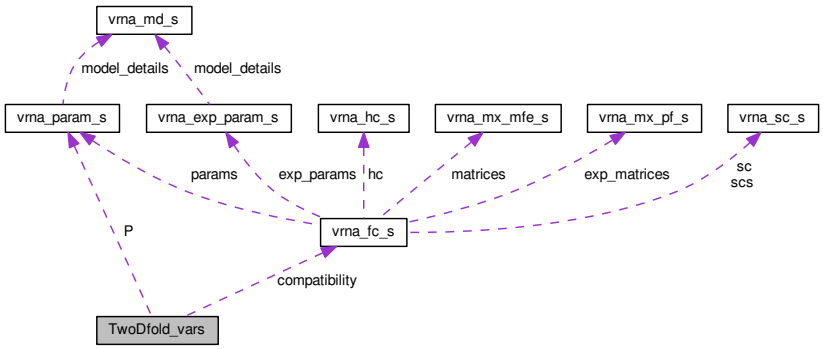
  MFE representative structure in dot-bracket notation.

13.36.2.2 struct TwoDfold_vars

Variables compound for 2Dfold MFE folding.

Deprecated This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Collaboration diagram for TwoDfold_vars:



Data Fields

- vrna_param_t * P

  Precomputed energy parameters and model details.
- int do_backtrack

  Flag whether to do backtracing of the structure(s) or not.
- char * ptype

  Precomputed array of pair types.
- char * sequence

  The input sequence.
- short * S1

  The input sequences in numeric form.
- unsigned int maxD1

  Maximum allowed base pair distance to first reference.
- unsigned int maxD2

  Maximum allowed base pair distance to second reference.
- unsigned int * mm1

  Maximum matching matrix, reference struct 1 disallowed.
- unsigned int * mm2

  Maximum matching matrix, reference struct 2 disallowed.
- int * my_iindx

  Index for moving in quadratic distancy dimensions.
- unsigned int * referenceBPs1

  Matrix containing number of basepairs of reference structure1 in interval [i,j].
- unsigned int * referenceBPs2

  Matrix containing number of basepairs of reference structure2 in interval [i,j].
- unsigned int * bpdist

  Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].

13.36.3 Typedef Documentation

13.36.3.1 typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t

```
#include <ViennaRNA/2Dfold.h>
```

Solution element returned from vrna_mfe_TwoD()

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'en' of type float representing the free energy in kcal/mol and an attribute 's' of type char* containg the secondary structure representative,

A value of INF in k denotes the end of a list

See also

vrna_mfe_TwoD()

13.36.3.2 typedef struct TwoDfold_vars TwoDfold_vars

```
#include <ViennaRNA/2Dfold.h>
```

Variables compound for 2Dfold MFE folding.

Deprecated This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

13.36.4 Function Documentation

13.36.4.1 vrna_sol_TwoD_t* vrna_mfe_TwoD (vrna_fold_compound_t * *vc,* int *distance1,* int *distance2*)

```
#include <ViennaRNA/2Dfold.h>
```

Compute MFE's and representative for distance partitioning.

This function computes the minimum free energies and a representative secondary structure for each distance class according to the two references specified in the datastructure 'vars'. The maximum basepair distance to each of both references may be set by the arguments 'distance1' and 'distance2', respectively. If both distance arguments are set to '-1', no restriction is assumed and the calculation is performed for each distance class possible.

The returned list contains an entry for each distance class. If a maximum basepair distance to either of the references was passed, an entry with k=l=-1 will be appended in the list, denoting the class where all structures exceeding the maximum will be thrown into. The end of the list is denoted by an attribute value of INF in the k-attribute of the list entry.

See also

vrna_fold_compound_TwoD(), vrna_fold_compound_free(), vrna_pf_TwoD() vrna_backtrack5_TwoD(), vrna_sol_TwoD_t, vrna_fold_compound_t

Parameters

vc	The datastructure containing all precomputed folding attributes
distance1	maximum distance to reference1 (-1 means no restriction)
distance2	maximum distance to reference2 (-1 means no restriction)

Returns

A list of minimum free energies (and corresponding structures) for each distance class

13.36.4.2 char* vrna_backtrack5_TwoD (vrna_fold_compound_t * *vc*, int *k*, int *l*, unsigned int *j*)

```
#include <ViennaRNA/2Dfold.h>
```

Backtrack a minimum free energy structure from a 5' section of specified length.

This function allows to backtrack a secondary structure beginning at the 5' end, a specified length and residing in a specific distance class. If the argument 'k' gets a value of -1, the structure that is backtracked is assumed to reside in the distance class where all structures exceeding the maximum basepair distance specified in vrna_mfe_TwoD() belong to.

Note

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to vrna_mfe_TwoD() preceding this function is mandatory!

See also

vrna_mfe_TwoD()

Parameters

vc	The datastructure containing all precomputed folding attributes
j	The length in nucleotides beginning from the 5' end
k	distance to reference1 (may be -1)
l	distance to reference2

13.36.4.3 TwoDfold_vars* get_TwoDfold_variables (const char * *seq*, const char * *structure1*, const char * *structure2*, int *circ*)

```
#include <ViennaRNA/2Dfold.h>
```

Get a structure of type TwoDfold_vars prefilled with current global settings.

This function returns a datastructure of type TwoDfold_vars. The data fields inside the TwoDfold_vars are prefilled by global settings and all memory allocations necessary to start a computation are already done for the convenience of the user

Note

Make sure that the reference structures are compatible with the sequence according to Watson-Crick- and Wobble-base pairing

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Parameters

seq	The RNA sequence
structure1	The first reference structure in dot-bracket notation
structure2	The second reference structure in dot-bracket notation
circ	A switch to indicate the assumption to fold a circular instead of linear RNA (0=OFF, 1=ON)

Returns

A datastructure prefilled with folding options and allocated memory

13.36.4.4 void destroy_TwoDfold_variables (TwoDfold_vars * *our_variables*)

```
#include <ViennaRNA/2Dfold.h>
```

Destroy a TwoDfold_vars datastructure without memory loss.

This function free's all allocated memory that depends on the datastructure given.

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Parameters

our_variables	A pointer to the datastructure to be destroyed

13.36.4.5 vrna_sol_TwoD_t* TwoDfoldList (TwoDfold_vars * *vars*, int *distance1*, int *distance2*)

```
#include <ViennaRNA/2Dfold.h>
```

Compute MFE's and representative for distance partitioning.

This function computes the minimum free energies and a representative secondary structure for each distance class according to the two references specified in the datastructure 'vars'. The maximum basepair distance to each of both references may be set by the arguments 'distance1' and 'distance2', respectively. If both distance arguments are set to '-1', no restriction is assumed and the calculation is performed for each distance class possible.

The returned list contains an entry for each distance class. If a maximum basepair distance to either of the references was passed, an entry with k=l=-1 will be appended in the list, denoting the class where all structures exceeding the maximum will be thrown into. The end of the list is denoted by an attribute value of INF in the k-attribute of the list entry.

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Parameters

vars	the datastructure containing all predefined folding attributes
distance1	maximum distance to reference1 (-1 means no restriction)
distance2	maximum distance to reference2 (-1 means no restriction)

13.36.4.6 char* TwoDfold_backtrack_f5 (unsigned int *j*, int *k*, int *l*, TwoDfold_vars * *vars*)

```
#include <ViennaRNA/2Dfold.h>
```

Backtrack a minimum free energy structure from a 5' section of specified length.

This function allows to backtrack a secondary structure beginning at the 5' end, a specified length and residing in a specific distance class. If the argument 'k' gets a value of -1, the structure that is backtracked is assumed to reside in the distance class where all structures exceeding the maximum basepair distance specified in TwoDfold() belong to.

Note

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to TwoDfold() preceding this function is mandatory!

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_mfe_TwoD(), vrna_backtrack5_TwoD(), and vrna_fold_compound_free() instead!

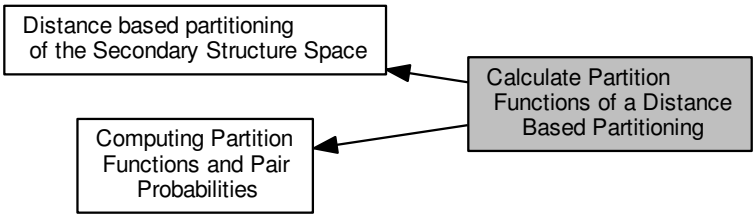
Parameters

j	The length in nucleotides beginning from the 5' end
k	distance to reference1 (may be -1)
l	distance to reference2
vars	the datastructure containing all predefined folding attributes

13.37 Calculate Partition Functions of a Distance Based Partitioning

Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.

Collaboration diagram for Calculate Partition Functions of a Distance Based Partitioning:



Files

- file 2Dpfold.h

Data Structures

- struct vrna_sol_TwoD_pf_t

  Solution element returned from vrna_pf_TwoD() More...

Typedefs

- typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t

  Solution element returned from vrna_pf_TwoD()

Functions

- vrna_sol_TwoD_pf_t * vrna_pf_TwoD (vrna_fold_compound_t *vc, int maxDistance1, int maxDistance2)

  Compute the partition function for all distance classes.

13.37.1 Detailed Description

Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.

13.37.2 Data Structure Documentation

13.37.2.1 struct vrna_sol_TwoD_pf_t

Solution element returned from vrna_pf_TwoD()

This element contains the partition function for the appropriate kappa (k), lambda (l) neighborhood The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'q' of type FLT_OR_DBL

A value of INF in k denotes the end of a list

See also

vrna_pf_TwoD()

Data Fields

- int k
  Distance to first reference.
- int l
  Distance to second reference.
- FLT_OR_DBL q
  partition function

13.37.3 Typedef Documentation

13.37.3.1 typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t

```
#include <ViennaRNA/2Dpfold.h>
```

Solution element returned from vrna_pf_TwoD()

This element contains the partition function for the appropriate kappa (k), lambda (l) neighborhood The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'q' of type FLT_OR_DBL

A value of INF in k denotes the end of a list

See also

vrna_pf_TwoD()

13.37.4 Function Documentation

13.37.4.1 vrna_sol_TwoD_pf_t* vrna_pf_TwoD (vrna_fold_compound_t * *vc*, int *maxDistance1*, int *maxDistance2*)

```
#include <ViennaRNA/2Dpfold.h>
```

Compute the partition function for all distance classes.

This function computes the partition functions for all distance classes according the two reference structures specified in the datastructure 'vars'. Similar to vrna_mfe_TwoD() the arguments maxDistance1 and maxDistance2 specify the maximum distance to both reference structures. A value of '-1' in either of them makes the appropriate distance restrictionless, i.e. all basepair distancies to the reference are taken into account during computation. In case there is a restriction, the returned solution contains an entry where the attribute k=l=-1 contains the partition function for all structures exceeding the restriction. A value of INF in the attribute 'k' of the returned list denotes the end of the list

See also

vrna_fold_compound_TwoD(), vrna_fold_compound_free(), vrna_fold_compound vrna_sol_TwoD_pf_t

Parameters

vc	The datastructure containing all necessary folding attributes and matrices
maxDistance1	The maximum basepair distance to reference1 (may be -1)
maxDistance2	The maximum basepair distance to reference2 (may be -1)

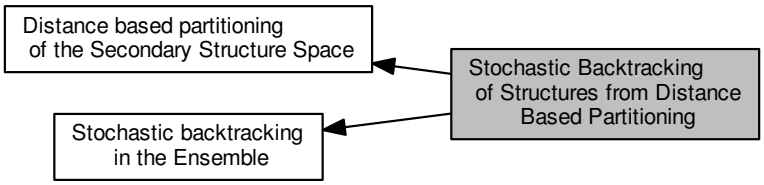
Returns

A list of partition funtions for the corresponding distance classes

13.38 Stochastic Backtracking of Structures from Distance Based Partitioning

Contains functions related to stochastic backtracking from a specified distance class.

Collaboration diagram for Stochastic Backtracking of Structures from Distance Based Partitioning:



Functions

- char * vrna_pbacktrack_TwoD (vrna_fold_compound_t *vc, int d1, int d2)

  Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.
- char * vrna_pbacktrack5_TwoD (vrna_fold_compound_t *vc, int d1, int d2, unsigned int length)

  Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

13.38.1 Detailed Description

Contains functions related to stochastic backtracking from a specified distance class.

13.38.2 Function Documentation

13.38.2.1 char* vrna_pbacktrack_TwoD (vrna_fold_compound_t * *vc*, int *d1*, int *d2*)

```
#include <ViennaRNA/2Dpfold.h>
```

Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

If the argument 'd1' is set to '-1', the structure will be backtracked in the distance class where all structures exceeding the maximum basepair distance to either of the references reside.

Precondition

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to vrna_pf_TwoD() preceding this function is mandatory!

See also

vrna_pf_TwoD()

Parameters

in,out	*vc*	The vrna_fold_compound_t datastructure containing all necessary folding attributes and matrices
in	*d1*	The distance to reference1 (may be -1)
in	*d2*	The distance to reference2

Returns

A sampled secondary structure in dot-bracket notation

13.38.2.2 char* vrna_pbacktrack5_TwoD (vrna_fold_compound_t * *vc*, int *d1*, int *d2*, unsigned int *length*)

```
#include <ViennaRNA/2Dpfold.h>
```

Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

This function does essentially the same as vrna_pbacktrack_TwoD() with the only difference that partial structures, i.e. structures beginning from the 5' end with a specified length of the sequence, are backtracked

Note

This function does not work (since it makes no sense) for circular RNA sequences!

Precondition

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to vrna_pf_TwoD() preceding this function is mandatory!

See also

vrna_pbacktrack_TwoD(), vrna_pf_TwoD()

Parameters

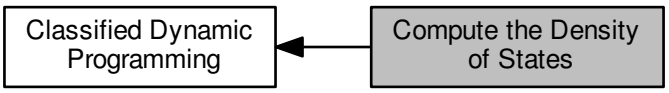
in,out	*vc*	The vrna_fold_compound_t datastructure containing all necessary folding attributes and matrices
in	*d1*	The distance to reference1 (may be -1)
in	*d2*	The distance to reference2
in	*length*	The length of the structure beginning from the 5' end

Returns

A sampled secondary structure in dot-bracket notation

13.39 Compute the Density of States

Collaboration diagram for Compute the Density of States:



Variables

- int density_of_states [MAXDOS+1]
  The Density of States.

13.39.1 Detailed Description

13.39.2 Variable Documentation

13.39.2.1 int density_of_states[MAXDOS+1]

```
#include <ViennaRNA/subopt.h>
```

The Density of States.

This array contains the density of states for an RNA sequences after a call to subopt_par(), subopt() or subopt_circ().

Precondition

Call one of the functions subopt_par(), subopt() or subopt_circ() prior accessing the contents of this array

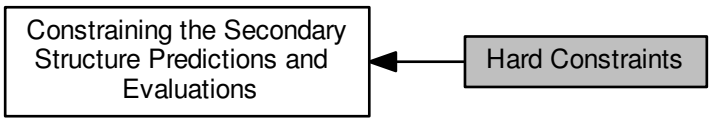
See also

subopt_par(), subopt(), subopt_circ()

13.40 Hard Constraints

This module covers all functionality for hard constraints in secondary structure prediction.

Collaboration diagram for Hard Constraints:



Files

- file constraints_hard.h

  Functions and data structures for handling of secondary structure hard constraints.

Data Structures

- struct vrna_hc_s

  The hard constraints data structure. More...
- struct vrna_hc_up_s

  A single hard constraint for a single nucleotide. More...

Macros

- #define VRNA_CONSTRAINT_DB 16384U

  Flag for vrna_constraints_add() to indicate that constraint is passed in pseudo dot-bracket notation.
- #define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U

  Switch for dot-bracket structure constraint to enforce base pairs.
- #define VRNA_CONSTRAINT_DB_PIPE 65536U

  Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.
- #define VRNA_CONSTRAINT_DB_DOT 131072U

  dot '.' switch for structure constraints (no constraint at all)
- #define VRNA_CONSTRAINT_DB_X 262144U

  'x' switch for structure constraint (base must not pair)
- #define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U

  round brackets '(',')' switch for structure constraint (base i pairs base j)
- #define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U

  Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.
- #define VRNA_CONSTRAINT_DB_INTERMOL 4194304U

  Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.
- #define VRNA_CONSTRAINT_DB_GQUAD 8388608U

  '+' switch for structure constraint (base is involved in a gquad)

- #define VRNA_CONSTRAINT_DB_DEFAULT

  Switch for dot-bracket structure constraint with default symbols.
- #define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP (char)0x01

  Hard constraints flag, base pair in the exterior loop.
- #define VRNA_CONSTRAINT_CONTEXT_HP_LOOP (char)0x02

  Hard constraints flag, base pair encloses hairpin loop.
- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP (char)0x04

  Hard constraints flag, base pair encloses an interior loop.
- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC (char)0x08

  Hard constraints flag, base pair encloses a multi branch loop.
- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP (char)0x10

  Hard constraints flag, base pair is enclosed in an interior loop.
- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC (char)0x20

  Hard constraints flag, base pair is enclosed in a multi branch loop.
- #define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

  Hard constraints flag, shortcut for all base pairs.

Typedefs

- typedef struct vrna_hc_s vrna_hc_t

  Typename for the hard constraints data structure vrna_hc_s.
- typedef struct vrna_hc_up_s vrna_hc_up_t

  Typename for the single nucleotide hard constraint data structure vrna_hc_up_s.
- typedef char(vrna_callback_hc_evaluate) (int i, int j, int k, int l, char d, void *data)

  Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.

Functions

- void vrna_hc_init (vrna_fold_compound_t *vc)

  Initialize/Reset hard constraints to default values.
- void vrna_hc_add_up (vrna_fold_compound_t *vc, int i, char option)

  Make a certain nucleotide unpaired.
- int vrna_hc_add_up_batch (vrna_fold_compound_t *vc, vrna_hc_up_t *constraints)

  Apply a list of hard constraints for single nucleotides.
- void vrna_hc_add_bp (vrna_fold_compound_t *vc, int i, int j, char option)

  Favorize/Enforce a certain base pair (i,j)
- void vrna_hc_add_bp_nonspecific (vrna_fold_compound_t *vc, int i, int d, char option)

  Enforce a nucleotide to be paired (upstream/downstream)
- void vrna_hc_free (vrna_hc_t *hc)

  Free the memory allocated by a vrna_hc_t data structure.
- int vrna_hc_add_from_db (vrna_fold_compound_t *vc, const char *constraint, unsigned int options)

  Add hard constraints from pseudo dot-bracket notation.

13.40.1 Detailed Description

This module covers all functionality for hard constraints in secondary structure prediction.

13.40.2 Data Structure Documentation

13.40.2.1 struct vrna_hc_s

The hard constraints data structure.

The content of this data structure determines the decomposition pattern used in the folding recursions. Attribute 'matrix' is used as source for the branching pattern of the decompositions during all folding recursions. Any entry in matrix[i,j] consists of the 6 LSB that allows to distinguish the following types of base pairs:

- in the exterior loop (VRNA_CONSTRAINT_CONTEXT_EXT_LOOP)
- enclosing a hairpin (VRNA_CONSTRAINT_CONTEXT_HP_LOOP)
- enclosing an interior loop (VRNA_CONSTRAINT_CONTEXT_INT_LOOP)
- enclosed by an exterior loop (VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC)
- enclosing a multi branch loop (VRNA_CONSTRAINT_CONTEXT_MB_LOOP)
- enclosed by a multi branch loop (VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC)

The four linear arrays 'up_xxx' provide the number of available unpaired nucleotides (including position i) 3' of each position in the sequence.

See also

vrna_hc_init(), vrna_hc_free(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC

Data Fields

- char * matrix

  Upper triangular matrix that encodes where a base pair or unpaired nucleotide is allowed.
- int * up_ext

  A linear array that holds the number of allowed unpaired nucleotides in an exterior loop.
- int * up_hp

  A linear array that holds the number of allowed unpaired nucleotides in a hairpin loop.
- int * up_int

  A linear array that holds the number of allowed unpaired nucleotides in an interior loop.
- int * up_ml

  A linear array that holds the number of allowed unpaired nucleotides in a multi branched loop.
- vrna_callback_hc_evaluate * f

  A function pointer that returns whether or not a certain decomposition may be evaluated.
- void * data

  A pointer to some structure where the user may store necessary data to evaluate its generic hard constraint function.
- vrna_callback_free_auxdata * free_data

  A pointer to a function to free memory occupied by auxiliary data.

13.40.2.1.1 Field Documentation

13.40.2.1.1.1 vrna_callback_free_auxdata* vrna_hc_s::free_data

A pointer to a function to free memory occupied by auxiliary data.

The function this pointer is pointing to will be called upon destruction of the vrna_hc_s, and provided with the vrna_hc_s.data pointer that may hold auxiliary data. Hence, to avoid leaking memory, the user may use this pointer to free memory occupied by auxiliary data.

13.40.2.2 struct vrna_hc_up_s

A single hard constraint for a single nucleotide.

Data Fields

- int position

  The sequence position (1-based)
- char options

  The hard constraint option.

13.40.3 Macro Definition Documentation

13.40.3.1 #define VRNA_CONSTRAINT_DB 16384U

```
#include <ViennaRNA/constraints_hard.h>
```

Flag for vrna_constraints_add() to indicate that constraint is passed in pseudo dot-bracket notation.

See also

vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.2 #define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U

```
#include <ViennaRNA/constraints_hard.h>
```

Switch for dot-bracket structure constraint to enforce base pairs.

This flag should be used to really enforce base pairs given in dot-bracket constraint rather than just weakly-enforcing them.

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.3 #define VRNA_CONSTRAINT_DB_PIPE 65536U

```
#include <ViennaRNA/constraints_hard.h>
```

Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the pipe sign '|' (paired with another base)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.4 #define VRNA_CONSTRAINT_DB_DOT 131072U

```
#include <ViennaRNA/constraints_hard.h>
```

dot '.' switch for structure constraints (no constraint at all)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.5 #define VRNA_CONSTRAINT_DB_X 262144U

```
#include <ViennaRNA/constraints_hard.h>
```

'x' switch for structure constraint (base must not pair)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.6 #define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U

```
#include <ViennaRNA/constraints_hard.h>
```

round brackets '(',')' switch for structure constraint (base i pairs base j)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.7 #define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U

```
#include <ViennaRNA/constraints_hard.h>
```

Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'l' character (intramolecular pairs only)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.8 #define VRNA_CONSTRAINT_DB_INTERMOL 4194304U

```
#include <ViennaRNA/constraints_hard.h>
```

Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'e' character (intermolecular pairs only)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.3.9 #define VRNA_CONSTRAINT_DB_GQUAD 8388608U

```
#include <ViennaRNA/constraints_hard.h>
```

'+' switch for structure constraint (base is involved in a gquad)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

Warning

This flag is for future purposes only! No implementation recognizes it yet.

13.40.3.10 #define VRNA_CONSTRAINT_DB_DEFAULT

```
#include <ViennaRNA/constraints_hard.h>
```

Value:

```
(   VRNA_CONSTRAINT_DB \
      | VRNA_CONSTRAINT_DB_PIPE \
      | VRNA_CONSTRAINT_DB_DOT \
      | VRNA_CONSTRAINT_DB_X \
      | VRNA_CONSTRAINT_DB_ANG_BRACK \
      | VRNA_CONSTRAINT_DB_RND_BRACK \
      | VRNA_CONSTRAINT_DB_INTRAMOL \
      | VRNA_CONSTRAINT_DB_INTERMOL \
      | VRNA_CONSTRAINT_DB_GQUAD \
    )
```

Switch for dot-bracket structure constraint with default symbols.

This flag conveniently combines all possible symbols in dot-bracket notation for hard constraints and VRNA_CONSTRAINT_DB

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

13.40.4 Typedef Documentation

13.40.4.1 typedef char(vrna_callback_hc_evaluate) (int i, int j, int k, int l, char d, void *data)

```
#include <ViennaRNA/constraints_hard.h>
```

Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.

This is the prototype for callback functions used by the folding recursions to evaluate generic hard constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via vrna_hc_add_data(), or NULL if no auxiliary data was added.

See also

VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_hc_add_f(), vrna_hc_add_data()

Parameters

i	Left (5') delimiter position of substructure
j	Right (3') delimiter position of substructure
k	Left delimiter of decomposition
l	Right delimiter of decomposition
d	Decomposition step indicator
data	Auxiliary data

Returns

Pseudo energy contribution in deka-kalories per mol

13.40.5 Function Documentation

13.40.5.1 void vrna_hc_init (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/constraints_hard.h>
```

Initialize/Reset hard constraints to default values.

This function resets the hard constraints to their default values, i.e. all positions may be unpaired in all contexts, and base pairs are allowed in all contexts, if they resemble canonical pairs. Previously set hard constraints will be removed vefore initialization.

See also

vrna_hc_add_bp(), vrna_hc_add_bp_nonspecific(), vrna_hc_add_up()

Parameters

vc	The fold compound

13.40.5.2 void vrna_hc_add_up (vrna_fold_compound_t * *vc*, int *i*, char *option*)

```
#include <ViennaRNA/constraints_hard.h>
```

Make a certain nucleotide unpaired.

See also

vrna_hc_add_bp(), vrna_hc_add_bp_nonspecific(), vrna_hc_init(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

Parameters

vc	The vrna_fold_compound_t the hard constraints are associated with
i	The position that needs to stay unpaired (1-based)
option	The options flag indicating how/where to store the hard constraints

13.40.5.3 int vrna_hc_add_up_batch (vrna_fold_compound_t * *vc*, vrna_hc_up_t * *constraints*)

```
#include <ViennaRNA/constraints_hard.h>
```

Apply a list of hard constraints for single nucleotides.

Parameters

vc	The vrna_fold_compound_t the hard constraints are associated with
constraints	The list off constraints to apply, last entry must have position attribute set to 0

13.40.5.4 void vrna_hc_add_bp (vrna_fold_compound_t * *vc*, int *i*, int *j*, char *option*)

```
#include <ViennaRNA/constraints_hard.h>
```

Favorize/Enforce a certain base pair (i,j)

See also

vrna_hc_add_bp_nonspecific(), vrna_hc_add_up(), vrna_hc_init(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC, VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC, VRNA_CONSTRAINT_CONTEXT_ENFORCE, VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

Parameters

vc	The vrna_fold_compound_t the hard constraints are associated with
i	The 5' located nucleotide position of the base pair (1-based)
j	The 3' located nucleotide position of the base pair (1-based)
option	The options flag indicating how/where to store the hard constraints

13.40.5.5 void vrna_hc_add_bp_nonspecific (vrna_fold_compound_t * *vc*, int *i*, int *d*, char *option*)

```
#include <ViennaRNA/constraints_hard.h>
```

Enforce a nucleotide to be paired (upstream/downstream)

See also

vrna_hc_add_bp(), vrna_hc_add_up(), vrna_hc_init(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC, VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC, VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

Parameters

vc	The vrna_fold_compound_t the hard constraints are associated with
i	The position that needs to stay unpaired (1-based)
d	The direction of base pairing ($d < 0$: pairs upstream, $d > 0$: pairs downstream, $d == 0$: no direction)
option	The options flag indicating in which loop type context the pairs may appear

13.40.5.6 void vrna_hc_free (vrna_hc_t * *hc*)

```
#include <ViennaRNA/constraints_hard.h>
```

Free the memory allocated by a vrna_hc_t data structure.

Use this function to free all memory that was allocated for a data structure of type vrna_hc_t .

See also

get_hard_constraints(), vrna_hc_t

13.40.5.7 int vrna_hc_add_from_db (vrna_fold_compound_t * *vc*, const char * *constraint*, unsigned int *options*)

```
#include <ViennaRNA/constraints_hard.h>
```

Add hard constraints from pseudo dot-bracket notation.

This function allows one to apply hard constraints from a pseudo dot-bracket notation. The `options` parameter controls, which characters are recognized by the parser. Use the VRNA_CONSTRAINT_DB_DEFAULT convenience macro, if you want to allow all known characters

See also

VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTRAMOL, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_GQUAD

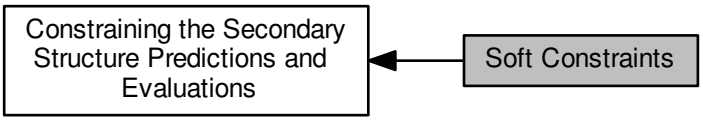
Parameters

vc	The fold compound
constraint	A pseudo dot-bracket notation of the hard constraint.
options	The option flags

13.41 Soft Constraints

Functions and data structures for secondary structure soft constraints.

Collaboration diagram for Soft Constraints:



Files

- file constraints_soft.h

  Functions and data structures for secondary structure soft constraints.

Data Structures

- struct vrna_sc_s

  The soft constraints data structure. More...

Typedefs

- typedef struct vrna_sc_s vrna_sc_t

  Typename for the soft constraints data structure vrna_sc_s.
- typedef int(vrna_callback_sc_energy) (int i, int j, int k, int l, char d, void *data)

  Callback to retrieve pseudo energy contribution for soft constraint feature.
- typedef FLT_OR_DBL(vrna_callback_sc_exp_energy) (int i, int j, int k, int l, char d, void *data)

  Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.
- typedef vrna_basepair_t *(vrna_callback_sc_backtrack) (int i, int j, int k, int l, char d, void *data)

  Callback to retrieve auxiliary base pairs for soft constraint feature.

Functions

- void vrna_sc_init (vrna_fold_compound_t *vc)

  Initialize an empty soft constraints data structure within a vrna_fold_compound_t.
- void vrna_sc_add_bp (vrna_fold_compound_t *vc, const FLT_OR_DBL **constraints, unsigned int options)

  Add soft constraints for paired nucleotides.
- void vrna_sc_add_up (vrna_fold_compound_t *vc, const FLT_OR_DBL *constraints, unsigned int options)

  Add soft constraints for unpaired nucleotides.
- void vrna_sc_remove (vrna_fold_compound_t *vc)

  Remove soft constraints from vrna_fold_compound_t.

- void vrna_sc_free (vrna_sc_t *sc)

  Free memory occupied by a vrna_sc_t data structure.
- void vrna_sc_add_data (vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *free_data)

  Add an auxiliary data structure for the generic soft constraints callback function.
- void vrna_sc_add_f (vrna_fold_compound_t *vc, vrna_callback_sc_energy *f)

  Bind a function pointer for generic soft constraint feature (MFE version)
- void vrna_sc_add_bt (vrna_fold_compound_t *vc, vrna_callback_sc_backtrack *f)

  Bind a backtracking function pointer for generic soft constraint feature.
- void vrna_sc_add_exp_f (vrna_fold_compound_t *vc, vrna_callback_sc_exp_energy *exp_f)

  Bind a function pointer for generic soft constraint feature (PF version)

13.41.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

Soft-constraints are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations.

13.41.2 Data Structure Documentation

13.41.2.1 struct vrna_sc_s

The soft constraints data structure.

Data Fields

- int ** energy_up

  Energy contribution for stretches of unpaired nucleotides.
- int * energy_bp

  Energy contribution for base pairs.
- FLT_OR_DBL ** exp_energy_up

  Boltzmann Factors of the energy contributions for unpaired sequence stretches.
- FLT_OR_DBL * exp_energy_bp

  Boltzmann Factors of the energy contribution for base pairs.
- int * energy_stack

  Pseudo Energy contribution per base pair involved in a stack.
- FLT_OR_DBL * exp_energy_stack

  Boltzmann weighted pseudo energy contribution per nucleotide involved in a stack.
- vrna_callback_sc_energy * f

  A function pointer used for pseudo energy contribution in MFE calculations.
- vrna_callback_sc_backtrack * bt

  A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.
- vrna_callback_sc_exp_energy * exp_f

  A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.
- void * data

  A pointer to the data object provided for for pseudo energy contribution functions of the generic soft constraints feature.

13.41.2.1.1 Field Documentation

13.41.2.1.1.1 vrna_callback_sc_energy* vrna_sc_s::f

A function pointer used for pseudo energy contribution in MFE calculations.

See also

vrna_sc_add_f()

13.41.2.1.1.2 vrna_callback_sc_backtrack* vrna_sc_s::bt

A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.

See also

vrna_sc_add_bt()

13.41.2.1.1.3 vrna_callback_sc_exp_energy* vrna_sc_s::exp_f

A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.

See also

vrna_sc_add_exp_f()

13.41.3 Typedef Documentation

13.41.3.1 typedef int(vrna_callback_sc_energy) (int i, int j, int k, int l, char d, void *data)

```
#include <ViennaRNA/constraints_soft.h>
```

Callback to retrieve pseudo energy contribution for soft constraint feature.

This is the prototype for callback functions used by the folding recursions to evaluate generic soft constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via vrna_sc_add_data(), or NULL if no auxiliary data was added.

See also

VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_↩ ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_↩ DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECO↩ MP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT↩ _STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_bt(), vrna_sc_add_data()

Parameters

i	Left (5') delimiter position of substructure
j	Right (3') delimiter position of substructure
k	Left delimiter of decomposition
l	Right delimiter of decomposition
d	Decomposition step indicator
data	Auxiliary data

Returns

Pseudo energy contribution in deka-kalories per mol

13.41.3.2 typedef FLT_OR_DBL(vrna_callback_sc_exp_energy) (int i, int j, int k, int l, char d, void *data)

```
#include <ViennaRNA/constraints_soft.h>
```

Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.

This is the prototype for callback functions used by the partition function recursions to evaluate generic soft constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via vrna_sc_add_data(), or NULL if no auxiliary data was added.

See also

VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_sc_add_exp_f(), vrna_sc_add_f(), vrna_sc_add_bt(), vrna_sc_add_data()

Parameters

i	Left (5') delimiter position of substructure
j	Right (3') delimiter position of substructure
k	Left delimiter of decomposition
l	Right delimiter of decomposition
d	Decomposition step indicator
data	Auxiliary data

Returns

Pseudo energy contribution in deka-kalories per mol

13.41.3.3 typedef vrna_basepair_t*(vrna_callback_sc_backtrack) (int i, int j, int k, int l, char d, void *data)

```
#include <ViennaRNA/constraints_soft.h>
```

Callback to retrieve auxiliary base pairs for soft constraint feature.

See also

VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_sc_add_bt(), vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_data()

Parameters

i	Left (5') delimiter position of substructure
j	Right (3') delimiter position of substructure
k	Left delimiter of decomposition
l	Right delimiter of decomposition
d	Decomposition step indicator
data	Auxiliary data

Returns

List of additional base pairs

13.41.4 Function Documentation

13.41.4.1 void vrna_sc_init (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/constraints_soft.h>
```

Initialize an empty soft constraints data structure within a vrna_fold_compound_t.

This function adds a proper soft constraints data structure to the vrna_fold_compound_t data structure. If soft constraints already exist within the fold compound, they are removed.

Note

Accepts vrna_fold_compound_t of type VRNA_VC_TYPE_SINGLE and VRNA_VC_TYPE_ALIGNMENT

See also

vrna_sc_add_bp(), vrna_sc_add_up(), vrna_sc_add_SHAPE_deigan(), vrna_sc_add_SHAPE_zarringhalam(), vrna_sc_remove(), vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_pre(), vrna_sc_add_post()

Parameters

vc	The vrna_fold_compound_t where an empty soft constraint feature is to be added to

13.41.4.2 void vrna_sc_add_bp (vrna_fold_compound_t * *vc*, const FLT_OR_DBL ** *constraints*, unsigned int *options*)

```
#include <ViennaRNA/constraints_soft.h>
```

Add soft constraints for paired nucleotides.

Parameters

vc	The vrna_fold_compound_t the soft constraints are associated with
constraints	A two-dimensional array of pseudo free energies in $kcal/mol$
options	The options flag indicating how/where to store the soft constraints

13.41.4.3 void vrna_sc_add_up (vrna_fold_compound_t * *vc*, const FLT_OR_DBL * *constraints*, unsigned int *options*)

```
#include <ViennaRNA/constraints_soft.h>
```

Add soft constraints for unpaired nucleotides.

Parameters

vc	The vrna_fold_compound_t the soft constraints are associated with
constraints	A vector of pseudo free energies in $kcal/mol$
options	The options flag indicating how/where to store the soft constraints

13.41.4.4 void vrna_sc_remove (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/constraints_soft.h>
```

Remove soft constraints from vrna_fold_compound_t.

Note

Accepts vrna_fold_compound_t of type VRNA_VC_TYPE_SINGLE and VRNA_VC_TYPE_ALIGNMENT

Parameters

vc	The vrna_fold_compound_t possibly containing soft constraints

13.41.4.5 void vrna_sc_free (vrna_sc_t * *sc*)

```
#include <ViennaRNA/constraints_soft.h>
```

Free memory occupied by a vrna_sc_t data structure.

Parameters

sc	The data structure to free from memory

13.41.4.6 void vrna_sc_add_data (vrna_fold_compound_t * *vc*, void * *data*, vrna_callback_free_auxdata * *free_data*)

```
#include <ViennaRNA/constraints_soft.h>
```

Add an auxiliary data structure for the generic soft constraints callback function.

See also

vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_bt()

Parameters

vc	The fold compound the generic soft constraint function should be bound to
data	A pointer to the data structure that holds required data for function 'f'
free_data	A pointer to a function that free's the memory occupied by `data` (Maybe NULL)

13.41.4.7 void vrna_sc_add_f (vrna_fold_compound_t * *vc*, vrna_callback_sc_energy * *f*)

```
#include <ViennaRNA/constraints_soft.h>
```

Bind a function pointer for generic soft constraint feature (MFE version)

This function allows to easily bind a function pointer and corresponding data structure to the soft constraint part vrna_sc_t of the vrna_fold_compound_t. The function for evaluating the generic soft constraint feature has to return a pseudo free energy $\hat{E}$ in $dacal/mol$, where $1dacal/mol = 10cal/mol$.

See also

vrna_sc_add_data(), vrna_sc_add_bt(), vrna_sc_add_exp_f()

Parameters

vc	The fold compound the generic soft constraint function should be bound to
f	A pointer to the function that evaluates the generic soft constraint feature

13.41.4.8 void vrna_sc_add_bt (vrna_fold_compound_t * *vc*, vrna_callback_sc_backtrack * *f*)

```
#include <ViennaRNA/constraints_soft.h>
```

Bind a backtracking function pointer for generic soft constraint feature.

This function allows to easily bind a function pointer to the soft constraint part vrna_sc_t of the vrna_fold_compound_t. The provided function should be used for backtracking purposes in loop regions that were altered via the generic soft constraint feature. It has to return an array of vrna_basepair_t data structures, were the last element in the list is indicated by a value of -1 in it's i position.

See also

vrna_sc_add_data(), vrna_sc_add_f(), vrna_sc_add_exp_f()

Parameters

vc	The fold compound the generic soft constraint function should be bound to
f	A pointer to the function that returns additional base pairs

13.41.4.9 void vrna_sc_add_exp_f (vrna_fold_compound_t * *vc*, vrna_callback_sc_exp_energy * *exp_f*)

```
#include <ViennaRNA/constraints_soft.h>
```

Bind a function pointer for generic soft constraint feature (PF version)

This function allows to easily bind a function pointer and corresponding data structure to the soft constraint part vrna_sc_t of the vrna_fold_compound_t. The function for evaluating the generic soft constraint feature has to return a pseudo free energy $\hat{E}$ as Boltzmann factor, i.e. $exp(-\hat{E}/kT)$. The required unit for E is cal/mol.

See also

vrna_sc_add_bt(), vrna_sc_add_f(), vrna_sc_add_data()

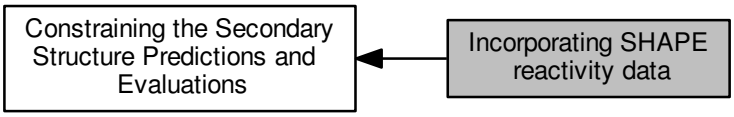
Parameters

vc	The fold compound the generic soft constraint function should be bound to
exp_f	A pointer to the function that evaluates the generic soft constraint feature

13.42 Incorporating SHAPE reactivity data

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

Collaboration diagram for Incorporating SHAPE reactivity data:



Files

- file constraints_SHAPE.h

  This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

Functions

- int vrna_sc_add_SHAPE_deigan (vrna_fold_compound_t *vc, const double *reactivities, double m, double b, unsigned int options)

  Add SHAPE reactivity data as soft constraints (Deigan et al. method)

- int vrna_sc_add_SHAPE_deigan_ali (vrna_fold_compound_t *vc, const char **shape_files, const int *shape_file_association, double m, double b, unsigned int options)

  Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)

- int vrna_sc_add_SHAPE_zarringhalam (vrna_fold_compound_t *vc, const double *reactivities, double b, double default_value, const char *shape_conversion, unsigned int options)

  Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)

- int vrna_sc_SHAPE_to_pr (const char *shape_conversion, double *values, int length, double default_value)

  Convert SHAPE reactivity values to probabilities for being unpaired.

13.42.1 Detailed Description

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

13.42.2 Function Documentation

13.42.2.1 int vrna_sc_add_SHAPE_deigan (vrna_fold_compound_t * *vc*, const double * *reactivities*, double *m*, double *b*, unsigned int *options*)

```
#include <ViennaRNA/constraints_SHAPE.h>
```

Add SHAPE reactivity data as soft constraints (Deigan et al. method)

This approach of SHAPE directed RNA folding uses the simple linear ansatz

$$\Delta G_{\text{SHAPE}}(i) = m \ln(\text{SHAPE reactivity}(i) + 1) + b$$

to convert SHAPE reactivity values to pseudo energies whenever a nucleotide i contributes to a stacked pair. A positive slope m penalizes high reactivities in paired regions, while a negative intercept b results in a confirmatory “bonus" free energy for correctly predicted base pairs. Since the energy evaluation of a base pair stack involves two pairs, the pseudo energies are added for all four contributing nucleotides. Consequently, the energy term is applied twice for pairs inside a helix and only once for pairs adjacent to other structures. For all other loop types the energy model remains unchanged even when the experimental data highly disagrees with a certain motif.

See also

For further details, we refer to [4].
vrna_sc_remove(), vrna_sc_add_SHAPE_zarringhalam(), vrna_sc_minimize_pertubation()

Parameters

vc	The vrna_fold_compound_t the soft constraints are associated with
reactivities	A vector of normalized SHAPE reactivities
m	The slope of the conversion function
b	The intercept of the conversion function
options	The options flag indicating how/where to store the soft constraints

Returns

1 on successful extraction of the method, 0 on errors

13.42.2.2 int vrna_sc_add_SHAPE_deigan_ali (vrna_fold_compound_t * *vc*, const char ** *shape_files*, const int * *shape_file_association*, double *m*, double *b*, unsigned int *options*)

```
#include <ViennaRNA/constraints_SHAPE.h>
```

Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)

Parameters

vc	The vrna_fold_compound_t the soft constraints are associated with
shape_files	A set of filenames that contain normalized SHAPE reactivity data
shape_file_association	An array of integers that associate the files with sequences in the alignment
m	The slope of the conversion function
b	The intercept of the conversion function
options	The options flag indicating how/where to store the soft constraints

Returns

1 on successful extraction of the method, 0 on errors

13.42.2.3 int vrna_sc_add_SHAPE_zarringhalam (vrna_fold_compound_t * *vc*, const double * *reactivities*, double *b*, double *default_value*, const char * *shape_conversion*, unsigned int *options*)

```
#include <ViennaRNA/constraints_SHAPE.h>
```

Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)

This method first converts the observed SHAPE reactivity of nucleotide i into a probability q_i that position i is unpaired by means of a non-linear map. Then pseudo-energies of the form

$$\Delta G_{\text{SHAPE}}(x, i) = \beta \, |x_i - q_i|$$

are computed, where $x_i = 0$ if position i is unpaired and $x_i = 1$ if i is paired in a given secondary structure. The parameter β serves as scaling factor. The magnitude of discrepancy between prediction and experimental observation is represented by $|x_i - q_i|$.

See also

For further details, we refer to [19]
vrna_sc_remove(), vrna_sc_add_SHAPE_deigan(), vrna_sc_minimize_pertubation()

Parameters

vc	The vrna_fold_compound_t the soft constraints are associated with
reactivities	A vector of normalized SHAPE reactivities
b	The scaling factor β of the conversion function
default_value	The default value for a nucleotide where reactivity data is missing for
shape_conversion	A flag that specifies how to convert reactivities to probabilities
options	The options flag indicating how/where to store the soft constraints

Returns

1 on successful extraction of the method, 0 on errors

13.42.2.4 int vrna_sc_SHAPE_to_pr (const char * *shape_conversion*, double * *values*, int *length*, double *default_value*)

```
#include <ViennaRNA/constraints_SHAPE.h>
```

Convert SHAPE reactivity values to probabilities for being unpaired.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the FLT_OR_DBL array values.

See also

vrna_file_SHAPE_read()

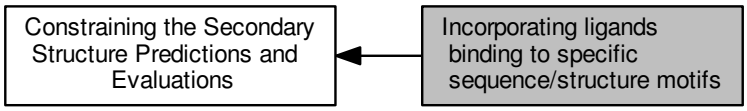
Parameters

shape_conversion	String defininining the method used for the conversion process
values	Pointer to an array of SHAPE reactivities
length	Length of the array of SHAPE reactivities
default_value	Result used for position with invalid/missing reactivity values

13.43 Incorporating ligands binding to specific sequence/structure motifs

This module covers functions that enable the incorporation of ligand binding free energies to specific hairpin/interior loop motifs by means of generic soft constraints.

Collaboration diagram for Incorporating ligands binding to specific sequence/structure motifs:



Files

- file ligand.h

  Functions for incorporation of ligands binding to haipirn and interior loop motifs.

Functions

- int vrna_sc_add_hi_motif (vrna_fold_compound_t *vc, const char *seq, const char *structure, FLT_OR_DBL energy, unsigned int options)

  Add soft constraints for hairpin or interior loop binding motif.

13.43.1 Detailed Description

This module covers functions that enable the incorporation of ligand binding free energies to specific hairpin/interior loop motifs by means of generic soft constraints.

13.43.2 Function Documentation

13.43.2.1 int vrna_sc_add_hi_motif (vrna_fold_compound_t * *vc*, const char * *seq*, const char * *structure*, FLT_OR_DBL *energy*, unsigned int *options*)

```
#include <ViennaRNA/ligand.h>
```

Add soft constraints for hairpin or interior loop binding motif.

Here is an example that adds a theophylline binding motif. Free energy contribution is derived from $k_d = 0.32 \mu mol/l$, taken from Jenison et al. 1994

```
vrna_sc_add_hi_motif( vc,
                      "GAUACCAG&CCCUUGGCAGC",
                      "(...((((&)...)))...)",
                      -9.22, VRNA_OPTION_DEFAULT);
```

Parameters

vc	The vrna_fold_compound_t the motif is applied to
seq	The sequence motif (may be interspaced by '&' character
structure	The structure motif (may be interspaced by '&' character
energy	The free energy of the motif (e.g. binding free energy)
options	Options

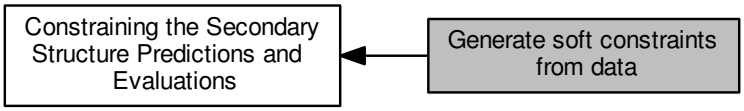
Returns

non-zero value if application of the motif using soft constraints was successful

13.44 Generate soft constraints from data

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Collaboration diagram for Generate soft constraints from data:



Files

- file perturbation_fold.h

  Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Macros

- #define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

  Use the sum of squared aberrations as objective function.
- #define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

  Use the sum of absolute aberrations as objective function.
- #define VRNA_MINIMIZER_DEFAULT 0

  Use a custom implementation of the gradient descent algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_CONJUGATE_FR 1

  Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_CONJUGATE_PR 2

  Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_VECTOR_BFGS 3

  Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_VECTOR_BFGS2 4

  Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_STEEPEST_DESCENT 5

  Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.

Typedefs

- typedef void(∗ progress_callback) (int iteration, double score, double ∗epsilon)

  Callback for following the progress of the minimization process.

Functions

- void vrna_sc_minimize_pertubation (vrna_fold_compound_t *vc, const double *q_prob_unpaired, int objective_function, double sigma_squared, double tau_squared, int algorithm, int sample_size, double *epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback)

  Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

13.44.1 Detailed Description

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

13.44.2 Macro Definition Documentation

13.44.2.1 #define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the sum of squared aberrations as objective function.

$F(\vec{\epsilon}) = \sum_{i=1}^{n} \frac{\epsilon_i^2}{\tau^2} + \sum_{i=1}^{n} \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow min$

13.44.2.2 #define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the sum of absolute aberrations as objective function.

$F(\vec{\epsilon}) = \sum_{i=1}^{n} \frac{|\epsilon_i|}{\tau^2} + \sum_{i=1}^{n} \frac{|p_i(\vec{\epsilon}) - q_i|}{\sigma^2} \rightarrow min$

13.44.2.3 #define VRNA_MINIMIZER_CONJUGATE_FR 1

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

13.44.2.4 #define VRNA_MINIMIZER_CONJUGATE_PR 2

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

13.44.2.5 #define VRNA_MINIMIZER_VECTOR_BFGS 3

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

13.44.2.6 #define VRNA_MINIMIZER_VECTOR_BFGS2 4

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

13.44.2.7 #define VRNA_MINIMIZER_STEEPEST_DESCENT 5

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

13.44.3 Typedef Documentation

13.44.3.1 typedef void(* progress_callback) (int iteration, double score, double *epsilon)

```
#include <ViennaRNA/perturbation_fold.h>
```

Callback for following the progress of the minimization process.

Parameters

iteration	The number of the current iteration
score	The score of the objective function
epsilon	The perturbation vector yielding the reported score

13.44.4 Function Documentation

13.44.4.1 void vrna_sc_minimize_pertubation (vrna_fold_compound_t * *vc*, const double * *q_prob_unpaired*, int *objective_function*, double *sigma_squared*, double *tau_squared*, int *algorithm*, int *sample_size*, double * *epsilon*, double *initialStepSize*, double *minStepSize*, double *minImprovement*, double *minimizerTolerance*, progress_callback *callback*)

```
#include <ViennaRNA/perturbation_fold.h>
```

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Use an iterative minimization algorithm to find a vector of perturbation energies whose incorporation as soft constraints shifts the predicted pairing probabilities closer to the experimentally observed probabilities. The algorithm aims to minimize an objective function that penalizes discripancies between predicted and observed pairing probabilities and energy model adjustments, i.e. an appropriate vector of perturbation energies satisfies

$$F(\vec{\epsilon}) = \sum_{\mu} \frac{\epsilon_{\mu}^2}{\tau^2} + \sum_{i=1}^{n} \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow \min .$$

An initialized fold compound and an array containing the observed probability for each nucleotide to be unbound are required as input data. The parameters objective_function, sigma_squared and tau_squared are responsible for adjusting the aim of the objective function. Dependend on which type of objective function is selected, either squared or absolute aberrations are contributing to the objective function. The ratio of the parameters sigma_↩ squared and tau_squared can be used to adjust the algorithm to find a solution either close to the thermodynamic prediction (sigma_squared $>>$ tau_squared) or close to the experimental data (tau_squared $>>$ sigma_squared). The minimization can be performed by makeing use of a custom gradient descent implementation or using one of the minimizing algorithms provided by the GNU Scientific Library. All algorithms require the evaluation of the gradient of the objective function, which includes the evaluation of conditional pairing probabilites. Since an exact evaluation is expensive, the probabilities can also be estimated from sampling by setting an appropriate sample size. The found vector of perturbation energies will be stored in the array epsilon. The progress of the minimization process can be tracked by implementing and passing a callback function.

See also

For further details we refere to [17].

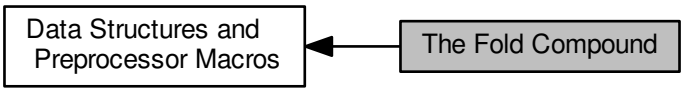
Parameters

vc	Pointer to a fold compound
q_prob_unpaired	Pointer to an array containing the probability to be unpaired for each nucleotide
objective_function	The type of objective function to be used (VRNA_OBJECTIVE_FUNCTION_QUADRATIC / VRNA_OBJECTIVE_FUNCTION_LINEAR)
sigma_squared	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the null vector.
tau_squared	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the data provided in q_prob_unpaired.
algorithm	The minimization algorithm (VRNA_MINIMIZER_$*$)
sample_size	The number of sampled sequences used for estimating the pairing probabilities. A value $<=$ 0 will lead to an exact evaluation.
epsilon	A pointer to an array used for storing the calculated vector of perturbation energies
callback	A pointer to a callback function used for reporting the current minimization progress

13.45 The Fold Compound

This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.

Collaboration diagram for The Fold Compound:



Data Structures

- struct vrna_fc_s

  The most basic data structure required by many functions throughout the RNAlib. More...

Macros

- #define VRNA_STATUS_MFE_PRE (unsigned char)1

  Status message indicating that MFE computations are about to begin.
- #define VRNA_STATUS_MFE_POST (unsigned char)2

  Status message indicating that MFE computations are finished.
- #define VRNA_STATUS_PF_PRE (unsigned char)3

  Status message indicating that Partition function computations are about to begin.
- #define VRNA_STATUS_PF_POST (unsigned char)4

  Status message indicating that Partition function computations are finished.
- #define VRNA_OPTION_DEFAULT 0U

  Option flag to specify default settings/requirements.
- #define VRNA_OPTION_MFE 1U

  Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.
- #define VRNA_OPTION_PF 2U

  Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.
- #define VRNA_OPTION_HYBRID 4U

  Option flag to specify requirement of dimer DP matrices.
- #define VRNA_OPTION_EVAL_ONLY 8U

  Option flag to specify that neither MFE, nor PF DP matrices are required.
- #define VRNA_OPTION_WINDOW 16U

  Option flag to specify requirement of DP matrices for local folding approaches.

Typedefs

- typedef struct vrna_fc_s vrna_fold_compound_t

  Typename for the fold_compound data structure vrna_fc_s.
- typedef void(vrna_callback_free_auxdata) (void *data)

  Callback to free memory allocated for auxiliary user-provided data.
- typedef void(vrna_callback_recursion_status) (unsigned char status, void *data)

  Callback to perform specific user-defined actions before, or after recursive computations.

Enumerations

Functions

- vrna_fold_compound_t * vrna_fold_compound (const char *sequence, vrna_md_t *md_p, unsigned int options)

  Retrieve a vrna_fold_compound_t data structure for single sequences and hybridizing sequences.
- vrna_fold_compound_t * vrna_fold_compound_comparative (const char **sequences, vrna_md_t *md_p, unsigned int options)

  Retrieve a vrna_fold_compound_t data structure for sequence alignments.
- void vrna_fold_compound_free (vrna_fold_compound_t *vc)

  Free memory occupied by a vrna_fold_compound_t.
- void vrna_fold_compound_add_auxdata (vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *f)

  Add auxiliary data to the vrna_fold_compound_t.
- void vrna_fold_compound_add_callback (vrna_fold_compound_t *vc, vrna_callback_recursion_status *f)

  Add a recursion status callback to the vrna_fold_compound_t.

13.45.1 Detailed Description

This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.

Throughout the entire RNAlib, the vrna_fold_compound_t, is used to group information and data that is required for structure prediction and energy evaluation. Here, you'll find interface functions to create, modify, and delete vrna_fold_compound_t data structures.

13.45.2 Data Structure Documentation

13.45.2.1 struct vrna_fc_s

The most basic data structure required by many functions throughout the RNAlib.

Note

Please read the documentation of this data structure carefully! Some attributes are only available for specific types this data structure can adopt.

Warning

Reading/Writing from/to attributes that are not within the scope of the current type usually result in undefined behavior!

See also

vrna_fold_compound_t.type, vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_↩ compound_free(), VRNA_VC_TYPE_SINGLE, VRNA_VC_TYPE_ALIGNMENT

Notes on Scripting Interface This data structure is wrapped as an object **fold_compound** with several related functions attached as methods.

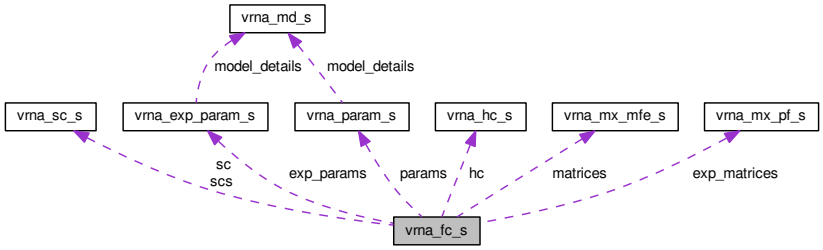
A new **fold_compound** can be obtained by calling one of its constructors:

- *fold_compound(seq)* – Initialize with a single sequence, or two concatenated sequences separated by an ampersand character '&' (for cofolding)
- *fold_compound(aln)* – Initialize with a sequence alignment *aln* stored as a list of sequences (with gap characters)

The resulting object has a list of attached methods which in most cases directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *type()* – Get the type of the *fold_compound* (See vrna_fc_type_e)
- *length()* – Get the length of the sequence(s) or alignment stored within the *fold_compound*

Collaboration diagram for vrna_fc_s:



Data Fields

Common data fields

- vrna_fc_type_e type
  The type of the vrna_fold_compound_t.
- unsigned int length
  The length of the sequence (or sequence alignment)
- int cutpoint
  The position of the (cofold) cutpoint within the provided sequence. If there is no cutpoint, this field will be set to -1.
- vrna_hc_t * hc
  The hard constraints data structure used for structure prediction.

- vrna_mx_mfe_t * matrices
  The MFE DP matrices.
- vrna_mx_pf_t * exp_matrices
  The PF DP matrices.
- vrna_param_t * params
  The precomputed free energy contributions for each type of loop.
- vrna_exp_param_t * exp_params
  The precomputed free energy contributions as Boltzmann factors.
- int * iindx
  DP matrix accessor.
- int * jindx
  DP matrix accessor.

User-defined data fields

- vrna_callback_recursion_status * stat_cb
  Recursion status callback (usually called just before, and after recursive computations in the library.
- void * auxdata
  A pointer to auxiliary, user-defined data.
- vrna_callback_free_auxdata * free_auxdata
  A callback to free auxiliary user data whenever the fold_compound itself is free'd.

Data fields available for single/hybrid structure prediction

- char * sequence
  The input sequence string.
- short * sequence_encoding
  Numerical encoding of the input sequence.
- short * **sequence_encoding2**
- char * ptype
  Pair type array.
- char * ptype_pf_compat
  ptype array indexed via iindx
- vrna_sc_t * sc
  The soft constraints for usage in structure prediction and evaluation.

Data fields for consensus structure prediction

- char ** sequences
  The aligned sequences.
- unsigned int n_seq
  The number of sequences in the alignment.
- char * cons_seq
  The consensus sequence of the aligned sequences.
- short * S_cons
  Numerical encoding of the consensus sequence.
- short ** S
  Numerical encoding of the sequences in the alignment.
- short ** S5
  S5[s][i] holds next base 5' of i in sequence s.
- short ** S3
  Sl[s][i] holds next base 3' of i in sequence s.
- char ** **Ss**
- unsigned short ** **a2s**
- int * pscore
  Precomputed array of pair types expressed as pairing scores.
- short * pscore_pf_compat

Precomputed array of pair types expressed as pairing scores indexed via iindx.

- vrna_sc_t ** scs

  A set of soft constraints (for each sequence in the alignment)

- int **oldAliEn**

Additional data fields for Distance Class Partitioning

These data fields are typically populated with meaningful data only if used in the context of Distance Class Partitioning

- unsigned int maxD1

  Maximum allowed base pair distance to first reference.

- unsigned int maxD2

  Maximum allowed base pair distance to second reference.

- short * reference_pt1

  A pairtable of the first reference structure.

- short * reference_pt2

  A pairtable of the second reference structure.

- unsigned int * referenceBPs1

  Matrix containing number of basepairs of reference structure1 in interval [i,j].

- unsigned int * referenceBPs2

  Matrix containing number of basepairs of reference structure2 in interval [i,j].

- unsigned int * bpdist

  Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].

- unsigned int * mm1

  Maximum matching matrix, reference struct 1 disallowed.

- unsigned int * mm2

  Maximum matching matrix, reference struct 2 disallowed.

Additional data fields for local folding

These data fields are typically populated with meaningful data only if used in the context of local folding

- int window_size

  window size for local folding sliding window approach

- char ** ptype_local

  Pair type array (for local folding)

13.45.2.1.1 Field Documentation

13.45.2.1.1.1 vrna_fc_type_e vrna_fc_s::type

The type of the vrna_fold_compound_t.

Currently possible values are VRNA_VC_TYPE_SINGLE, and VRNA_VC_TYPE_ALIGNMENT

Warning

Do not edit this attribute, it will be automagically set by the corresponding get() methods for the vrna_fold_compound_t. The value specified in this attribute dictates the set of other attributes to use within this data structure.

13.45.2.1.1.2 vrna_callback_recursion_status* vrna_fc_s::stat_cb

Recursion status callback (usually called just before, and after recursive computations in the library.

See also

vrna_callback_recursion_status(), vrna_fold_compound_add_callback()

13.45.2.1.1.3 void* vrna_fc_s::auxdata

A pointer to auxiliary, user-defined data.

See also

vrna_fold_compound_add_auxdata(), vrna_fold_compound_t.free_auxdata

13.45.2.1.1.4 vrna_callback_free_auxdata* vrna_fc_s::free_auxdata

A callback to free auxiliary user data whenever the fold_compound itself is free'd.

See also

vrna_fold_compound_t.auxdata, vrna_callback_free_auxdata()

13.45.2.1.1.5 char* vrna_fc_s::sequence

The input sequence string.

Warning

Only available if

`type==VRNA_VC_TYPE_SINGLE`

13.45.2.1.1.6 short* vrna_fc_s::sequence_encoding

Numerical encoding of the input sequence.

See also

vrna_sequence_encode()

Warning

Only available if

`type==VRNA_VC_TYPE_SINGLE`

13.45.2.1.1.7 char* vrna_fc_s::ptype

Pair type array.

Contains the numerical encoding of the pair type for each pair (i,j) used in MFE, Partition function and Evaluation computations.

Note

This array is always indexed via jindx, in contrast to previously different indexing between mfe and pf variants!

Warning

Only available if

`type==VRNA_VC_TYPE_SINGLE`

See also

vrna_idx_col_wise(), vrna_ptypes()

13.45.2.1.1.8 char* vrna_fc_s::ptype_pf_compat

ptype array indexed via iindx

Deprecated This attribute will vanish in the future! It's meant for backward compatibility only!

Warning

Only available if

`type==VRNA_VC_TYPE_SINGLE`

13.45.2.1.1.9 vrna_sc_t* vrna_fc_s::sc

The soft constraints for usage in structure prediction and evaluation.

Warning

Only available if

`type==VRNA_VC_TYPE_SINGLE`

13.45.2.1.1.10 char** vrna_fc_s::sequences

The aligned sequences.

Note

The end of the alignment is indicated by a NULL pointer in the second dimension

Warning

Only available if

`type==VRNA_VC_TYPE_ALIGNMENT`

13.45.2.1.1.11 unsigned int vrna_fc_s::n_seq

The number of sequences in the alignment.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.12 char* vrna_fc_s::cons_seq

The consensus sequence of the aligned sequences.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.13 short* vrna_fc_s::S_cons

Numerical encoding of the consensus sequence.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.14 short** vrna_fc_s::S

Numerical encoding of the sequences in the alignment.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.15 short** vrna_fc_s::S5

S5[s][i] holds next base 5' of i in sequence s.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.16 short** vrna_fc_s::S3

Sl[s][i] holds next base 3' of i in sequence s.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.17 int* vrna_fc_s::pscore

Precomputed array of pair types expressed as pairing scores.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.18 short* vrna_fc_s::pscore_pf_compat

Precomputed array of pair types expressed as pairing scores indexed via iindx.

Deprecated This attribute will vanish in the future!

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.2.1.1.19 vrna_sc_t** vrna_fc_s::scs

A set of soft constraints (for each sequence in the alignment)

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

13.45.3 Macro Definition Documentation

13.45.3.1 #define VRNA_STATUS_MFE_PRE (unsigned char)1

```
#include <ViennaRNA/data_structures.h>
```

Status message indicating that MFE computations are about to begin.

See also

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_mfe(), vrna_fold(), vrna_circfold(), vrna_alifold(), vrna_circalifold(), vrna_cofold()

13.45.3.2 #define VRNA_STATUS_MFE_POST (unsigned char)2

```
#include <ViennaRNA/data_structures.h>
```

Status message indicating that MFE computations are finished.

See also

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_mfe(), vrna_fold(), vrna_circfold(), vrna_alifold(), vrna_circalifold(), vrna_cofold()

13.45.3.3 #define VRNA_STATUS_PF_PRE (unsigned char)3

```
#include <ViennaRNA/data_structures.h>
```

Status message indicating that Partition function computations are about to begin.

See also

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_pf()

13.45.3.4 #define VRNA_STATUS_PF_POST (unsigned char)4

```
#include <ViennaRNA/data_structures.h>
```

Status message indicating that Partition function computations are finished.

See also

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_pf()

13.45.3.5 #define VRNA_OPTION_MFE 1U

```
#include <ViennaRNA/data_structures.h>
```

Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.

See also

vrna_fold_compound(), vrna_fold_compound_comparative(), VRNA_OPTION_EVAL_ONLY

13.45.3.6 #define VRNA_OPTION_PF 2U

```
#include <ViennaRNA/data_structures.h>
```

Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.

See also

vrna_fold_compound(), vrna_fold_compound_comparative(), VRNA_OPTION_EVAL_ONLY

13.45.3.7 #define VRNA_OPTION_EVAL_ONLY 8U

```
#include <ViennaRNA/data_structures.h>
```

Option flag to specify that neither MFE, nor PF DP matrices are required.

Use this flag in conjuntion with VRNA_OPTION_MFE, and VRNA_OPTION_PF to save memory for a vrna_fold↩_compound_t obtained from vrna_fold_compound(), or vrna_fold_compound_comparative() in cases where only energy evaluation but no structure prediction is required.

See also

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_eval_structure()

13.45.4 Typedef Documentation

13.45.4.1 typedef void(vrna_callback_free_auxdata) (void *data)

```
#include <ViennaRNA/data_structures.h>
```

Callback to free memory allocated for auxiliary user-provided data.

This type of user-implemented function usually deletes auxiliary data structures. The user must take care to free all the memory occupied by the data structure passed.

Parameters

data	The data that needs to be free'd

13.45.4.2 typedef void(vrna_callback_recursion_status) (unsigned char status, void *data)

```
#include <ViennaRNA/data_structures.h>
```

Callback to perform specific user-defined actions before, or after recursive computations.

See also

VRNA_STATUS_MFE_PRE, VRNA_STATUS_MFE_POST, VRNA_STATUS_PF_PRE, VRNA_STATUS_↩PF_POST

Parameters

status	The status indicator
data	The data structure that was assigned with vrna_fold_compound_add_auxdata()
status	The status indicator

13.45.5 Enumeration Type Documentation

13.45.5.1 enum vrna_fc_type_e

```
#include <ViennaRNA/data_structures.h>
```

An enumerator that is used to specify the type of a vrna_fold_compound_t.

Enumerator

VRNA_VC_TYPE_SINGLE Type is suitable for single, and hybridizing sequences

VRNA_VC_TYPE_ALIGNMENT Type is suitable for sequence alignments (consensus structure prediction)

13.45.6 Function Documentation

13.45.6.1 vrna_fold_compound_t* vrna_fold_compound (const char * *sequence*, vrna_md_t * *md_p*, unsigned int *options*)

```
#include <ViennaRNA/data_structures.h>
```

Retrieve a vrna_fold_compound_t data structure for single sequences and hybridizing sequences.

This function provides an easy interface to obtain a prefilled vrna_fold_compound_t by passing a single sequence, or two contatenated sequences as input. For the latter, sequences need to be seperated by an '&' character like this:

```
char *sequence = "GGGG&CCCC";
```

The optional parameter `md_p` can be used to specify the model details for successive computations based on the content of the generated vrna_fold_compound_t. Passing NULL will instruct the function to use default model details. The third parameter `options` may be used to specify dynamic programming (DP) matrix requirements. Use the macros:

- VRNA_OPTION_MFE
- VRNA_OPTION_PF
- VRNA_OPTION_WINDOW
- VRNA_OPTION_EVAL_ONLY
- VRNA_OPTION_DEFAULT

to specify the required type of computations that will be performed with the vrna_fold_compound_t.

If you just need the folding compound serving as a container for your data, you can simply pass VRNA_OPTIO↩N_DEFAULT to the `option` parameter. This creates a vrna_fold_compound_t without DP matrices, thus saving memory. Subsequent calls of any structure prediction function will then take care of allocating the memory required for the DP matrices. If you only intend to evaluate structures instead of actually predicting them, you may use the VRNA_OPTION_EVAL_ONLY macro. This will seriously speedup the creation of the vrna_fold_compound_t.

Note

The sequence string must be uppercase, and should contain only RNA (resp. DNA) alphabet depending on what energy parameter set is used

See also

vrna_fold_compound_free(), vrna_fold_compound_comparative(), vrna_md_t, VRNA_OPTION_MFE, VRN↩A_OPTION_PF, VRNA_OPTION_EVAL_ONLY, VRNA_OPTION_WINDOW

Parameters

sequence	A single sequence, or two concatenated sequences seperated by an '&' character
md_p	An optional set of model details
options	The options for DP matrices memory allocation

Returns

A prefilled vrna_fold_compound_t that can be readily used for computations

13.45.6.2 vrna_fold_compound_t* vrna_fold_compound_comparative (const char ** *sequences*, vrna_md_t * *md_p*, unsigned int *options*)

```
#include <ViennaRNA/data_structures.h>
```

Retrieve a vrna_fold_compound_t data structure for sequence alignments.

This function provides an easy interface to obtain a prefilled vrna_fold_compound_t by passing an alignment of sequences.

The optional parameter `md_p` can be used to specify the model details for successive computations based on the content of the generated vrna_fold_compound_t. Passing NULL will instruct the function to use default model details. The third parameter `options` may be used to specify dynamic programming (DP) matrix requirements. Use the macros:

- VRNA_OPTION_MFE
- VRNA_OPTION_PF
- VRNA_OPTION_EVAL_ONLY
- VRNA_OPTION_DEFAULT

to specify the required type of computations that will be performed with the vrna_fold_compound_t.

If you just need the folding compound serving as a container for your data, you can simply pass VRNA_OPTIO↩N_DEFAULT to the `option` parameter. This creates a vrna_fold_compound_t without DP matrices, thus saving memory. Subsequent calls of any structure prediction function will then take care of allocating the memory required for the DP matrices. If you only intend to evaluate structures instead of actually predicting them, you may use the VRNA_OPTION_EVAL_ONLY macro. This will seriously speedup the creation of the vrna_fold_compound_t.

Note

The sequence strings must be uppercase, and should contain only RNA (resp. DNA) alphabet including gap characters depending on what energy parameter set is used.

See also

vrna_fold_compound_free(), vrna_fold_compound(), vrna_md_t, VRNA_OPTION_MFE, VRNA_OPTION_↩PF, VRNA_OPTION_EVAL_ONLY, read_clustal()

Parameters

sequences	A sequence alignment including 'gap' characters
md_p	An optional set of model details
options	The options for DP matrices memory allocation

Returns

A prefilled vrna_fold_compound_t that can be readily used for computations

13.45.6.3 void vrna_fold_compound_free (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/data_structures.h>
```

Free memory occupied by a vrna_fold_compound_t.

See also

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_mx_mfe_free(), vrna_mx_pf_free()

Parameters

vc	The vrna_fold_compound_t that is to be erased from memory

13.45.6.4 void vrna_fold_compound_add_auxdata (vrna_fold_compound_t * *vc*, void * *data*, vrna_callback_free_auxdata * *f*)

```
#include <ViennaRNA/data_structures.h>
```

Add auxiliary data to the vrna_fold_compound_t.

This function allows to bind arbitrary data to a vrna_fold_compound_t which may later on be used by one of the callback functions, e.g. vrna_callback_recursion_status(). To allow for proper cleanup of the memory occupied by this auxiliary data, the user may also provide a pointer to a cleanup function that free's the corresponding memory. This function will be called automatically when the vrna_fold_compound_t is free'd with vrna_fold_compound_free().

Note

Before attaching the arbitrary data pointer, this function will call the vrna_callback_free_auxdata() on any pre-existing data that is already attached.

See also

vrna_callback_free_auxdata()

Parameters

vc	The fold_compound the arbitrary data pointer should be associated with
~~*data*~~	~~A pointer to an arbitrary data structure~~
f	A pointer to function that free's memory occupied by the arbitrary data (May be NULL)

13.45.6.5 void vrna_fold_compound_add_callback (vrna_fold_compound_t * *vc*, vrna_callback_recursion_status * *f*)

```
#include <ViennaRNA/data_structures.h>
```

Add a recursion status callback to the vrna_fold_compound_t.

Binding a recursion status callback function to a vrna_fold_compound_t allows to perform arbitrary operations just before, or after an actual recursive computations, e.g. MFE prediction, is performed by the RNAlib. The callback function will be provided with a pointer to its vrna_fold_compound_t, and a status message. Hence, it has complete access to all variables that incluence the recursive computations.

See also

vrna_callback_recursion_status(), vrna_fold_compound_t, VRNA_STATUS_MFE_PRE, VRNA_STATUS_MFE_POST, VRNA_STATUS_PF_PRE, VRNA_STATUS_PF_POST

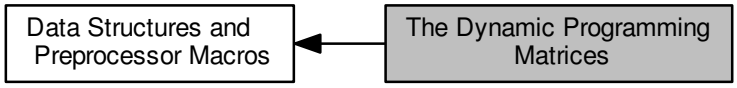
Parameters

vc	The fold_compound the callback function should be attached to
f	The pointer to the recursion status callback function

13.46 The Dynamic Programming Matrices

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

Collaboration diagram for The Dynamic Programming Matrices:



Data Structures

- struct vrna_mx_mfe_s

  Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...
- struct vrna_mx_pf_s

  Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...

Typedefs

- typedef struct vrna_mx_mfe_s vrna_mx_mfe_t

  Typename for the Minimum Free Energy (MFE) DP matrices data structure vrna_mx_mfe_s.
- typedef struct vrna_mx_pf_s vrna_mx_pf_t

  Typename for the Partition Function (PF) DP matrices data structure vrna_mx_pf_s.

Enumerations

Functions

- int vrna_mx_add (vrna_fold_compound_t *vc, vrna_mx_type_e type, unsigned int options)

  Add Dynamic Programming (DP) matrices (allocate memory)
- void vrna_mx_mfe_free (vrna_fold_compound_t *vc)

  Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.
- void vrna_mx_pf_free (vrna_fold_compound_t *vc)

  Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

13.46.1 Detailed Description

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

13.46.2 Data Structure Documentation

13.46.2.1 struct vrna_mx_mfe_s

Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t.

Data Fields

Common fields for MFE matrices

- vrna_mx_type_e **type**
- unsigned int length
  Length of the sequence, therefore an indicator of the size of the DP matrices.

Default DP matrices

Note

These data fields are available if
`vrna_mx_mfe_t.type == VRNA_MX_DEFAULT`

- int * c
  Energy array, given that i-j pair.
- int * f5
  Energy of 5' end.
- int * f3
  Energy of 3' end.
- int * fc
  Energy from i to cutpoint (and vice versa if i>cut)
- int * fML
  Multi-loop auxiliary energy array.
- int * fM1
  Second ML array, only for unique multibrnach loop decomposition.
- int * fM2
  Energy for a multibranch loop region with exactly two stems, extending to 3' end.
- int * ggg
  Energies of g-quadruplexes.
- int Fc
  Minimum Free Energy of entire circular RNA.
- int **FcH**
- int **FcI**
- int **FcM**

Local Folding DP matrices using window approach

Note

These data fields are available if
`vrna_mx_mfe_t.type == VRNA_MX_WINDOW`

- int ** c_local
  Energy array, given that i-j pair.
- int * f3_local
  Energy of 5' end.
- int ** fML_local
  Multi-loop auxiliary energy array.
- int ** ggg_local
  Energies of g-quadruplexes.

Distance Class DP matrices

Note

These data fields are available if

`vrna_mx_mfe_t.type == VRNA_MX_2DFOLD`

- int *** **E_F5**
- int ** **l_min_F5**
- int ** **l_max_F5**
- int * **k_min_F5**
- int * **k_max_F5**
- int *** **E_F3**
- int ** **l_min_F3**
- int ** **l_max_F3**
- int * **k_min_F3**
- int * **k_max_F3**
- int *** **E_C**
- int ** **l_min_C**
- int ** **l_max_C**
- int * **k_min_C**
- int * **k_max_C**
- int *** **E_M**
- int ** **l_min_M**
- int ** **l_max_M**
- int * **k_min_M**
- int * **k_max_M**
- int *** **E_M1**
- int ** **l_min_M1**
- int ** **l_max_M1**
- int * **k_min_M1**
- int * **k_max_M1**
- int *** **E_M2**
- int ** **l_min_M2**
- int ** **l_max_M2**
- int * **k_min_M2**
- int * **k_max_M2**
- int ** **E_Fc**
- int * **l_min_Fc**
- int * **l_max_Fc**
- int **k_min_Fc**
- int **k_max_Fc**
- int ** **E_FcH**
- int * **l_min_FcH**
- int * **l_max_FcH**
- int **k_min_FcH**
- int **k_max_FcH**
- int ** **E_FcI**
- int * **l_min_FcI**
- int * **l_max_FcI**
- int **k_min_FcI**
- int **k_max_FcI**
- int ** **E_FcM**
- int * **l_min_FcM**
- int * **l_max_FcM**
- int **k_min_FcM**
- int **k_max_FcM**
- int * **E_F5_rem**
- int * **E_F3_rem**
- int * **E_C_rem**
- int * **E_M_rem**
- int * **E_M1_rem**
- int * **E_M2_rem**
- int **E_Fc_rem**
- int **E_FcH_rem**
- int **E_FcI_rem**
- int **E_FcM_rem**

13.46.2.2 struct vrna_mx_pf_s

Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t.

Data Fields

Common fields for DP matrices

- vrna_mx_type_e **type**
- unsigned int **length**
- FLT_OR_DBL * **scale**
- FLT_OR_DBL * **expMLbase**

Default PF matrices

Note

These data fields are available if

`vrna_mx_pf_t.type == VRNA_MX_DEFAULT`

- FLT_OR_DBL * **q**
- FLT_OR_DBL * **qb**
- FLT_OR_DBL * **qm**
- FLT_OR_DBL * **qm1**
- FLT_OR_DBL * **probs**
- FLT_OR_DBL * **q1k**
- FLT_OR_DBL * **qln**
- FLT_OR_DBL * **G**
- FLT_OR_DBL **qo**
- FLT_OR_DBL * **qm2**
- FLT_OR_DBL **qho**
- FLT_OR_DBL **qio**
- FLT_OR_DBL **qmo**

Distance Class DP matrices

Note

These data fields are available if

`vrna_mx_pf_t.type == VRNA_MX_2DFOLD`

- FLT_OR_DBL *** **Q**
- int ** **l_min_Q**
- int ** **l_max_Q**
- int * **k_min_Q**
- int * **k_max_Q**
- FLT_OR_DBL *** **Q_B**
- int ** **l_min_Q_B**
- int ** **l_max_Q_B**
- int * **k_min_Q_B**
- int * **k_max_Q_B**
- FLT_OR_DBL *** **Q_M**
- int ** **l_min_Q_M**
- int ** **l_max_Q_M**
- int * **k_min_Q_M**
- int * **k_max_Q_M**
- FLT_OR_DBL *** **Q_M1**
- int ** **l_min_Q_M1**
- int ** **l_max_Q_M1**
- int * **k_min_Q_M1**

- int * **k_max_Q_M1**
- FLT_OR_DBL *** **Q_M2**
- int ** **l_min_Q_M2**
- int ** **l_max_Q_M2**
- int * **k_min_Q_M2**
- int * **k_max_Q_M2**
- FLT_OR_DBL ** **Q_c**
- int * **l_min_Q_c**
- int * **l_max_Q_c**
- int **k_min_Q_c**
- int **k_max_Q_c**
- FLT_OR_DBL ** **Q_cH**
- int * **l_min_Q_cH**
- int * **l_max_Q_cH**
- int **k_min_Q_cH**
- int **k_max_Q_cH**
- FLT_OR_DBL ** **Q_cI**
- int * **l_min_Q_cI**
- int * **l_max_Q_cI**
- int **k_min_Q_cI**
- int **k_max_Q_cI**
- FLT_OR_DBL ** **Q_cM**
- int * **l_min_Q_cM**
- int * **l_max_Q_cM**
- int **k_min_Q_cM**
- int **k_max_Q_cM**
- FLT_OR_DBL * **Q_rem**
- FLT_OR_DBL * **Q_B_rem**
- FLT_OR_DBL * **Q_M_rem**
- FLT_OR_DBL * **Q_M1_rem**
- FLT_OR_DBL * **Q_M2_rem**
- FLT_OR_DBL **Q_c_rem**
- FLT_OR_DBL **Q_cH_rem**
- FLT_OR_DBL **Q_cI_rem**
- FLT_OR_DBL **Q_cM_rem**

13.46.3 Enumeration Type Documentation

13.46.3.1 enum vrna_mx_type_e

```
#include <ViennaRNA/dp_matrices.h>
```

An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.

See also

vrna_mx_mfe_t, vrna_mx_pf_t

Enumerator

VRNA_MX_DEFAULT Default DP matrices.

VRNA_MX_WINDOW DP matrices suitable for local structure prediction using window approach.

See also

vrna_mfe_window(), vrna_mfe_window_zscore(), pfl_fold()

VRNA_MX_2DFOLD DP matrices suitable for distance class partitioned structure prediction.

See also

vrna_mfe_TwoD(), vrna_pf_TwoD()

13.46.4 Function Documentation

13.46.4.1 int vrna_mx_add (vrna_fold_compound_t * *vc*, vrna_mx_type_e *type*, unsigned int *options*)

```
#include <ViennaRNA/dp_matrices.h>
```

Add Dynamic Programming (DP) matrices (allocate memory)

This function adds DP matrices of a specific type to the provided vrna_fold_compound_t, such that successive DP recursion can be applied. The function caller has to specify which type of DP matrix is requested, see vrna_mx_type_e, and what kind of recursive algorithm will be applied later on, using the parameters type, and options, respectively. For the latter, Minimum free energy (MFE), and Partition function (PF) computations are distinguished. A third option that may be passed is VRNA_OPTION_HYBRID, indicating that auxiliary DP arrays are required for RNA-RNA interaction prediction.

Note

Usually, there is no need to call this function, since the constructors of vrna_fold_compound_t are handling all the DP matrix memory allocation.

See also

vrna_mx_mfe_add(), vrna_mx_pf_add(), vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), vrna_mx_pf_free(), vrna_mx_mfe_free(), vrna_mx_type_e, VRNA_OPTION_MFE, VRNA_OPTION_PF, VRNA_OPTION_HYBRID, VRNA_OPTION_EVAL_ONLY

Parameters

vc	The vrna_fold_compound_t that holds pointers to the DP matrices
type	The type of DP matrices requested
options	Option flags that specify the kind of DP matrices, such as MFE or PF arrays, and auxiliary requirements

Returns

1 if DP matrices were properly allocated and attached, 0 otherwise

13.46.4.2 void vrna_mx_mfe_free (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/dp_matrices.h>
```

Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.

See also

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), vrna_mx_pf_free()

Parameters

vc	The vrna_fold_compound_t storing the MFE DP matrices that are to be erased from memory

13.46.4.3 void vrna_mx_pf_free (vrna_fold_compound_t * *vc*)

```
#include <ViennaRNA/dp_matrices.h>
```

Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

See also

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), vrna_mx_mfe_free()

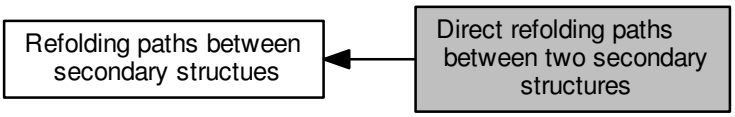
Parameters

vc	The vrna_fold_compound_t storing the PF DP matrices that are to be erased from memory

13.47 Direct refolding paths between two secondary structures

Implementation of heuristics to explore optimal (re-)folding paths between two secondary structures.

Collaboration diagram for Direct refolding paths between two secondary structures:



Data Structures

- struct vrna_path_s

  An element of a refolding path list. More...

Typedefs

- typedef struct vrna_path_s vrna_path_t

  Typename for the refolding path data structure vrna_path_s.
- typedef struct vrna_path_s path_t

  Old typename of vrna_path_s.

Functions

- int vrna_path_findpath_saddle (vrna_fold_compound_t *vc, const char *struc1, const char *struc2, int max)

  Find energy of a saddle point between 2 structures (search only direct path)
- vrna_path_t * vrna_path_findpath (vrna_fold_compound_t *vc, const char *s1, const char *s2, int maxkeep)

  Find refolding path between 2 structures (search only direct path)
- int find_saddle (const char *seq, const char *struc1, const char *struc2, int max)

  Find energy of a saddle point between 2 structures (search only direct path)
- void free_path (vrna_path_t *path)

  Free memory allocated by get_path() function.
- vrna_path_t * get_path (const char *seq, const char *s1, const char *s2, int maxkeep)

  Find refolding path between 2 structures (search only direct path)

13.47.1 Detailed Description

Implementation of heuristics to explore optimal (re-)folding paths between two secondary structures.

13.47.2 Data Structure Documentation

13.47.2.1 struct vrna_path_s

An element of a refolding path list.

See also

vrna_path_findpath()

Data Fields

- double en

  Free energy of current structure.
- char * s

  Secondary structure in dot-bracket notation.

13.47.3 Typedef Documentation

13.47.3.1 typedef struct vrna_path_s path_t

```
#include <ViennaRNA/findpath.h>
```

Old typename of vrna_path_s.

Deprecated Use vrna_path_t instead!

13.47.4 Function Documentation

13.47.4.1 int vrna_path_findpath_saddle (vrna_fold_compound_t * *vc*, const char * *struc1*, const char * *struc2*, int *max*)

```
#include <ViennaRNA/findpath.h>
```

Find energy of a saddle point between 2 structures (search only direct path)

This function uses an inplementation of the *findpath* algorithm [5] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

See also

vrna_fold_compound(), vrna_fold_compound_t, vrna_path_findpath()

Parameters

vc	The vrna_fold_compound_t with precomputed sequence encoding and model details
struc1	The start structure in dot-brakcet notation
struc2	The target structure in dot-bracket notation
max	A number specifying how many strutures are being kept at each step during the search

Returns

The saddle energy in 10cal/mol

13.47.4.2 vrna_path_t* vrna_path_findpath (vrna_fold_compound_t * *vc*, const char * *s1*, const char * *s2*, int *maxkeep*)

```
#include <ViennaRNA/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

This function uses an inplementation of the *findpath* algorithm [5] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

See also

vrna_fold_compound(), vrna_fold_compound_t, vrna_path_findpath_saddle()

Parameters

vc	The vrna_fold_compound_t with precomputed sequence encoding and model details
s1	The start structure in dot-brakcet notation
s2	The target structure in dot-bracket notation
maxkeep	A number specifying how many strutures are being kept at each step during the search

Returns

The saddle energy in 10cal/mol

13.47.4.3 int find_saddle (const char * *seq*, const char * *struc1*, const char * *struc2*, int *max*)

```
#include <ViennaRNA/findpath.h>
```

Find energy of a saddle point between 2 structures (search only direct path)

Parameters

seq	RNA sequence
struc1	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
struc2	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
max	integer how many strutures are being kept during the search

Returns

the saddle energy in 10cal/mol

13.47.4.4 void free_path (vrna_path_t * *path*)

```
#include <ViennaRNA/findpath.h>
```

Free memory allocated by get_path() function.

Parameters

path	pointer to memory to be freed

13.47.4.5 vrna_path_t* get_path (const char * *seq*, const char * *s1*, const char * *s2*, int *maxkeep*)

```
#include <ViennaRNA/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

Parameters

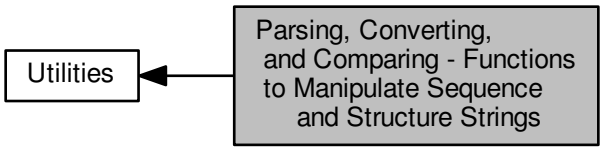
seq	RNA sequence
s1	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
s2	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
maxkeep	integer how many strutures are being kept during the search

Returns

direct refolding path between two structures

13.48 Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings

Collaboration diagram for Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings:



Files

- file string_utils.h

  General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

Macros

- #define XSTR(s) STR(s)

  Stringify a macro after expansion.
- #define STR(s) #s

  Stringify a macro argument.
- #define FILENAME_MAX_LENGTH 80

  Maximum length of filenames that are generated by our programs.
- #define FILENAME_ID_LENGTH 42

  Maximum length of id taken from fasta header for filename generation.

Functions

- char * vrna_random_string (int l, const char symbols[])

  Create a random string using characters from a specified symbol set.
- int vrna_hamming_distance (const char *s1, const char *s2)

  Calculate hamming distance between two sequences.
- int vrna_hamming_distance_bound (const char *s1, const char *s2, int n)

  Calculate hamming distance between two sequences up to a specified length.
- void vrna_seq_toRNA (char *sequence)

  Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.
- void vrna_seq_toupper (char *sequence)

  Convert an input sequence to uppercase.
- char * vrna_cut_point_insert (const char *string, int cp)

  Add a separating '&' character into a string according to cut-point position.
- char * vrna_cut_point_remove (const char *string, int *cp)

  Remove a separating '&' character from a string.

13.48.1 Detailed Description

13.48.2 Macro Definition Documentation

13.48.2.1 #define FILENAME_MAX_LENGTH 80

```
#include <ViennaRNA/string_utils.h>
```

Maximum length of filenames that are generated by our programs.

This definition should be used throughout the complete ViennaRNA package wherever a static array holding filenames of output files is declared.

13.48.2.2 #define FILENAME_ID_LENGTH 42

```
#include <ViennaRNA/string_utils.h>
```

Maximum length of id taken from fasta header for filename generation.

this has to be smaller than FILENAME_MAX_LENGTH since in most cases, some suffix will be appended to the ID

13.48.3 Function Documentation

13.48.3.1 char* vrna_random_string (int *l*, const char *symbols[]*)

```
#include <ViennaRNA/string_utils.h>
```

Create a random string using characters from a specified symbol set.

Parameters

l	The length of the sequence
symbols	The symbol set

Returns

A random string of length 'l' containing characters from the symbolset

13.48.3.2 int vrna_hamming_distance (const char * *s1*, const char * *s2*)

```
#include <ViennaRNA/string_utils.h>
```

Calculate hamming distance between two sequences.

Parameters

s1	The first sequence
s2	The second sequence

Returns

The hamming distance between s1 and s2

13.48.3.3 int vrna_hamming_distance_bound (const char * *s1*, const char * *s2*, int *n*)

```
#include <ViennaRNA/string_utils.h>
```

Calculate hamming distance between two sequences up to a specified length.

This function is similar to vrna_hamming_distance() but instead of comparing both sequences up to their actual length only the first 'n' characters are taken into account

Parameters

s1	The first sequence
s2	The second sequence
n	The length of the subsequences to consider (starting from the 5' end)

Returns

The hamming distance between s1 and s2

13.48.3.4 void vrna_seq_toRNA (char * *sequence*)

```
#include <ViennaRNA/string_utils.h>
```

Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.

This function substitudes *T* and *t* with *U* and *u*, respectively

Parameters

sequence	The sequence to be converted

13.48.3.5 void vrna_seq_toupper (char * *sequence*)

```
#include <ViennaRNA/string_utils.h>
```

Convert an input sequence to uppercase.

Parameters

sequence	The sequence to be converted

13.48.3.6 char* vrna_cut_point_insert (const char * *string*, int *cp*)

```
#include <ViennaRNA/string_utils.h>
```

Add a separating '&' character into a string according to cut-point position.

If the cut-point position is less or equal to zero, this function just returns a copy of the provided string. Otherwise, the cut-point character is set at the corresponding position

Parameters

string	The original string
cp	The cut-point position

Returns

A copy of the provided string including the cut-point character

13.48.3.7 char* vrna_cut_point_remove (const char * *string*, int * *cp*)

```
#include <ViennaRNA/string_utils.h>
```

Remove a separating '&' character from a string.

This function removes the cut-point indicating '&' character from a string and memorizes its position in a provided integer variable. If not '&' is found in the input, the integer variable is set to -1. The function returns a copy of the input string with the '&' being sliced out.

Parameters

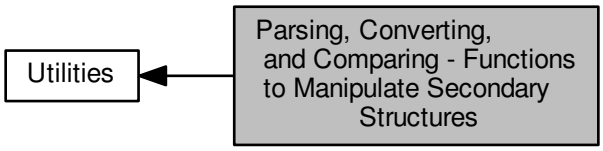
string	The original string
cp	The cut-point position

Returns

A copy of the input string with the '&' being sliced out

13.49 Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures

Collaboration diagram for Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures:



Files

- file RNAstruct.h

  Parsing and Coarse Graining of Structures.
- file structure_utils.h

  Various utility- and helper-functions for secondary structure parsing, converting, etc.

Data Structures

- struct vrna_hx_s

  Data structure representing an entry of a helix list. More...

Typedefs

- typedef struct vrna_hx_s vrna_hx_t

  Convenience typedef for data structure vrna_hx_s.

Functions

- char * b2HIT (const char *structure)

  Converts the full structure from bracket notation to the HIT notation including root.
- char * b2C (const char *structure)

  Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.
- char * b2Shapiro (const char *structure)

  Converts the full structure from bracket notation to the weighted coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.
- char * add_root (const char *structure)

  Adds a root to an un-rooted tree in any except bracket notation.
- char * expand_Shapiro (const char *coarse)

  Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

- char * expand_Full (const char *structure)

  Convert the full structure from bracket notation to the expanded notation including root.
- char * unexpand_Full (const char *ffull)

  Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.
- char * unweight (const char *wcoarse)

  Strip weights from any weighted tree.
- void unexpand_aligned_F (char *align[2])

  Converts two aligned structures in expanded notation.
- void parse_structure (const char *structure)

  Collects a statistic of structure elements of the full structure in bracket notation.
- char * vrna_db_pack (const char *struc)

  Pack secondary secondary structure, 5:1 compression using base 3 encoding.
- char * vrna_db_unpack (const char *packed)

  Unpack secondary structure previously packed with vrna_db_pack()
- short * vrna_ptable (const char *structure)

  Create a pair table of a secondary structure.
- short * vrna_pt_pk_get (const char *structure)

  Create a pair table of a secondary structure (pseudo-knot version)
- short * vrna_ptable_copy (const short *pt)

  Get an exact copy of a pair table.
- short * vrna_pt_ali_get (const char *structure)

  Create a pair table of a secondary structure (snoop align version)
- short * vrna_pt_snoop_get (const char *structure)

  Create a pair table of a secondary structure (snoop version)
- int * vrna_loopidx_from_ptable (const short *pt)

  Get a loop index representation of a structure.
- char * vrna_db_from_ptable (short *pt)

  Convert a pair table into dot-parenthesis notation.
- int vrna_bp_distance (const char *str1, const char *str2)

  Compute the "base pair" distance between two secondary structures s1 and s2.
- unsigned int * vrna_refBPcnt_matrix (const short *reference_pt, unsigned int turn)

  Make a reference base pair count matrix.
- unsigned int * vrna_refBPdist_matrix (const short *pt1, const short *pt2, unsigned int turn)

  Make a reference base pair distance matrix.
- char * vrna_db_from_probs (const FLT_OR_DBL *pr, unsigned int length)

  Create a dot-bracket like structure string from base pair probability matrix.
- char vrna_bpp_symbol (const float *x)

  Get a pseudo dot bracket notation for a given probability information.
- char * vrna_db_from_bp_stack (vrna_bp_stack_t *bp, unsigned int length)

  Create a dot-backet/parenthesis structure from backtracking stack.
- vrna_plist_t * vrna_plist (const char *struc, float pr)

  Create a vrna_plist_t from a dot-bracket string.
- char * vrna_db_from_plist (vrna_plist_t *pairs, unsigned int n)

  Convert a list of base pairs into dot-bracket notation.
- void assign_plist_from_db (vrna_plist_t **pl, const char *struc, float pr)

  Create a vrna_plist_t from a dot-bracket string.
- char * pack_structure (const char *struc)

  Pack secondary secondary structure, 5:1 compression using base 3 encoding.
- char * unpack_structure (const char *packed)

  Unpack secondary structure previously packed with pack_structure()
- short * make_pair_table (const char *structure)

Create a pair table of a secondary structure.

- short * copy_pair_table (const short *pt)

  Get an exact copy of a pair table.
- short * alimake_pair_table (const char *structure)
- short * make_pair_table_snoop (const char *structure)
- int bp_distance (const char *str1, const char *str2)

  Compute the "base pair" distance between two secondary structures s1 and s2.
- unsigned int * make_referenceBP_array (short *reference_pt, unsigned int turn)

  Make a reference base pair count matrix.
- unsigned int * compute_BPdifferences (short *pt1, short *pt2, unsigned int turn)

  Make a reference base pair distance matrix.
- void parenthesis_structure (char *structure, vrna_bp_stack_t *bp, int length)

  Create a dot-backet/parenthesis structure from backtracking stack.
- void parenthesis_zuker (char *structure, vrna_bp_stack_t *bp, int length)

  Create a dot-backet/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.
- void bppm_to_structure (char *structure, FLT_OR_DBL *pr, unsigned int length)

  Create a dot-bracket like structure string from base pair probability matrix.
- char bppm_symbol (const float *x)

  Get a pseudo dot bracket notation for a given probability information.

Variables

- int loop_size [STRUC]

  contains a list of all loop sizes. loop_size[0] contains the number of external bases.
- int helix_size [STRUC]

  contains a list of all stack sizes.
- int loop_degree [STRUC]

  contains the corresponding list of loop degrees.
- int loops

  contains the number of loops (and therefore of stacks).
- int unpaired

  contains the number of unpaired bases.
- int pairs

  contains the number of base pairs in the last parsed structure.

13.49.1 Detailed Description

13.49.2 Data Structure Documentation

13.49.2.1 struct vrna_hx_s

Data structure representing an entry of a helix list.

13.49.3 Function Documentation

13.49.3.1 char* b2HIT (const char * *structure*)

```
#include <ViennaRNA/RNAstruct.h>
```

Converts the full structure from bracket notation to the HIT notation including root.

Parameters

structure	

Returns

13.49.3.2 char* b2C (const char * *structure*)

```
#include <ViennaRNA/RNAstruct.h>
```

Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

Parameters

structure	

Returns

13.49.3.3 char* b2Shapiro (const char * *structure*)

```
#include <ViennaRNA/RNAstruct.h>
```

Converts the full structure from bracket notation to the *weighted* coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.

Parameters

structure	

Returns

13.49.3.4 char* add_root (const char * *structure*)

```
#include <ViennaRNA/RNAstruct.h>
```

Adds a root to an un-rooted tree in any except bracket notation.

Parameters

structure	

Returns

13.49.3.5 char* expand_Shapiro (const char * *coarse*)

```
#include <ViennaRNA/RNAstruct.h>
```

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

Parameters

coarse	

Returns

13.49.3.6 char* expand_Full (const char * *structure*)

```
#include <ViennaRNA/RNAstruct.h>
```

Convert the full structure from bracket notation to the expanded notation including root.

Parameters

structure	

Returns

13.49.3.7 char* unexpand_Full (const char * *ffull*)

```
#include <ViennaRNA/RNAstruct.h>
```

Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

Parameters

ffull	

Returns

13.49.3.8 char* unweight (const char * *wcoarse*)

```
#include <ViennaRNA/RNAstruct.h>
```

Strip weights from any weighted tree.

Parameters

wcoarse	

Returns

13.49.3.9 void unexpand_aligned_F (char * *align[2]*)

```
#include <ViennaRNA/RNAstruct.h>
```

Converts two aligned structures in expanded notation.

Takes two aligned structures as produced by tree_edit_distance() function back to bracket notation with '_' as the gap character. The result overwrites the input.

Parameters

align	

13.49.3.10 void parse_structure (const char * *structure*)

```
#include <ViennaRNA/RNAstruct.h>
```

Collects a statistic of structure elements of the full structure in bracket notation.

The function writes to the following global variables: loop_size, loop_degree, helix_size, loops, pairs, unpaired

Parameters

structure	

Returns

13.49.3.11 char* vrna_db_pack (const char * *struc*)

```
#include <ViennaRNA/structure_utils.h>
```

Pack secondary secondary structure, 5:1 compression using base 3 encoding.

Returns a binary string encoding of the secondary structure using a 5:1 compression scheme. The string is NULL terminated and can therefore be used with standard string functions such as strcmp(). Useful for programs that need to keep many structures in memory.

See also

vrna_db_unpack()

Parameters

struc	The secondary structure in dot-bracket notation

Returns

The binary encoded structure

13.49.3.12 char* vrna_db_unpack (const char * *packed*)

```
#include <ViennaRNA/structure_utils.h>
```

Unpack secondary structure previously packed with vrna_db_pack()

Translate a compressed binary string produced by vrna_db_pack() back into the familiar dot-bracket notation.

See also

vrna_db_pack()

Parameters

packed	The binary encoded packed secondary structure

Returns

The unpacked secondary structure in dot-bracket notation

13.49.3.13 short* vrna_ptable (const char * *structure*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a pair table of a secondary structure.

Returns a newly allocated table, such that table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

Parameters

structure	The secondary structure in dot-bracket notation

Returns

A pointer to the created pair_table

13.49.3.14 short* vrna_pt_pk_get (const char * *structure*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a pair table of a secondary structure (pseudo-knot version)

Returns a newly allocated table, such that table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

In contrast to vrna_ptable() this function also recognizes the base pairs denoted by '[' and ']' brackets.

Parameters

structure	The secondary structure in (extended) dot-bracket notation

Returns

A pointer to the created pair_table

13.49.3.15 short* vrna_ptable_copy (const short * *pt*)

```
#include <ViennaRNA/structure_utils.h>
```

Get an exact copy of a pair table.

Parameters

pt	The pair table to be copied

Returns

A pointer to the copy of 'pt'

13.49.3.16 short* vrna_pt_snoop_get (const char * *structure*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a pair table of a secondary structure (snoop version)

returns a newly allocated table, such that: table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure. The special pseudoknotted H/ACA-mRNA structure is taken into account.

13.49.3.17 char* vrna_db_from_ptable (short * *pt*)

```
#include <ViennaRNA/structure_utils.h>
```

Convert a pair table into dot-parenthesis notation.

Parameters

pt	The pair table to be copied

Returns

A char pointer to the dot-bracket string

13.49.3.18 int vrna_bp_distance (const char * *str1*, const char * *str2*)

```
#include <ViennaRNA/structure_utils.h>
```

Compute the "base pair" distance between two secondary structures s1 and s2.

The sequences should have the same length. dist = number of base pairs in one structure but not in the other same as edit distance with open-pair close-pair as move-set

Parameters

str1	First structure in dot-bracket notation
str2	Second structure in dot-bracket notation

Returns

The base pair distance between str1 and str2

13.49.3.19 unsigned int* vrna_refBPcnt_matrix (const short * *reference_pt*, unsigned int *turn*)

```
#include <ViennaRNA/structure_utils.h>
```

Make a reference base pair count matrix.

Get an upper triangular matrix containing the number of basepairs of a reference structure for each interval [i,j] with i<j. Access it via iindx!!!

13.49.3.20 unsigned int* vrna_refBPdist_matrix (const short * *pt1*, const short * *pt2*, unsigned int *turn*)

```
#include <ViennaRNA/structure_utils.h>
```

Make a reference base pair distance matrix.

Get an upper triangular matrix containing the base pair distance of two reference structures for each interval [i,j] with i<j. Access it via iindx!!!

13.49.3.21 char* vrna_db_from_bp_stack (vrna_bp_stack_t * *bp*, unsigned int *length*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a dot-backet/parenthesis structure from backtracking stack.

This function is capable to create dot-bracket structures from suboptimal structure prediction sensu M. Zuker

Parameters

bp	Base pair stack containing the traced base pairs
length	The length of the structure

Returns

The secondary structure in dot-bracket notation as provided in the input

13.49.3.22 vrna_plist_t* vrna_plist (const char * *struc*, float *pr*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a vrna_plist_t from a dot-bracket string.

The dot-bracket string is parsed and for each base pair an entry in the plist is created. The probability of each pair in the list is set by a function parameter.

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

Parameters

struc	The secondary structure in dot-bracket notation
pr	The probability for each base pair used in the plist

Returns

The plist array

13.49.3.23 char* vrna_db_from_plist (vrna_plist_t * *pairs*, unsigned int *n*)

```
#include <ViennaRNA/structure_utils.h>
```

Convert a list of base pairs into dot-bracket notation.

See also

vrna_plist()

Parameters

pairs	A vrna_plist_t containing the pairs to be included in the dot-bracket string
n	The length of the structure (number of nucleotides)

Returns

The dot-bracket string containing the provided base pairs

13.49.3.24 void assign_plist_from_db (vrna_plist_t ** *pl*, const char * *struc*, float *pr*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a vrna_plist_t from a dot-bracket string.

The dot-bracket string is parsed and for each base pair an entry in the plist is created. The probability of each pair in the list is set by a function parameter.

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

Deprecated Use vrna_plist() instead

Parameters

pl	A pointer to the vrna_plist_t that is to be created
struc	The secondary structure in dot-bracket notation
pr	The probability for each base pair

13.49.3.25 char* pack_structure (const char * *struc*)

```
#include <ViennaRNA/structure_utils.h>
```

Pack secondary secondary structure, 5:1 compression using base 3 encoding.

Returns a binary string encoding of the secondary structure using a 5:1 compression scheme. The string is NULL terminated and can therefore be used with standard string functions such as strcmp(). Useful for programs that need to keep many structures in memory.

Deprecated Use vrna_db_pack() as a replacement

Parameters

struc	The secondary structure in dot-bracket notation

Returns

The binary encoded structure

13.49.3.26 char* unpack_structure (const char * *packed*)

```
#include <ViennaRNA/structure_utils.h>
```

Unpack secondary structure previously packed with pack_structure()

Translate a compressed binary string produced by pack_structure() back into the familiar dot-bracket notation.

Deprecated Use vrna_db_unpack() as a replacement

Parameters

packed	The binary encoded packed secondary structure

Returns

The unpacked secondary structure in dot-bracket notation

13.49.3.27 short* make_pair_table (const char * *structure*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a pair table of a secondary structure.

Returns a newly allocated table, such that table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

Deprecated Use vrna_ptable() instead

Parameters

structure	The secondary structure in dot-bracket notation

Returns

A pointer to the created pair_table

13.49.3.28 short* copy_pair_table (const short * *pt*)

```
#include <ViennaRNA/structure_utils.h>
```

Get an exact copy of a pair table.

Deprecated Use vrna_ptable_copy() instead

Parameters

pt	The pair table to be copied

Returns

A pointer to the copy of 'pt'

13.49.3.29 short* alimake_pair_table (const char * *structure*)

```
#include <ViennaRNA/structure_utils.h>
```

Pair table for snoop align

Deprecated Use vrna_pt_ali_get() instead!

13.49.3.30 short* make_pair_table_snoop (const char * *structure*)

```
#include <ViennaRNA/structure_utils.h>
```

returns a newly allocated table, such that: table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure. The special pseudoknotted H/ACA-mRNA structure is taken into account.

Deprecated Use vrna_pt_snoop_get() instead!

13.49.3.31 int bp_distance (const char * *str1*, const char * *str2*)

```
#include <ViennaRNA/structure_utils.h>
```

Compute the "base pair" distance between two secondary structures s1 and s2.

The sequences should have the same length. dist = number of base pairs in one structure but not in the other same as edit distance with open-pair close-pair as move-set

Deprecated Use vrna_bp_distance instead

Parameters

str1	First structure in dot-bracket notation
str2	Second structure in dot-bracket notation

Returns

The base pair distance between str1 and str2

13.49.3.32 unsigned int* make_referenceBP_array (short * *reference_pt,* unsigned int *turn*)

```
#include <ViennaRNA/structure_utils.h>
```

Make a reference base pair count matrix.

Get an upper triangular matrix containing the number of basepairs of a reference structure for each interval [i,j] with i<j. Access it via iindx!!!

Deprecated Use vrna_refBPcnt_matrix() instead

13.49.3.33 unsigned int* compute_BPdifferences (short * *pt1,* short * *pt2,* unsigned int *turn*)

```
#include <ViennaRNA/structure_utils.h>
```

Make a reference base pair distance matrix.

Get an upper triangular matrix containing the base pair distance of two reference structures for each interval [i,j] with i<j. Access it via iindx!!!

Deprecated Use vrna_refBPdist_matrix() instead

13.49.3.34 void parenthesis_structure (char * *structure,* vrna_bp_stack_t * *bp,* int *length*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a dot-backet/parenthesis structure from backtracking stack.

Deprecated use vrna_parenthesis_structure() instead

Note

This function is threadsafe

13.49.3.35 void parenthesis_zuker (char * *structure,* vrna_bp_stack_t * *bp,* int *length*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a dot-backet/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.

Deprecated use vrna_parenthesis_zuker instead

Note

This function is threadsafe

13.49.3.36 void bppm_to_structure (char * *structure*, FLT_OR_DBL * *pr*, unsigned int *length*)

```
#include <ViennaRNA/structure_utils.h>
```

Create a dot-bracket like structure string from base pair probability matrix.

Deprecated Use vrna_db_from_probs() instead!

13.49.3.37 char bppm_symbol (const float * *x*)

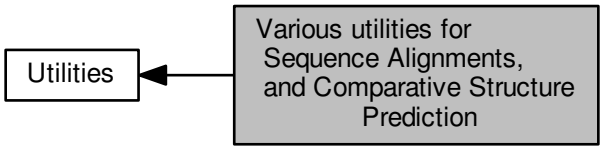
```
#include <ViennaRNA/structure_utils.h>
```

Get a pseudo dot bracket notation for a given probability information.

Deprecated Use vrna_bpp_symbol() instead!

13.50 Various utilities for Sequence Alignments, and Comparative Structure Prediction

Collaboration diagram for Various utilities for Sequence Alignments, and Comparative Structure Prediction:



Files

- file aln_util.h

  Various utility- and helper-functions for sequence alignments and comparative structure prediction.

Data Structures

- struct vrna_pinfo_s

  A base pair info structure. More...

Typedefs

- typedef struct vrna_pinfo_s vrna_pinfo_t

  Typename for the base pair info repesenting data structure vrna_pinfo_s.
- typedef struct vrna_pinfo_s pair_info

  Old typename of vrna_pinfo_s.

Functions

- vrna_pinfo_t * vrna_aln_pinfo (vrna_fold_compound_t *vc, const char *structure, double threshold)

  Retrieve an array of vrna_pinfo_t structures from precomputed pair probabilities.

13.50.1 Detailed Description

13.50.2 Data Structure Documentation

13.50.2.1 struct vrna_pinfo_s

A base pair info structure.

For each base pair (i,j) with i,j in [0, n-1] the structure lists:

- its probability 'p'
- an entropy-like measure for its well-definedness 'ent'
- the frequency of each type of pair in 'bp[]'
  - 'bp[0]' contains the number of non-compatible sequences
  - 'bp[1]' the number of CG pairs, etc.

Data Fields

- unsigned i

  nucleotide position i
- unsigned j

  nucleotide position j
- float p

  Probability.
- float ent

  Pseudo entropy for $p(i,j) = S_i + S_j - p_ij * ln(p_ij)$.
- short bp [8]

  Frequencies of pair_types.
- char comp

  1 iff pair is in mfe structure

13.50.3 Typedef Documentation

13.50.3.1 typedef struct vrna_pinfo_s pair_info

```
#include <ViennaRNA/aln_util.h>
```

Old typename of vrna_pinfo_s.

Deprecated Use vrna_pinfo_t instead!

13.50.4 Function Documentation

13.50.4.1 vrna_pinfo_t* vrna_aln_pinfo (vrna_fold_compound_t * *vc*, const char * *structure*, double *threshold*)

```
#include <ViennaRNA/aln_util.h>
```

Retrieve an array of vrna_pinfo_t structures from precomputed pair probabilities.

This array of structures contains information about positionwise pair probabilies, base pair entropy and more

See also

vrna_pinfo_t, and vrna_pf()

Parameters

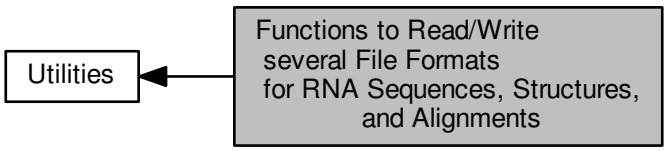
vc	The vrna_fold_compound_t of type VRNA_VC_TYPE_ALIGNMENT with precomputed partition function matrices
structure	An optional structure in dot-bracket notation (Maybe NULL)
threshold	Do not include results with pair probabilities below threshold

Returns

The vrna_pinfo_t array

13.51 Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments

Collaboration diagram for Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments:



Files

- file file_formats.h

  Functions dealing with file formats for RNA sequences, structures, and alignments.
- file ribo.h

  Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

Macros

- #define VRNA_OPTION_MULTILINE 32U

  Tell a function that an input is assumed to span several lines.
- #define VRNA_CONSTRAINT_MULTILINE 32U

  parse multiline constraint

Functions

- void vrna_file_helixlist (const char *seq, const char *db, float energy, FILE *file)

  Print a secondary structure as helix list.
- void vrna_file_connect (const char *seq, const char *db, float energy, const char *identifier, FILE *file)

  Print a secondary structure as connect table.
- void vrna_file_bpseq (const char *seq, const char *db, FILE *file)

  Print a secondary structure in bpseq format.
- void vrna_file_json (const char *seq, const char *db, double energy, const char *identifier, FILE *file)

  Print a secondary structure in jsonformat.
- unsigned int vrna_file_fasta_read_record (char **header, char **sequence, char ***rest, FILE *file, unsigned int options)

  Get a (fasta) data set from a file or stdin.
- char * vrna_extract_record_rest_structure (const char **lines, unsigned int length, unsigned int option)

  Extract a dot-bracket structure string from (multiline)character array.
- int vrna_file_SHAPE_read (const char *file_name, int length, double default_value, char *sequence, double *values)

Read data from a given SHAPE reactivity input file.

- vrna_plist_t * vrna_file_constraints_read (const char *filename, unsigned int length, unsigned int options)

  Read constraints from an input file.
- void vrna_extract_record_rest_constraint (char **cstruc, const char **lines, unsigned int option)

  Extract a hard constraint encoded as pseudo dot-bracket string.
- unsigned int read_record (char **header, char **sequence, char ***rest, unsigned int options)

  Get a data record from stdin.
- float ** readribosum (char *name)

  Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.

13.51.1 Detailed Description

13.51.2 Macro Definition Documentation

13.51.2.1 #define VRNA_OPTION_MULTILINE 32U

```
#include <ViennaRNA/file_formats.h>
```

Tell a function that an input is assumed to span several lines.

If used as input-option a function might also be returning this state telling that it has read data from multiple lines.

See also

vrna_extract_record_rest_structure(), vrna_file_fasta_read_record()

13.51.2.2 #define VRNA_CONSTRAINT_MULTILINE 32U

```
#include <ViennaRNA/file_formats.h>
```

parse multiline constraint

Deprecated see vrna_extract_record_rest_structure()

13.51.3 Function Documentation

13.51.3.1 void vrna_file_helixlist (const char * *seq*, const char * *db*, float *energy*, FILE * *file*)

```
#include <ViennaRNA/file_formats.h>
```

Print a secondary structure as helix list.

Parameters

seq	The RNA sequence
db	The structure in dot-bracket format
energy	Free energy of the structure in kcal/mol
file	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

13.51.3.2 void vrna_file_connect (const char * *seq*, const char * *db*, float *energy*, const char * *identifier*, FILE * *file*)

```
#include <ViennaRNA/file_formats.h>
```

Print a secondary structure as connect table.

Connect table file format looks like this:

```
300  ENERGY = 7.0  example
  1 G       0    2   22    1
  2 G       1    3   21    2
```

where the headerline is followed by 6 columns with:

1. Base number: index n
2. Base (A, C, G, T, U, X)
3. Index n-1 (0 if first nucleotide)
4. Index n+1 (0 if last nucleotide)
5. Number of the base to which n is paired. No pairing is indicated by 0 (zero).
6. Natural numbering.

Parameters

seq	The RNA sequence
db	The structure in dot-bracket format
energy	The free energy of the structure
identifier	An optional identifier for the sequence
file	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

13.51.3.3 void vrna_file_bpseq (const char * *seq*, const char * *db*, FILE * *file*)

```
#include <ViennaRNA/file_formats.h>
```

Print a secondary structure in bpseq format.

Parameters

seq	The RNA sequence
db	The structure in dot-bracket format
file	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

13.51.3.4 void vrna_file_json (const char * *seq*, const char * *db*, double *energy*, const char * *identifier*, FILE * *file*)

```
#include <ViennaRNA/file_formats.h>
```

Print a secondary structure in jsonformat.

Parameters

seq	The RNA sequence
db	The structure in dot-bracket format
energy	The free energy
identifier	An identifier for the sequence
file	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

13.51.3.5 unsigned int vrna_file_fasta_read_record (char ** *header*, char ** *sequence*, char *** *rest*, FILE * *file*, unsigned int *options*)

```
#include <ViennaRNA/file_formats.h>
```

Get a (fasta) data set from a file or stdin.

This function may be used to obtain complete datasets from a filehandle or stdin. A dataset is always defined to contain at least a sequence. If data starts with a fasta header, i.e. a line like

```
>some header info
```

then vrna_file_fasta_read_record() will assume that the sequence that follows the header may span over several lines. To disable this behavior and to assign a single line to the argument 'sequence' one can pass VRNA_INPUT_NO_SPAN in the 'options' argument. If no fasta header is read in the beginning of a data block, a sequence must not span over multiple lines!
Unless the options VRNA_INPUT_NOSKIP_COMMENTS or VRNA_INPUT_NOSKIP_BLANK_LINES are passed, a sequence may be interrupted by lines starting with a comment character or empty lines.
A sequence is regarded as completely read if it was either assumed to not span over multiple lines, a secondary structure or structure constraint follows the sequence on the next line, or a new header marks the beginning of a new sequence...
All lines following the sequence (this includes comments) that do not initiate a new dataset according to the above definition are available through the line-array 'rest'. Here one can usually find the structure constraint or other information belonging to the current dataset. Filling of 'rest' may be prevented by passing VRNA_INPUT_NO_REST to the options argument.

Note

This function will exit any program with an error message if no sequence could be read!
This function is NOT threadsafe! It uses a global variable to store information about the next data block.

The main purpose of this function is to be able to easily parse blocks of data in the header of a loop where all calculations for the appropriate data is done inside the loop. The loop may be then left on certain return values, e.g.:

```
00001 char *id, *seq, **rest;
00002 int  i;
00003 id = seq = NULL;
00004 rest = NULL;
00005 while(!(vrna_file_fasta_read_record(&id, &seq, &rest, NULL, 0) & (VRNA_INPUT_ERROR |
       VRNA_INPUT_QUIT))){
00006   if(id) printf("%s\n", id);
00007   printf("%s\n", seq);
00008   if(rest)
00009     for(i=0;rest[i];i++){
00010       printf("%s\n", rest[i]);
00011       free(rest[i]);
00012     }
00013   free(rest);
00014   free(seq);
00015   free(id);
00016 }
```

In the example above, the while loop will be terminated when vrna_file_fasta_read_record() returns either an error, EOF, or a user initiated quit request.
As long as data is read from stdin (we are passing NULL as the file pointer), the id is printed if it is available for the current block of data. The sequence will be printed in any case and if some more lines belong to the current block of data each line will be printed as well.

Note

Do not forget to free the memory occupied by header, sequence and rest!

Parameters

header	A pointer which will be set such that it points to the header of the record
sequence	A pointer which will be set such that it points to the sequence of the record
rest	A pointer which will be set such that it points to an array of lines which also belong to the record
file	A file handle to read from (if NULL, this function reads from stdin)
options	Some options which may be passed to alter the behavior of the function, use 0 for no options

Returns

A flag with information about what the function actually did read

13.51.3.6 char* vrna_extract_record_rest_structure (const char ** *lines*, unsigned int *length*, unsigned int *option*)

```
#include <ViennaRNA/file_formats.h>
```

Extract a dot-bracket structure string from (multiline)character array.

This function extracts a dot-bracket structure string from the 'rest' array as returned by vrna_file_fasta_read_record() and returns it. All occurences of comments within the 'lines' array will be skipped as long as they do not break the structure string. If no structure could be read, this function returns NULL.

Precondition

The argument 'lines' has to be a 2-dimensional character array as obtained by vrna_file_fasta_read_record()

See also

vrna_file_fasta_read_record()

Parameters

lines	The (multiline) character array to be parsed
length	The assumed length of the dot-bracket string (passing a value < 1 results in no length limit)
option	Some options which may be passed to alter the behavior of the function, use 0 for no options

Returns

The dot-bracket string read from lines or NULL

13.51.3.7 int vrna_file_SHAPE_read (const char * *file_name*, int *length*, double *default_value*, char * *sequence*, double * *values*)

```
#include <ViennaRNA/file_formats.h>
```

Read data from a given SHAPE reactivity input file.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the double array values.

Parameters

file_name	Path to the constraints file
length	Length of the sequence (file entries exceeding this limit will cause an error)
default_value	Value for missing indices
sequence	Pointer to an array used for storing the sequence obtained from the SHAPE reactivity file
values	Pointer to an array used for storing the values obtained from the SHAPE reactivity file

13.51.3.8 vrna_plist_t* vrna_file_constraints_read (const char * *filename*, unsigned int *length*, unsigned int *options*)

```
#include <ViennaRNA/file_formats.h>
```

Read constraints from an input file.

This function reads constraint definitions from a file and converts them into an array of vrna_plist_t data structures. The data fields of each individual returned plist entry may adopt the following configurations:

- plist.i == plist.j → single nucleotide constraint
- plist.i != plist.j → base pair constraint
- plist.i == 0 → End of list

13.51.3.9 void vrna_extract_record_rest_constraint (char ** *cstruc*, const char ** *lines*, unsigned int *option*)

```
#include <ViennaRNA/file_formats.h>
```

Extract a hard constraint encoded as pseudo dot-bracket string.

Deprecated Use vrna_extract_record_rest_structure() instead!

Precondition

The argument 'lines' has to be a 2-dimensional character array as obtained by vrna_file_fasta_read_record()

See also

vrna_file_fasta_read_record(), VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_↩ CONSTRAINT_DB_X VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK

Parameters

cstruc	A pointer to a character array that is used as pseudo dot-bracket output
lines	A 2-dimensional character array with the extension lines from the FASTA input
option	The option flags that define the behavior and recognition pattern of this function

13.51.3.10 unsigned int read_record (char ** *header*, char ** *sequence*, char *** *rest*, unsigned int *options*)

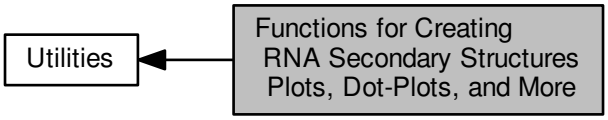
```
#include <ViennaRNA/file_formats.h>
```

Get a data record from stdin.

Deprecated This function is deprecated! Use vrna_file_fasta_read_record() as a replacment.

13.52 Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More

Collaboration diagram for Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More:



Files

- file naview.h
- file plot_aln.h

  Various functions for plotting Sequence / Structure Alignments.
- file plot_layouts.h

  Secondary structure plot layout algorithms.
- file plot_structure.h

  Various functions for plotting RNA secondary structures.
- file PS_dot.h

  Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Data Structures

- struct COORDINATE

  this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE More...

Macros

- #define VRNA_PLOT_TYPE_SIMPLE 0

  Definition of Plot type simple
- #define VRNA_PLOT_TYPE_NAVIEW 1

  Definition of Plot type Naview
- #define VRNA_PLOT_TYPE_CIRCULAR 2

  Definition of Plot type Circular

Functions

- int PS_color_aln (const char *structure, const char *filename, const char *seqs[ ], const char *names[ ])

  Produce PostScript sequence alignment color-annotated by consensus structure.
- int aliPS_color_aln (const char *structure, const char *filename, const char *seqs[ ], const char *names[ ])
- int simple_xy_coordinates (short *pair_table, float *X, float *Y)

  Calculate nucleotide coordinates for secondary structure plot the Simple way
- int simple_circplot_coordinates (short *pair_table, float *x, float *y)

  Calculate nucleotide coordinates for Circular Plot
- int vrna_file_PS_rnaplot (const char *seq, const char *structure, const char *file, vrna_md_t *md_p)

  Produce a secondary structure graph in PostScript and write it to 'filename'.
- int vrna_file_PS_rnaplot_a (const char *seq, const char *structure, const char *file, const char *pre, const char *post, vrna_md_t *md_p)

  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.
- int gmlRNA (char *string, char *structure, char *ssfile, char option)

  Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.
- int ssv_rna_plot (char *string, char *structure, char *ssfile)

  Produce a secondary structure graph in SStructView format.
- int svg_rna_plot (char *string, char *structure, char *ssfile)

  Produce a secondary structure plot in SVG format and write it to a file.
- int xrna_plot (char *string, char *structure, char *ssfile)

  Produce a secondary structure plot for further editing in XRNA.
- int PS_rna_plot (char *string, char *structure, char *file)

  Produce a secondary structure graph in PostScript and write it to 'filename'.
- int PS_rna_plot_a (char *string, char *structure, char *file, char *pre, char *post)

  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.
- int PS_rna_plot_a_gquad (char *string, char *structure, char *ssfile, char *pre, char *post)

  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)
- int PS_dot_plot_list (char *seq, char *filename, plist *pl, plist *mf, char *comment)

  Produce a postscript dot-plot from two pair lists.
- int PS_dot_plot (char *string, char *file)

  Produce postscript dot-plot.

Variables

- int rna_plot_type

  Switch for changing the secondary structure layout algorithm.

13.52.1 Detailed Description

13.52.2 Data Structure Documentation

13.52.2.1 struct COORDINATE

this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE

13.52.3 Macro Definition Documentation

13.52.3.1 #define VRNA_PLOT_TYPE_SIMPLE 0

```
#include <ViennaRNA/plot_layouts.h>
```

Definition of Plot type *simple*

This is the plot type definition for several RNA structure plotting functions telling them to use **Simple** plotting algorithm

See also

rna_plot_type, vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot(), svg_rna_plot(), gmlRNA(), ssv_rna_plot(), xrna_plot()

13.52.3.2 #define VRNA_PLOT_TYPE_NAVIEW 1

```
#include <ViennaRNA/plot_layouts.h>
```

Definition of Plot type *Naview*

This is the plot type definition for several RNA structure plotting functions telling them to use **Naview** plotting algorithm

See also

rna_plot_type, vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot(), svg_rna_plot(), gmlRNA(), ssv_rna_plot(), xrna_plot()

13.52.3.3 #define VRNA_PLOT_TYPE_CIRCULAR 2

```
#include <ViennaRNA/plot_layouts.h>
```

Definition of Plot type *Circular*

This is the plot type definition for several RNA structure plotting functions telling them to produce a **Circular plot**

See also

rna_plot_type, vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot(), svg_rna_plot(), gmlRNA(), ssv_rna_plot(), xrna_plot()

13.52.4 Function Documentation

13.52.4.1 int aliPS_color_aln (const char * *structure*, const char * *filename*, const char * *seqs[ ]*, const char * *names[ ]*)

```
#include <ViennaRNA/plot_aln.h>
```

PS_color_aln for duplexes

13.52.4.2 int simple_xy_coordinates (short * *pair_table*, float * *X*, float * *Y*)

```
#include <ViennaRNA/plot_layouts.h>
```

Calculate nucleotide coordinates for secondary structure plot the *Simple way*

See also

make_pair_table(), rna_plot_type, simple_circplot_coordinates(), naview_xy_coordinates(), vrna_file_PS_↩ rnaplot_a(), vrna_file_PS_rnaplot, svg_rna_plot()

Parameters

pair_table	The pair table of the secondary structure
X	a pointer to an array with enough allocated space to hold the x coordinates
Y	a pointer to an array with enough allocated space to hold the y coordinates

Returns

length of sequence on success, 0 otherwise

13.52.4.3 int simple_circplot_coordinates (short * *pair_table*, float * *x*, float * *y*)

```
#include <ViennaRNA/plot_layouts.h>
```

Calculate nucleotide coordinates for *Circular Plot*

This function calculates the coordinates of nucleotides mapped in equal distancies onto a unit circle.

Note

In order to draw nice arcs using quadratic bezier curves that connect base pairs one may calculate a second tangential point P^t in addition to the actual R^2 coordinates. the simplest way to do so may be to compute a radius scaling factor rs in the interval $[0, 1]$ that weights the proportion of base pair span to the actual length of the sequence. This scaling factor can then be used to calculate the coordinates for P^t, i.e. $P^t_x[i] = X[i] * rs$ and $P^t_y[i] = Y[i] * rs$.

See also

make_pair_table(), rna_plot_type, simple_xy_coordinates(), naview_xy_coordinates(), vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot, svg_rna_plot()

Parameters

pair_table	The pair table of the secondary structure
x	a pointer to an array with enough allocated space to hold the x coordinates
y	a pointer to an array with enough allocated space to hold the y coordinates

Returns

length of sequence on success, 0 otherwise

13.52.4.4 int vrna_file_PS_rnaplot (const char * *seq*, const char * *structure*, const char * *file*, vrna_md_t * *md_p*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

Note that this function has changed from previous versions and now expects the structure to be plotted in dot-bracket notation as an argument. It does not make use of the global base_pair array anymore.

Parameters

seq	The RNA sequence
structure	The secondary structure in dot-bracket notation
file	The filename of the postscript output
md_p	Model parameters used to generate a commandline option string in the output (Maybe NULL)

Returns

1 on success, 0 otherwise

13.52.4.5 int vrna_file_PS_rnaplot_a (const char * *seq*, const char * *structure*, const char * *file*, const char * *pre*, const char * *post*, vrna_md_t * *md_p*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

Same as vrna_file_PS_rnaplot() but adds extra PostScript macros for various annotations (see generated PS code). The 'pre' and 'post' variables contain PostScript code that is verbatim copied in the resulting PS file just before and after the structure plot. If both arguments ('pre' and 'post') are NULL, no additional macros will be printed into the PostScript.

Parameters

seq	The RNA sequence
structure	The secondary structure in dot-bracket notation
file	The filename of the postscript output
pre	PostScript code to appear before the secondary structure plot
post	PostScript code to appear after the secondary structure plot
md_p	Model parameters used to generate a commandline option string in the output (Maybe NULL)

Returns

1 on success, 0 otherwise

13.52.4.6 int gmlRNA (char * *string*, char * *structure*, char * *ssfile*, char *option*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.

If 'option' is an uppercase letter the RNA sequence is used to label nodes, if 'option' equals *'X'* or *'x'* the resulting file will coordinates for an initial layout of the graph.

Parameters

string	The RNA sequence
structure	The secondary structure in dot-bracket notation
ssfile	The filename of the gml output
option	The option flag

Returns

1 on success, 0 otherwise

13.52.4.7 int ssv_rna_plot (char * *string*, char * *structure*, char * *ssfile*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in SStructView format.

Write coord file for SStructView

Parameters

string	The RNA sequence
structure	The secondary structure in dot-bracket notation
ssfile	The filename of the ssv output

Returns

1 on success, 0 otherwise

13.52.4.8 int svg_rna_plot (char * *string*, char * *structure*, char * *ssfile*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure plot in SVG format and write it to a file.

Parameters

string	The RNA sequence
structure	The secondary structure in dot-bracket notation
ssfile	The filename of the svg output

Returns

1 on success, 0 otherwise

13.52.4.9 int xrna_plot (char * *string*, char * *structure*, char * *ssfile*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure plot for further editing in XRNA.

Parameters

string	The RNA sequence
structure	The secondary structure in dot-bracket notation
ssfile	The filename of the xrna output

Returns

1 on success, 0 otherwise

13.52.4.10 int PS_rna_plot (char * *string*, char * *structure*, char * *file*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

Deprecated Use vrna_file_PS_rnaplot() instead!

13.52.4.11 int PS_rna_plot_a (char * *string*, char * *structure*, char * *file*, char * *pre*, char * *post*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

Deprecated Use vrna_file_PS_rnaplot_a() instead!

13.52.4.12 int PS_rna_plot_a_gquad (char * *string*, char * *structure*, char * *ssfile*, char * *pre*, char * *post*)

```
#include <ViennaRNA/plot_structure.h>
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)

Deprecated Use vrna_file_PS_rnaplot_a() instead!

13.52.4.13 int PS_dot_plot_list (char * *seq*, char * *filename*, plist * *pl*, plist * *mf*, char * *comment*)

```
#include <ViennaRNA/PS_dot.h>
```

Produce a postscript dot-plot from two pair lists.

This function reads two plist structures (e.g. base pair probabilities and a secondary structure) as produced by assign_plist_from_pr() and assign_plist_from_db() and produces a postscript "dot plot" that is written to 'filename'. Using base pair probabilities in the first and mfe structure in the second plist, the resulting "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy structure.

See also

assign_plist_from_pr(), assign_plist_from_db()

Parameters

seq	The RNA sequence
filename	A filename for the postscript output
pl	The base pair probability pairlist
mf	The mfe secondary structure pairlist
comment	A comment

Returns

1 if postscript was successfully written, 0 otherwise

13.52.4.14 int PS_dot_plot (char * *string*, char * *file*)

```
#include <ViennaRNA/PS_dot.h>
```

Produce postscript dot-plot.

Wrapper to PS_dot_plot_list

Reads base pair probabilities produced by pf_fold() from the global array pr and the pair list base_pair produced by fold() and produces a postscript "dot plot" that is written to 'filename'. The "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy

Note

DO NOT USE THIS FUNCTION ANYMORE SINCE IT IS NOT THREADSAFE

Deprecated This function is deprecated and will be removed soon! Use PS_dot_plot_list() instead!

13.52.5 Variable Documentation

13.52.5.1 int rna_plot_type

```
#include <ViennaRNA/plot_layouts.h>
```

Switch for changing the secondary structure layout algorithm.

Current possibility are 0 for a simple radial drawing or 1 for the modified radial drawing taken from the *naview* program of [3].

Note

To provide thread safety please do not rely on this global variable in future implementations but pass a plot type flag directly to the function that decides which layout algorithm it may use!

See also

VRNA_PLOT_TYPE_SIMPLE, VRNA_PLOT_TYPE_NAVIEW, VRNA_PLOT_TYPE_CIRCULAR

Chapter 14

Data Structure Documentation

14.1 _struct_en Struct Reference

Data structure for energy_of_move()

14.1.1 Detailed Description

Data structure for energy_of_move()

The documentation for this struct was generated from the following file:

- ViennaRNA/move_set.h

14.2 Postorder_list Struct Reference

Postorder data structure.

14.2.1 Detailed Description

Postorder data structure.

The documentation for this struct was generated from the following file:

- ViennaRNA/dist_vars.h

14.3 swString Struct Reference

Some other data structure.

14.3.1 Detailed Description

Some other data structure.

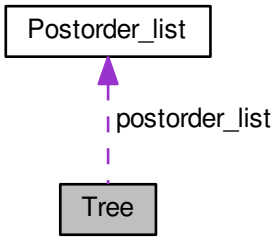
The documentation for this struct was generated from the following file:

- ViennaRNA/dist_vars.h

14.4 Tree Struct Reference

Tree data structure.

Collaboration diagram for Tree:



14.4.1 Detailed Description

Tree data structure.

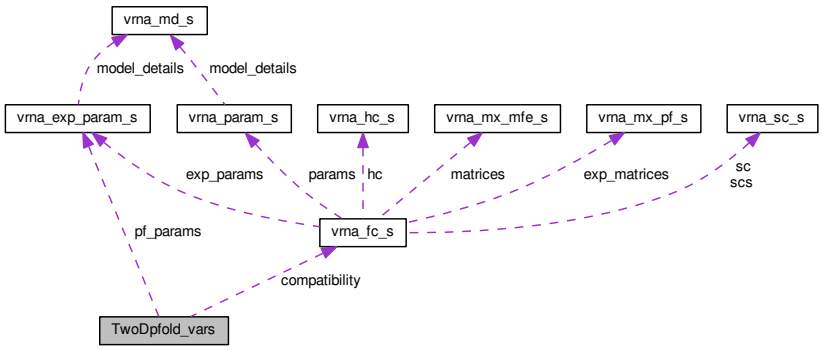
The documentation for this struct was generated from the following file:

- ViennaRNA/dist_vars.h

14.5 TwoDpfold_vars Struct Reference

Variables compound for 2Dfold partition function folding.

Collaboration diagram for TwoDpfold_vars:



Data Fields

- char * ptype
  Precomputed array of pair types.
- char * sequence
  The input sequence.
- short * S1
  The input sequences in numeric form.
- unsigned int maxD1
  Maximum allowed base pair distance to first reference.
- unsigned int maxD2
  Maximum allowed base pair distance to second reference.
- int * my_iindx
  Index for moving in quadratic distancy dimensions.
- int * jindx
  Index for moving in the triangular matrix qm1.
- unsigned int * referenceBPs1
  Matrix containing number of basepairs of reference structure1 in interval [i,j].
- unsigned int * referenceBPs2
  Matrix containing number of basepairs of reference structure2 in interval [i,j].
- unsigned int * bpdist
  Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].
- unsigned int * mm1
  Maximum matching matrix, reference struct 1 disallowed.
- unsigned int * mm2
  Maximum matching matrix, reference struct 2 disallowed.

14.5.1 Detailed Description

Variables compound for 2Dfold partition function folding.

Deprecated This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

The documentation for this struct was generated from the following file:

- ViennaRNA/2Dpfold.h

14.6 vrna_subopt_sol_s Struct Reference

Solution element from subopt.c.

Data Fields

- float energy

  Free Energy of structure in kcal/mol.
- char * structure

  Structure in dot-bracket notation.

14.6.1 Detailed Description

Solution element from subopt.c.

The documentation for this struct was generated from the following file:

- ViennaRNA/subopt.h

Chapter 15

File Documentation

15.1 ViennaRNA/1.8.4_epars.h File Reference

Free energy parameters for parameter file conversion.

15.1.1 Detailed Description

Free energy parameters for parameter file conversion.

This file contains the free energy parameters used in ViennaRNAPackage 1.8.4. They are summarized in:

D.H.Mathews, J. Sabina, M. ZUker, D.H. Turner "Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure" JMB, 288, pp 911-940, 1999

Enthalpies taken from:

A. Walter, D Turner, J Kim, M Lyttle, P M"uller, D Mathews, M Zuker "Coaxial stckaing of helices enhances binding of oligoribonucleotides.." PNAS, 91, pp 9218-9222, 1994

D.H. Turner, N. Sugimoto, and S.M. Freier. "RNA Structure Prediction", Ann. Rev. Biophys. Biophys. Chem. 17, 167-192, 1988.

John A.Jaeger, Douglas H.Turner, and Michael Zuker. "Improved predictions of secondary structures for RNA", PNAS, 86, 7706-7710, October 1989.

L. He, R. Kierzek, J. SantaLucia, A.E. Walter, D.H. Turner "Nearest-Neughbor Parameters for GU Mismatches...." Biochemistry 1991, 30 11124-11132

A.E. Peritz, R. Kierzek, N, Sugimoto, D.H. Turner "Thermodynamic Study of Internal Loops in Oligoribonucleotides..." Biochemistry 1991, 30, 6428–6435

15.2 ViennaRNA/1.8.4_intloops.h File Reference

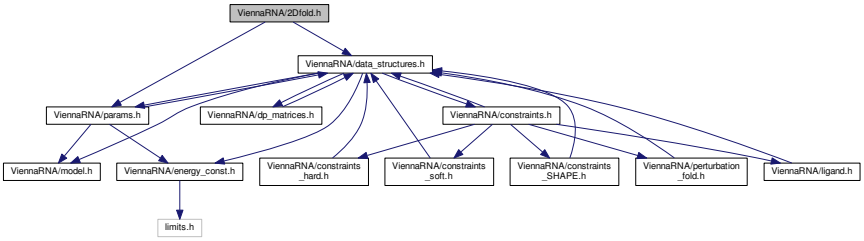
Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

15.2.1 Detailed Description

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

15.3 ViennaRNA/2Dfold.h File Reference

Include dependency graph for 2Dfold.h:



Data Structures

- struct vrna_sol_TwoD_t

  Solution element returned from vrna_mfe_TwoD() More...
- struct TwoDfold_vars

  Variables compound for 2Dfold MFE folding. More...

Typedefs

- typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t

  Solution element returned from vrna_mfe_TwoD()
- typedef struct TwoDfold_vars TwoDfold_vars

  Variables compound for 2Dfold MFE folding.

Functions

- vrna_sol_TwoD_t * vrna_mfe_TwoD (vrna_fold_compound_t *vc, int distance1, int distance2)

  Compute MFE's and representative for distance partitioning.
- char * vrna_backtrack5_TwoD (vrna_fold_compound_t *vc, int k, int l, unsigned int j)

  Backtrack a minimum free energy structure from a 5' section of specified length.
- TwoDfold_vars * get_TwoDfold_variables (const char *seq, const char *structure1, const char *structure2, int circ)

  Get a structure of type TwoDfold_vars prefilled with current global settings.
- void destroy_TwoDfold_variables (TwoDfold_vars *our_variables)

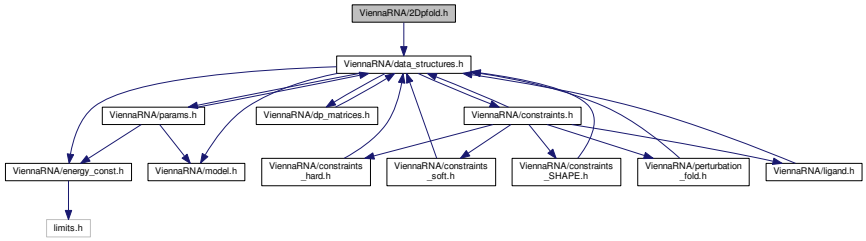
  Destroy a TwoDfold_vars datastructure without memory loss.
- vrna_sol_TwoD_t * TwoDfoldList (TwoDfold_vars *vars, int distance1, int distance2)

  Compute MFE's and representative for distance partitioning.
- char * TwoDfold_backtrack_f5 (unsigned int j, int k, int l, TwoDfold_vars *vars)

  Backtrack a minimum free energy structure from a 5' section of specified length.

15.4 ViennaRNA/2Dpfold.h File Reference

Include dependency graph for 2Dpfold.h:



Data Structures

- struct vrna_sol_TwoD_pf_t

  Solution element returned from vrna_pf_TwoD() More...
- struct TwoDpfold_vars

  Variables compound for 2Dfold partition function folding.

Typedefs

- typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t

  Solution element returned from vrna_pf_TwoD()

Functions

- vrna_sol_TwoD_pf_t * vrna_pf_TwoD (vrna_fold_compound_t *vc, int maxDistance1, int maxDistance2)

  Compute the partition function for all distance classes.
- char * vrna_pbacktrack_TwoD (vrna_fold_compound_t *vc, int d1, int d2)

  Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.
- char * vrna_pbacktrack5_TwoD (vrna_fold_compound_t *vc, int d1, int d2, unsigned int length)

  Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.
- TwoDpfold_vars * get_TwoDpfold_variables (const char *seq, const char *structure1, char *structure2, int circ)

  Get a datastructure containing all necessary attributes and global folding switches.
- void destroy_TwoDpfold_variables (TwoDpfold_vars *vars)

  Free all memory occupied by a TwoDpfold_vars datastructure.
- vrna_sol_TwoD_pf_t * TwoDpfoldList (TwoDpfold_vars *vars, int maxDistance1, int maxDistance2)

  Compute the partition function for all distance classes.
- char * TwoDpfold_pbacktrack (TwoDpfold_vars *vars, int d1, int d2)

  Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.
- char * TwoDpfold_pbacktrack5 (TwoDpfold_vars *vars, int d1, int d2, unsigned int length)

  Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

15.4.1 Function Documentation

15.4.1.1 TwoDpfold_vars* get_TwoDpfold_variables (const char * *seq*, const char * *structure1*, char * *structure2*, int *circ*)

Get a datastructure containing all necessary attributes and global folding switches.

This function prepares all necessary attributes and matrices etc which are needed for a call of TwoDpfold() . A snapshot of all current global model switches (dangles, temperature and so on) is done and stored in the returned datastructure. Additionally, all matrices that will hold the partition function values are prepared.

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

Parameters

seq	the RNA sequence in uppercase format with letters from the alphabet {AUCG}
structure1	the first reference structure in dot-bracket notation
structure2	the second reference structure in dot-bracket notation
circ	a switch indicating if the sequence is linear (0) or circular (1)

Returns

the datastructure containing all necessary partition function attributes

15.4.1.2 void destroy_TwoDpfold_variables (TwoDpfold_vars * *vars*)

Free all memory occupied by a TwoDpfold_vars datastructure.

This function free's all memory occupied by a datastructure obtained from from get_TwoDpfold_variabless() or get_TwoDpfold_variables_from_MFE()

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

See also

get_TwoDpfold_variables(), get_TwoDpfold_variables_from_MFE()

Parameters

vars	the datastructure to be free'd

15.4.1.3 vrna_sol_TwoD_pf_t* TwoDpfoldList (TwoDpfold_vars * *vars*, int *maxDistance1*, int *maxDistance2*)

Compute the partition function for all distance classes.

This function computes the partition functions for all distance classes according the two reference structures specified in the datastructure 'vars'. Similar to TwoDfold() the arguments maxDistance1 and maxDistance2 specify the maximum distance to both reference structures. A value of '-1' in either of them makes the appropriate distance restrictionless, i.e. all basepair distancies to the reference are taken into account during computation. In case there is a restriction, the returned solution contains an entry where the attribute k=l=-1 contains the partition function for all structures exceeding the restriction. A values of INF in the attribute 'k' of the returned list denotes the end of the list

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

See also

get_TwoDpfold_variables(), destroy_TwoDpfold_variables(), vrna_sol_TwoD_pf_t

Parameters

vars	the datastructure containing all necessary folding attributes and matrices
maxDistance1	the maximum basepair distance to reference1 (may be -1)
maxDistance2	the maximum basepair distance to reference2 (may be -1)

Returns

a list of partition funtions for the appropriate distance classes

15.4.1.4 char* TwoDpfold_pbacktrack (TwoDpfold_vars * *vars*, int *d1*, int *d2*)

Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

If the argument 'd1' is set to '-1', the structure will be backtracked in the distance class where all structures exceeding the maximum basepair distance to either of the references reside.

Precondition

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to TwoDpfold() preceding this function is mandatory!

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold↩ _compound_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack_TwoD(), and vrna_fold_compound_free() instead!

See also

TwoDpfold()

Parameters

in	*vars*	the datastructure containing all necessary folding attributes and matrices
in	*d1*	the distance to reference1 (may be -1)
in	*d2*	the distance to reference2

Returns

A sampled secondary structure in dot-bracket notation

15.4.1.5 char* TwoDpfold_pbacktrack5 (TwoDpfold_vars * *vars*, int *d1*, int *d2*, unsigned int *length*)

Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

This function does essentially the same as TwoDpfold_pbacktrack() with the only difference that partial structures, i.e. structures beginning from the 5' end with a specified length of the sequence, are backtracked

Note

This function does not work (since it makes no sense) for circular RNA sequences!

Precondition

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to TwoDpfold() preceding this function is mandatory!

Deprecated Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack5_TwoD(), and vrna_fold_compound_free() instead!

See also

TwoDpfold_pbacktrack(), TwoDpfold()

Parameters

in	*vars*	the datastructure containing all necessary folding attributes and matrices
in	*d1*	the distance to reference1 (may be -1)
in	*d2*	the distance to reference2
in	*length*	the length of the structure beginning from the 5' end

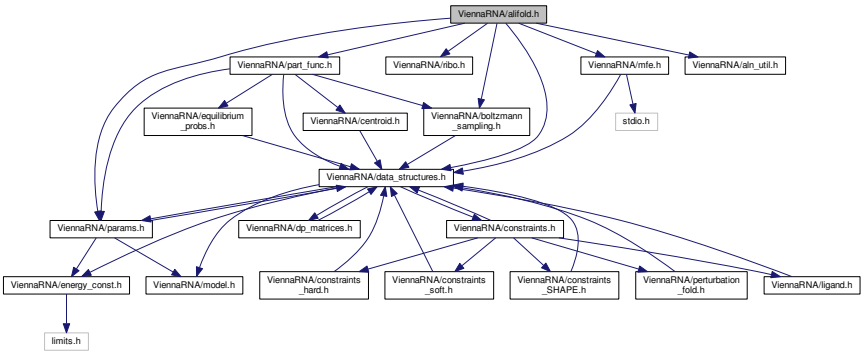
Returns

A sampled secondary structure in dot-bracket notation

15.5 ViennaRNA/alifold.h File Reference

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

Include dependency graph for alifold.h:



Functions

- float vrna_alifold (const char **sequences, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.
- float vrna_circalifold (const char **sequences, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.
- float vrna_pf_alifold (const char **sequences, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for an RNA sequence alignment using a comparative method.
- float vrna_pf_circalifold (const char **sequences, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.
- float alifold (const char **strings, char *structure)

  Compute MFE and according consensus structure of an alignment of sequences.
- float circalifold (const char **strings, char *structure)

  Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.
- void free_alifold_arrays (void)

  Free the memory occupied by MFE alifold functions.
- float energy_of_alistruct (const char **sequences, const char *structure, int n_seq, float *energy)

  Calculate the free energy of a consensus structure given a set of aligned sequences.
- float alipf_fold_par (const char **sequences, char *structure, vrna_plist_t **pl, vrna_exp_param_↩t *parameters, int calculate_bppm, int is_constrained, int is_circular)
- float alipf_fold (const char **sequences, char *structure, vrna_plist_t **pl)

  The partition function version of alifold() works in analogy to pf_fold(). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of vrna_pinfo_t structs. The list is terminated by the first entry with pi.i = 0.
- float alipf_circ_fold (const char **sequences, char *structure, vrna_plist_t **pl)
- FLT_OR_DBL * export_ali_bppm (void)

  Get a pointer to the base pair probability array.
- void free_alipf_arrays (void)

  Free the memory occupied by folding matrices allocated by alipf_fold, alipf_circ_fold, etc.
- char * alipbacktrack (double *prob)

  Sample a consensus secondary structure from the Boltzmann ensemble according its probability.

- int get_alipf_arrays (short ***S_p, short ***S5_p, short ***S3_p, unsigned short ***a2s_p, char ***Ss↩ _p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm_p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p, short **pscore)

  Get pointers to (almost) all relavant arrays used in alifold's partition function computation.
- void update_alifold_params (void)

  Update the energy parameters for alifold function.

Variables

- double cv_fact

  This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.
- double nc_fact

  This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

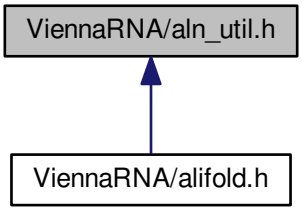
15.5.1 Detailed Description

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

15.6 ViennaRNA/aln_util.h File Reference

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_pinfo_s

  A base pair info structure. More...

Typedefs

- typedef struct vrna_pinfo_s vrna_pinfo_t

  Typename for the base pair info repesenting data structure vrna_pinfo_s.
- typedef struct vrna_pinfo_s pair_info

  Old typename of vrna_pinfo_s.

Functions

- int vrna_aln_mpi (char ∗Alseq[ ], int n_seq, int length, int ∗mini)

  Get the mean pairwise identity in steps from ?to?(ident)
- vrna_pinfo_t ∗ vrna_aln_pinfo (vrna_fold_compound_t ∗vc, const char ∗structure, double threshold)

  Retrieve an array of vrna_pinfo_t structures from precomputed pair probabilities.
- int get_mpi (char ∗Alseq[ ], int n_seq, int length, int ∗mini)

  Get the mean pairwise identity in steps from ?to?(ident)
- void encode_ali_sequence (const char ∗sequence, short ∗S, short ∗s5, short ∗s3, char ∗ss, unsigned short ∗as, int circ)

  Get arrays with encoded sequence of the alignment.
- void alloc_sequence_arrays (const char ∗∗sequences, short ∗∗∗S, short ∗∗∗S5, short ∗∗∗S3, unsigned short ∗∗∗a2s, char ∗∗∗Ss, int circ)

  Allocate memory for sequence array used to deal with aligned sequences.
- void free_sequence_arrays (unsigned int n_seq, short ∗∗∗S, short ∗∗∗S5, short ∗∗∗S3, unsigned short ∗∗∗a2s, char ∗∗∗Ss)

  Free the memory of the sequence arrays used to deal with aligned sequences.

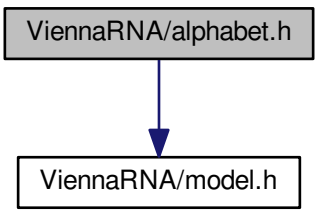
15.6.1 Detailed Description

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

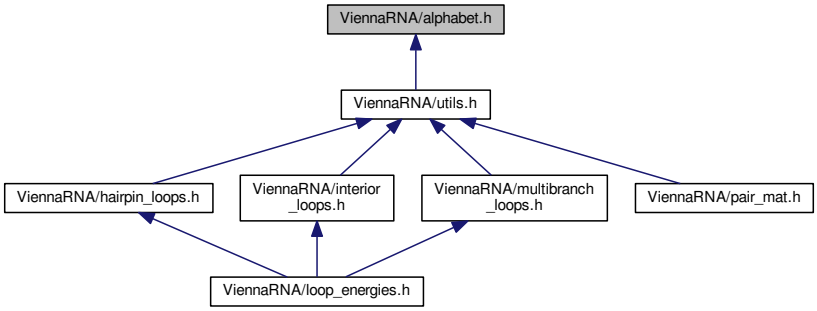
15.7 ViennaRNA/alphabet.h File Reference

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

Include dependency graph for alphabet.h:



This graph shows which files directly or indirectly include this file:



Functions

- char * vrna_ptypes (const short *S, vrna_md_t *md)

  Get an array of the numerical encoding for each possible base pair (i,j)
- short * vrna_seq_encode (const char *sequence, vrna_md_t *md)

  Get a numerical representation of the nucleotide sequence.
- short * vrna_seq_encode_simple (const char *sequence, vrna_md_t *md)

  Get a numerical representation of the nucleotide sequence (simple version)
- int vrna_nucleotide_encode (char c, vrna_md_t *md)

  Encode a nucleotide character to numerical value.
- char vrna_nucleotide_decode (int enc, vrna_md_t *md)

  Decode a numerical representation of a nucleotide back into nucleotide alphabet.

15.7.1 Detailed Description

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

15.7.2 Function Documentation

15.7.2.1 char* vrna_ptypes (const short * *S*, vrna_md_t * *md*)

Get an array of the numerical encoding for each possible base pair (i,j)

Note

This array is always indexed in column-wise order, in contrast to previously different indexing between mfe and pf variants!

See also

vrna_idx_col_wise(), vrna_fold_compound_t

15.7.2.2 int vrna_nucleotide_encode (char *c*, vrna_md_t * *md*)

Encode a nucleotide character to numerical value.

This function encodes a nucleotide character to its numerical representation as required by many functions in RNAlib.

See also

vrna_nucleotide_decode(), vrna_seq_encode()

Parameters

c	The nucleotide character to encode
md	The model details that determine the kind of encoding

Returns

The encoded nucleotide

15.7.2.3 char vrna_nucleotide_decode (int *enc*, vrna_md_t * *md*)

Decode a numerical representation of a nucleotide back into nucleotide alphabet.

This function decodes a numerical representation of a nucleotide character back into nucleotide alphabet

See also

vrna_nucleotide_encode(), vrna_seq_encode()

Parameters

enc	The encoded nucleotide
md	The model details that determine the kind of decoding

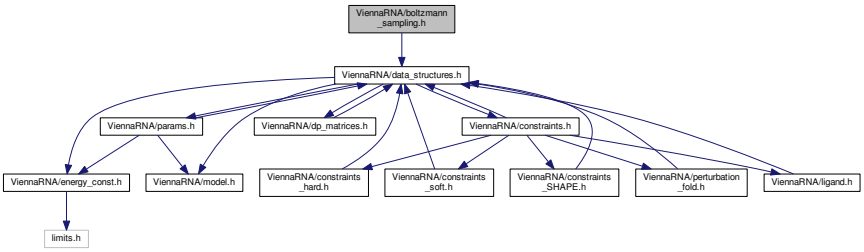
Returns

The decoded nucleotide character

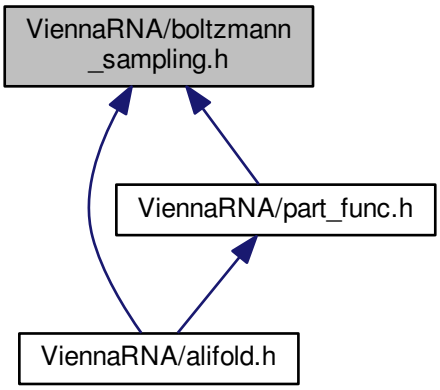
15.8 ViennaRNA/boltzmann_sampling.h File Reference

Boltzmann Sampling of secondary structures from the ensemble.

Include dependency graph for boltzmann_sampling.h:



This graph shows which files directly or indirectly include this file:



Functions

- char * vrna_pbacktrack5 (vrna_fold_compound_t *vc, int length)

  Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.
- char * vrna_pbacktrack (vrna_fold_compound_t *vc)

  Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.

15.8.1 Detailed Description

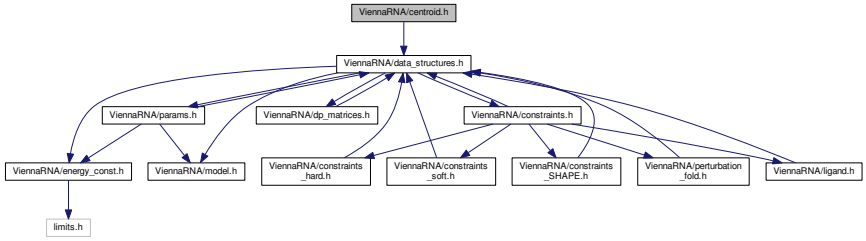
Boltzmann Sampling of secondary structures from the ensemble.

A.k.a. Stochastic backtracking

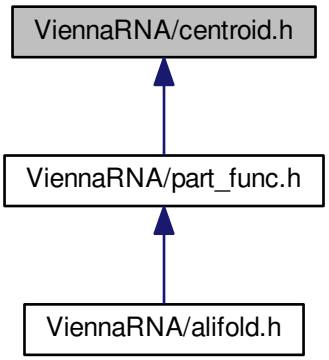
15.9 ViennaRNA/centroid.h File Reference

Centroid structure computation.

Include dependency graph for centroid.h:



This graph shows which files directly or indirectly include this file:



Functions

- char * vrna_centroid (vrna_fold_compound_t *vc, double *dist)

  Get the centroid structure of the ensemble.
- char * vrna_centroid_from_plist (int length, double *dist, vrna_plist_t *pl)

  Get the centroid structure of the ensemble.
- char * vrna_centroid_from_probs (int length, double *dist, FLT_OR_DBL *probs)

  Get the centroid structure of the ensemble.
- char * get_centroid_struct_pl (int length, double *dist, vrna_plist_t *pl)

  Get the centroid structure of the ensemble.
- char * get_centroid_struct_pr (int length, double *dist, FLT_OR_DBL *pr)

  Get the centroid structure of the ensemble.

15.9.1 Detailed Description

Centroid structure computation.

15.9.2 Function Documentation

15.9.2.1 char* get_centroid_struct_pl (int *length,* double * *dist,* vrna_plist_t * *pl*)

Get the centroid structure of the ensemble.

Deprecated This function was renamed to vrna_centroid_from_plist()

15.9.2.2 char* get_centroid_struct_pr (int *length,* double * *dist,* FLT_OR_DBL * *pr*)

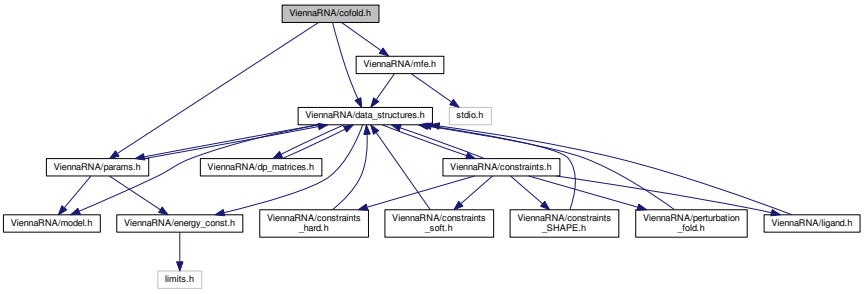
Get the centroid structure of the ensemble.

Deprecated This function was renamed to vrna_centroid_from_probs()

15.10 ViennaRNA/cofold.h File Reference

MFE version of cofolding routines.

Include dependency graph for cofold.h:



Functions

- float vrna_cofold (const char *sequence, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.
- float cofold (const char *sequence, char *structure)

  Compute the minimum free energy of two interacting RNA molecules.
- float cofold_par (const char *string, char *structure, vrna_param_t *parameters, int is_constrained)

  Compute the minimum free energy of two interacting RNA molecules.
- void free_co_arrays (void)

  Free memory occupied by cofold()
- void update_cofold_params (void)

  Recalculate parameters.
- void update_cofold_params_par (vrna_param_t *parameters)

  Recalculate parameters.
- void export_cofold_arrays_gq (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **ggg_p, int **indx_p, char **ptype_p)

  Export the arrays of partition function cofold (with gquadruplex support)
- void export_cofold_arrays (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **indx_p, char **ptype_p)

  Export the arrays of partition function cofold.
- void get_monomere_mfes (float *e1, float *e2)

  get_monomer_free_energies
- void initialize_cofold (int length)

15.10.1 Detailed Description

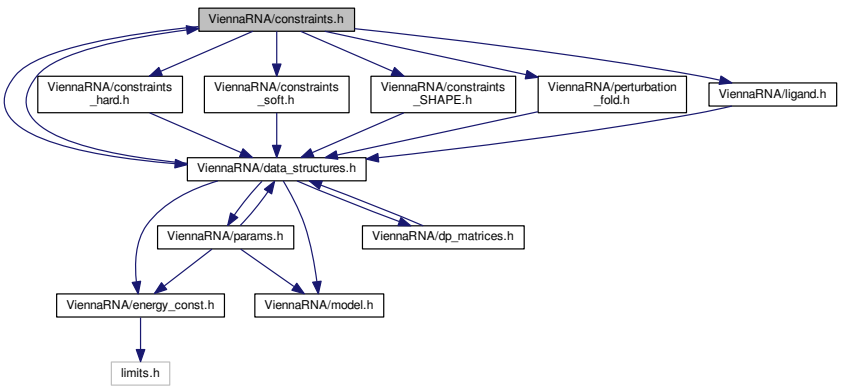
MFE version of cofolding routines.

This file includes (almost) all function declarations within the **RNAlib** that are related to MFE Cofolding... This also includes the Zuker suboptimals calculations, since they are implemented using the cofold routines.

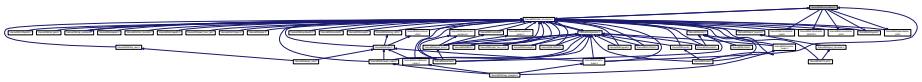
15.11 ViennaRNA/constraints.h File Reference

Functions and data structures for constraining secondary structure predictions and evaluation.

Include dependency graph for constraints.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VRNA_CONSTRAINT_FILE 0

  Flag for vrna_constraints_add() to indicate that constraints are present in a text file.
- #define VRNA_CONSTRAINT_SOFT_MFE 0

  Indicate generation of constraints for MFE folding.
- #define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF

  Indicate generation of constraints for partition function computation.
- #define VRNA_DECOMP_PAIR_HP 1

  Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.
- #define VRNA_DECOMP_PAIR_IL 2

  Indicator for interior loop decomposition step.
- #define VRNA_DECOMP_PAIR_ML 3

  Indicator for multibranch loop decomposition step.
- #define VRNA_DECOMP_ML_ML_ML 5

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_STEM 4

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_ML 6

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_UP 11

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_ML_STEM 20

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_ML_COAXIAL 13

  Indicator for decomposition of multibranch loop part.
- #define VRNA_DECOMP_EXT_EXT 9

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_UP 8

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_STEM 14

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_EXT_EXT 15

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_STEM_EXT 16

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_STEM_OUTSIDE 17

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_EXT_STEM 18

  Indicator for decomposition of exterior loop part.
- #define VRNA_DECOMP_EXT_EXT_STEM1 19

  Indicator for decomposition of exterior loop part.

Functions

- void vrna_constraints_add (vrna_fold_compound_t *vc, const char *constraint, unsigned int options)

  Add constraints to a vrna_fold_compound_t data structure.

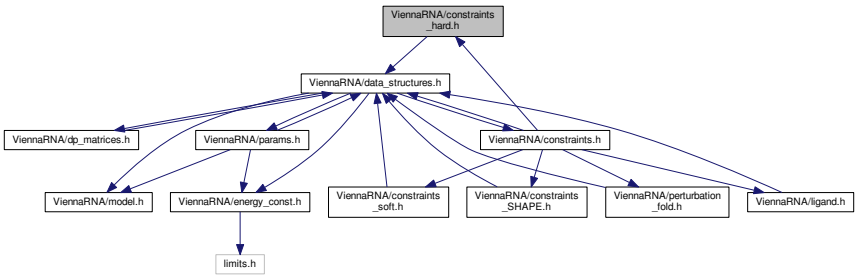
15.11.1 Detailed Description

Functions and data structures for constraining secondary structure predictions and evaluation.

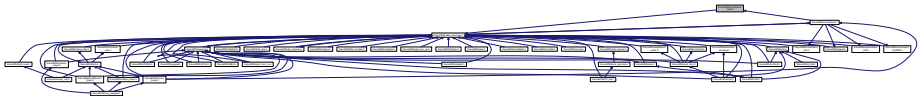
15.12 ViennaRNA/constraints_hard.h File Reference

Functions and data structures for handling of secondary structure hard constraints.

Include dependency graph for constraints_hard.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_hc_s

  The hard constraints data structure. More...

- struct vrna_hc_up_s

  A single hard constraint for a single nucleotide. More...

Macros

- #define VRNA_CONSTRAINT_NO_HEADER 0

  do not print the header information line
- #define VRNA_CONSTRAINT_DB 16384U

  Flag for vrna_constraints_add() to indicate that constraint is passed in pseudo dot-bracket notation.
- #define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U

  Switch for dot-bracket structure constraint to enforce base pairs.
- #define VRNA_CONSTRAINT_DB_PIPE 65536U

  Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.
- #define VRNA_CONSTRAINT_DB_DOT 131072U

  dot '.' switch for structure constraints (no constraint at all)
- #define VRNA_CONSTRAINT_DB_X 262144U

  'x' switch for structure constraint (base must not pair)
- #define VRNA_CONSTRAINT_DB_ANG_BRACK 524288U

  angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)
- #define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U

  round brackets '(',')' switch for structure constraint (base i pairs base j)
- #define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U

  Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.
- #define VRNA_CONSTRAINT_DB_INTERMOL 4194304U

  Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.
- #define VRNA_CONSTRAINT_DB_GQUAD 8388608U

  '+' switch for structure constraint (base is involved in a gquad)
- #define VRNA_CONSTRAINT_DB_DEFAULT

  Switch for dot-bracket structure constraint with default symbols.
- #define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP (char)0x01

  Hard constraints flag, base pair in the exterior loop.
- #define VRNA_CONSTRAINT_CONTEXT_HP_LOOP (char)0x02

  Hard constraints flag, base pair encloses hairpin loop.
- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP (char)0x04

  Hard constraints flag, base pair encloses an interior loop.
- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC (char)0x08

  Hard constraints flag, base pair encloses a multi branch loop.
- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP (char)0x10

  Hard constraints flag, base pair is enclosed in an interior loop.
- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC (char)0x20

  Hard constraints flag, base pair is enclosed in a multi branch loop.
- #define VRNA_CONSTRAINT_CONTEXT_ENFORCE (char)0x40

  Hard constraint flag to indicate enforcement of constraints.
- #define VRNA_CONSTRAINT_CONTEXT_NO_REMOVE (char)0x80

  Hard constraint flag to indicate not to remove base pairs that conflict with a given constraint.
- #define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

  Hard constraints flag, shortcut for all base pairs.

Typedefs

- typedef struct vrna_hc_s vrna_hc_t

  Typename for the hard constraints data structure vrna_hc_s.
- typedef struct vrna_hc_up_s vrna_hc_up_t

  Typename for the single nucleotide hard constraint data structure vrna_hc_up_s.
- typedef char(vrna_callback_hc_evaluate) (int i, int j, int k, int l, char d, void *data)

  Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.

Functions

- void vrna_message_constraint_options (unsigned int option)

  Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)
- void vrna_message_constraint_options_all (void)

  Print structure constraint characters to stdout (full constraint support)
- void vrna_hc_init (vrna_fold_compound_t *vc)

  Initialize/Reset hard constraints to default values.
- void vrna_hc_add_up (vrna_fold_compound_t *vc, int i, char option)

  Make a certain nucleotide unpaired.
- int vrna_hc_add_up_batch (vrna_fold_compound_t *vc, vrna_hc_up_t *constraints)

  Apply a list of hard constraints for single nucleotides.
- void vrna_hc_add_bp (vrna_fold_compound_t *vc, int i, int j, char option)

  Favorize/Enforce a certain base pair (i,j)
- void vrna_hc_add_bp_nonspecific (vrna_fold_compound_t *vc, int i, int d, char option)

  Enforce a nucleotide to be paired (upstream/downstream)
- void vrna_hc_free (vrna_hc_t *hc)

  Free the memory allocated by a vrna_hc_t data structure.
- int vrna_hc_add_from_db (vrna_fold_compound_t *vc, const char *constraint, unsigned int options)

  Add hard constraints from pseudo dot-bracket notation.
- void print_tty_constraint (unsigned int option)

  Print structure constraint characters to stdout. (constraint support is specified by option parameter)
- void print_tty_constraint_full (void)

  Print structure constraint characters to stdout (full constraint support)
- void constrain_ptypes (const char *constraint, unsigned int length, char *ptype, int *BP, int min_loop_size, unsigned int idx_type)

  Insert constraining pair types according to constraint structure string.

15.12.1 Detailed Description

Functions and data structures for handling of secondary structure hard constraints.

15.12.2 Macro Definition Documentation

15.12.2.1 #define VRNA_CONSTRAINT_NO_HEADER 0

do not print the header information line

Deprecated This mode is not supported anymore!

15.12.2.2 #define VRNA_CONSTRAINT_DB_ANG_BRACK 524288U

angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)

See also

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_↩ constraint_options_all()

15.12.3 Function Documentation

15.12.3.1 void print_tty_constraint (unsigned int *option*)

Print structure constraint characters to stdout. (constraint support is specified by option parameter)

Deprecated Use vrna_message_constraints() instead!

Parameters

option	Option switch that tells which constraint help will be printed

15.12.3.2 void print_tty_constraint_full (void)

Print structure constraint characters to stdout (full constraint support)

Deprecated Use vrna_message_constraint_options_all() instead!

15.12.3.3 void constrain_ptypes (const char * *constraint*, unsigned int *length*, char * *ptype*, int * *BP*, int *min_loop_size*, unsigned int *idx_type*)

Insert constraining pair types according to constraint structure string.

Deprecated Do not use this function anymore! Structure constraints are now handled through vrna_hc_t and related functions.

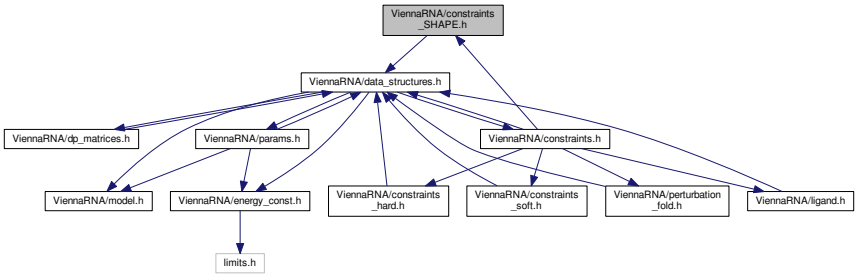
Parameters

constraint	The structure constraint string
length	The actual length of the sequence (constraint may be shorter)
ptype	A pointer to the basepair type array
BP	(not used anymore)
min_loop_size	The minimal loop size (usually TURN)
idx_type	Define the access type for base pair type array (0 = indx, 1 = iindx)

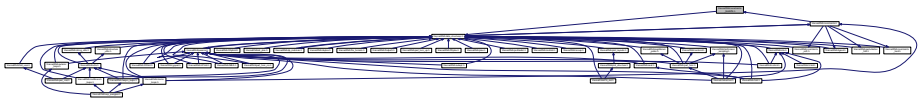
15.13 ViennaRNA/constraints_SHAPE.h File Reference

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

Include dependency graph for constraints_SHAPE.h:



This graph shows which files directly or indirectly include this file:



Functions

- int vrna_sc_add_SHAPE_deigan (vrna_fold_compound_t *vc, const double *reactivities, double m, double b, unsigned int options)

  Add SHAPE reactivity data as soft constraints (Deigan et al. method)
- int vrna_sc_add_SHAPE_deigan_ali (vrna_fold_compound_t *vc, const char **shape_files, const int *shape_file_association, double m, double b, unsigned int options)

  Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)
- int vrna_sc_add_SHAPE_zarringhalam (vrna_fold_compound_t *vc, const double *reactivities, double b, double default_value, const char *shape_conversion, unsigned int options)

  Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)
- int vrna_sc_SHAPE_parse_method (const char *method_string, char *method, float *param_1, float *param_2)

  Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.
- int vrna_sc_SHAPE_to_pr (const char *shape_conversion, double *values, int length, double default_value)

  Convert SHAPE reactivity values to probabilities for being unpaired.

15.13.1 Detailed Description

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

15.13.2 Function Documentation

15.13.2.1 int vrna_sc_SHAPE_parse_method (const char * *method_string*, char * *method*, float * *param_1*, float * *param_2*)

Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.

Parameters

method_string	The string that contains the encoded SHAPE reactivity conversion method
method	A pointer to the memory location where the method character will be stored
param_1	A pointer to the memory location where the first parameter of the corresponding method will be stored
param_2	A pointer to the memory location where the second parameter of the corresponding method will be stored

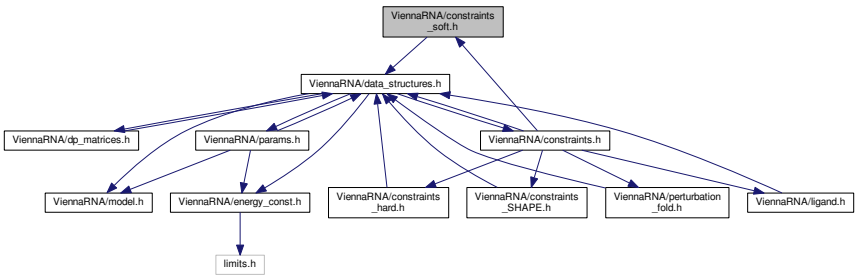
Returns

1 on successful extraction of the method, 0 on errors

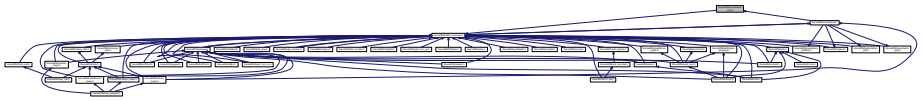
15.14 ViennaRNA/constraints_soft.h File Reference

Functions and data structures for secondary structure soft constraints.

Include dependency graph for constraints_soft.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_sc_s

  The soft constraints data structure. *More...*

Typedefs

- typedef struct vrna_sc_s vrna_sc_t

  Typename for the soft constraints data structure vrna_sc_s.
- typedef int(vrna_callback_sc_energy) (int i, int j, int k, int l, char d, void ∗data)

  Callback to retrieve pseudo energy contribution for soft constraint feature.
- typedef FLT_OR_DBL(vrna_callback_sc_exp_energy) (int i, int j, int k, int l, char d, void ∗data)

  Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.
- typedef vrna_basepair_t ∗(vrna_callback_sc_backtrack) (int i, int j, int k, int l, char d, void ∗data)

  Callback to retrieve auxiliary base pairs for soft constraint feature.

Functions

- void vrna_sc_init (vrna_fold_compound_t *vc)

  Initialize an empty soft constraints data structure within a vrna_fold_compound_t.
- void vrna_sc_add_bp (vrna_fold_compound_t *vc, const FLT_OR_DBL **constraints, unsigned int options)

  Add soft constraints for paired nucleotides.
- void vrna_sc_add_up (vrna_fold_compound_t *vc, const FLT_OR_DBL *constraints, unsigned int options)

  Add soft constraints for unpaired nucleotides.
- void vrna_sc_remove (vrna_fold_compound_t *vc)

  Remove soft constraints from vrna_fold_compound_t.
- void vrna_sc_free (vrna_sc_t *sc)

  Free memory occupied by a vrna_sc_t data structure.
- void vrna_sc_add_data (vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *free_data)

  Add an auxiliary data structure for the generic soft constraints callback function.
- void vrna_sc_add_f (vrna_fold_compound_t *vc, vrna_callback_sc_energy *f)

  Bind a function pointer for generic soft constraint feature (MFE version)
- void vrna_sc_add_bt (vrna_fold_compound_t *vc, vrna_callback_sc_backtrack *f)

  Bind a backtracking function pointer for generic soft constraint feature.
- void vrna_sc_add_exp_f (vrna_fold_compound_t *vc, vrna_callback_sc_exp_energy *exp_f)

  Bind a function pointer for generic soft constraint feature (PF version)

15.14.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

15.15 ViennaRNA/convert_epars.h File Reference

Functions and definitions for energy parameter file format conversion.

Macros

- #define VRNA_CONVERT_OUTPUT_ALL 1U
- #define VRNA_CONVERT_OUTPUT_HP 2U
- #define VRNA_CONVERT_OUTPUT_STACK 4U
- #define VRNA_CONVERT_OUTPUT_MM_HP 8U
- #define VRNA_CONVERT_OUTPUT_MM_INT 16U
- #define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U
- #define VRNA_CONVERT_OUTPUT_MM_INT_23 64U
- #define VRNA_CONVERT_OUTPUT_MM_MULTI 128U
- #define VRNA_CONVERT_OUTPUT_MM_EXT 256U
- #define VRNA_CONVERT_OUTPUT_DANGLE5 512U
- #define VRNA_CONVERT_OUTPUT_DANGLE3 1024U
- #define VRNA_CONVERT_OUTPUT_INT_11 2048U
- #define VRNA_CONVERT_OUTPUT_INT_21 4096U
- #define VRNA_CONVERT_OUTPUT_INT_22 8192U
- #define VRNA_CONVERT_OUTPUT_BULGE 16384U
- #define VRNA_CONVERT_OUTPUT_INT 32768U
- #define VRNA_CONVERT_OUTPUT_ML 65536U
- #define VRNA_CONVERT_OUTPUT_MISC 131072U
- #define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U
- #define VRNA_CONVERT_OUTPUT_VANILLA 524288U
- #define VRNA_CONVERT_OUTPUT_NINIO 1048576U
- #define VRNA_CONVERT_OUTPUT_DUMP 2097152U

Functions

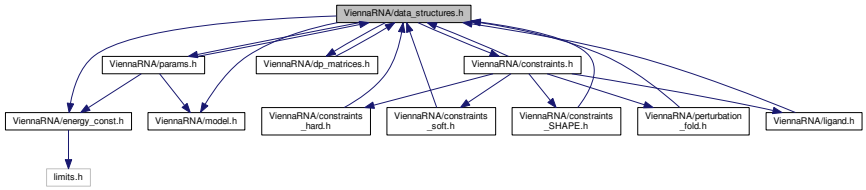
- void convert_parameter_file (const char *iname, const char *oname, unsigned int options)

15.15.1 Detailed Description

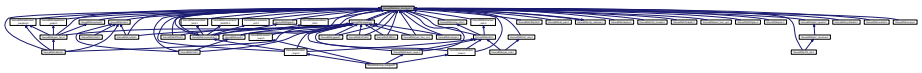
Functions and definitions for energy parameter file format conversion.

15.16 ViennaRNA/data_structures.h File Reference

Include dependency graph for data_structures.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_basepair_s

  Base pair data structure used in subopt.c. More...
- struct vrna_plist_s

  this datastructure is used as input parameter in functions of PS_dot.h and others More...
- struct vrna_cpair_s

  this datastructure is used as input parameter in functions of PS_dot.c More...
- struct vrna_sect_s

  Stack of partial structures for backtracking. More...
- struct vrna_bp_stack_s

  Base pair stack element. More...
- struct pu_contrib

  contributions to p_u More...
- struct interact

  interaction data structure for RNAup More...
- struct pu_out

  Collection of all free_energy of beeing unpaired values for output. More...
- struct constrain

  constraints for cofolding More...

- struct duplexT

  Data structure for RNAduplex. More...
- struct node

  Data structure for RNAsnoop (fold energy list) More...
- struct snoopT

  Data structure for RNAsnoop. More...
- struct dupVar

  Data structure used in RNApkplex. More...
- struct vrna_fc_s

  The most basic data structure required by many functions throughout the RNAlib. More...

Macros

- #define VRNA_STATUS_MFE_PRE (unsigned char)1

  Status message indicating that MFE computations are about to begin.
- #define VRNA_STATUS_MFE_POST (unsigned char)2

  Status message indicating that MFE computations are finished.
- #define VRNA_STATUS_PF_PRE (unsigned char)3

  Status message indicating that Partition function computations are about to begin.
- #define VRNA_STATUS_PF_POST (unsigned char)4

  Status message indicating that Partition function computations are finished.
- #define VRNA_OPTION_DEFAULT 0U

  Option flag to specify default settings/requirements.
- #define VRNA_OPTION_MFE 1U

  Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.
- #define VRNA_OPTION_PF 2U

  Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.
- #define VRNA_OPTION_HYBRID 4U

  Option flag to specify requirement of dimer DP matrices.
- #define VRNA_OPTION_EVAL_ONLY 8U

  Option flag to specify that neither MFE, nor PF DP matrices are required.
- #define VRNA_OPTION_WINDOW 16U

  Option flag to specify requirement of DP matrices for local folding approaches.

Typedefs

- typedef struct vrna_fc_s vrna_fold_compound_t

  Typename for the fold_compound data structure vrna_fc_s.
- typedef struct vrna_basepair_s vrna_basepair_t

  Typename for the base pair repesenting data structure vrna_basepair_s.
- typedef struct vrna_plist_s vrna_plist_t

  Typename for the base pair list repesenting data structure vrna_plist_s.
- typedef struct vrna_bp_stack_s vrna_bp_stack_t

  Typename for the base pair stack repesenting data structure vrna_bp_stack_s.
- typedef struct vrna_cpair_s vrna_cpair_t

  Typename for data structure vrna_cpair_s.
- typedef struct vrna_sect_s vrna_sect_t

  Typename for stack of partial structures vrna_sect_s.

- typedef double FLT_OR_DBL

  Typename for floating point number in partition function computations.
- typedef void(vrna_callback_free_auxdata) (void *data)

  Callback to free memory allocated for auxiliary user-provided data.
- typedef void(vrna_callback_recursion_status) (unsigned char status, void *data)

  Callback to perform specific user-defined actions before, or after recursive computations.
- typedef struct vrna_basepair_s PAIR

  Old typename of vrna_basepair_s.
- typedef struct vrna_plist_s plist

  Old typename of vrna_plist_s.
- typedef struct vrna_cpair_s cpair

  Old typename of vrna_cpair_s.
- typedef struct vrna_sect_s sect

  Old typename of vrna_sect_s.
- typedef struct vrna_bp_stack_s bondT

  Old typename of vrna_bp_stack_s.
- typedef struct pu_contrib pu_contrib

  contributions to p_u
- typedef struct interact interact

  interaction data structure for RNAup
- typedef struct pu_out pu_out

  Collection of all free_energy of beeing unpaired values for output.
- typedef struct constrain constrain

  constraints for cofolding
- typedef struct node folden

  Data structure for RNAsnoop (fold energy list)
- typedef struct dupVar dupVar

  Data structure used in RNApkplex.

Enumerations

Functions

- vrna_fold_compound_t * vrna_fold_compound (const char *sequence, vrna_md_t *md_p, unsigned int options)

  Retrieve a vrna_fold_compound_t data structure for single sequences and hybridizing sequences.
- vrna_fold_compound_t * vrna_fold_compound_comparative (const char **sequences, vrna_md_t *md_p, unsigned int options)

  Retrieve a vrna_fold_compound_t data structure for sequence alignments.
- void vrna_fold_compound_free (vrna_fold_compound_t *vc)

  Free memory occupied by a vrna_fold_compound_t.
- void vrna_fold_compound_add_auxdata (vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *f)

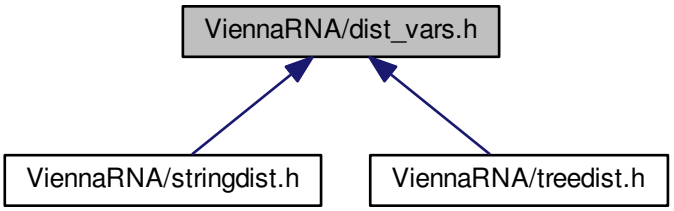
  Add auxiliary data to the vrna_fold_compound_t.
- void vrna_fold_compound_add_callback (vrna_fold_compound_t *vc, vrna_callback_recursion_status *f)

  Add a recursion status callback to the vrna_fold_compound_t.

15.17 ViennaRNA/dist_vars.h File Reference

Global variables for Distance-Package.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct Postorder_list

  Postorder data structure.
- struct Tree

  Tree data structure.
- struct swString

  Some other data structure.

Variables

- int edit_backtrack

  Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.
- char * aligned_line [4]

  Contains the two aligned structures after a call to one of the distance functions with edit_backtrack set to 1.
- int cost_matrix

  Specify the cost matrix to be used for distance calculations.

15.17.1 Detailed Description

Global variables for Distance-Package.

15.17.2 Variable Documentation

15.17.2.1 int edit_backtrack

Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.

set to 1 if you want backtracking

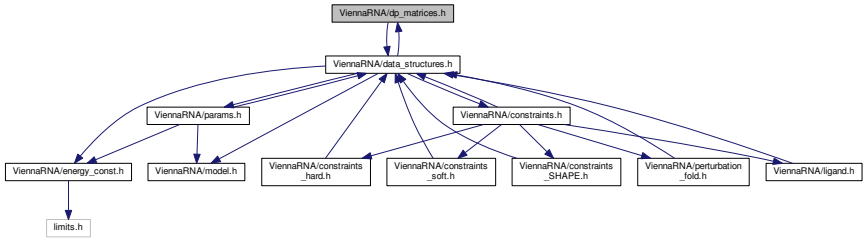
15.17.2.2 int cost_matrix

Specify the cost matrix to be used for distance calculations.

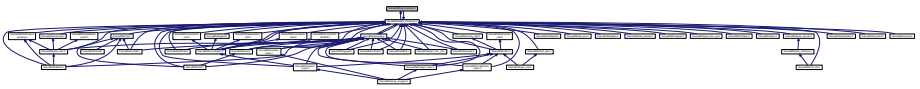
if 0, use the default cost matrix (upper matrix in example), otherwise use Shapiro's costs (lower matrix).

15.18 ViennaRNA/dp_matrices.h File Reference

Include dependency graph for dp_matrices.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_mx_mfe_s

  Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...

- struct vrna_mx_pf_s

  Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...

Typedefs

- typedef struct vrna_mx_mfe_s vrna_mx_mfe_t

  Typename for the Minimum Free Energy (MFE) DP matrices data structure vrna_mx_mfe_s.

- typedef struct vrna_mx_pf_s vrna_mx_pf_t

  Typename for the Partition Function (PF) DP matrices data structure vrna_mx_pf_s.

Enumerations

Functions

- int vrna_mx_add (vrna_fold_compound_t *vc, vrna_mx_type_e type, unsigned int options)

  Add Dynamic Programming (DP) matrices (allocate memory)
- void vrna_mx_mfe_free (vrna_fold_compound_t *vc)

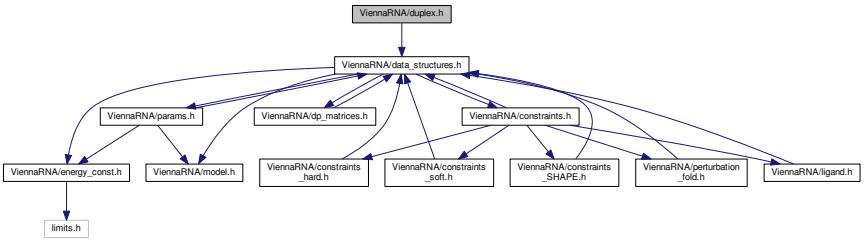
  Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.
- void vrna_mx_pf_free (vrna_fold_compound_t *vc)

  Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

15.19 ViennaRNA/duplex.h File Reference

Duplex folding function declarations...

Include dependency graph for duplex.h:



15.19.1 Detailed Description

Duplex folding function declarations...

15.20 ViennaRNA/edit_cost.h File Reference

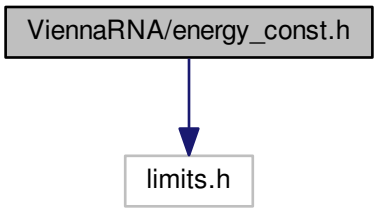
global variables for Edit Costs included by treedist.c and stringdist.c

15.20.1 Detailed Description

global variables for Edit Costs included by treedist.c and stringdist.c

15.21 ViennaRNA/energy_const.h File Reference

Include dependency graph for energy_const.h:



This graph shows which files directly or indirectly include this file:

Macros

- #define GASCONST 1.98717 /∗ in [cal/K] ∗/
- #define K0 273.15
- #define INF 10000000 /∗ (INT_MAX/10) ∗/
- #define FORBIDDEN 9999
- #define BONUS 10000
- #define NBPAIRS 7
- #define TURN 3
- #define MAXLOOP 30

15.21.1 Detailed Description

energy constants

15.21.2 Macro Definition Documentation

15.21.2.1 #define GASCONST 1.98717 /∗ in [cal/K] ∗/

The gas constant

15.21.2.2 #define K0 273.15

0 deg Celsius in Kelvin

15.21.2.3 #define INF 10000000 /∗ (INT_MAX/10) ∗/

Infinity as used in minimization routines

15.21.2.4 #define FORBIDDEN 9999

forbidden

15.21.2.5 #define BONUS 10000

bonus contribution

15.21.2.6 #define NBPAIRS 7

The number of distinguishable base pairs

15.21.2.7 #define TURN 3

The minimum loop length

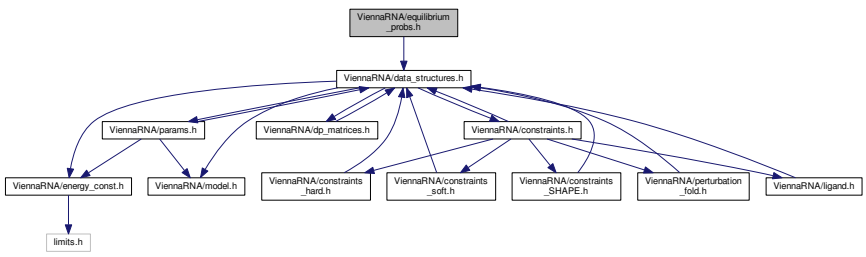
15.21.2.8 #define MAXLOOP 30

The maximum loop length

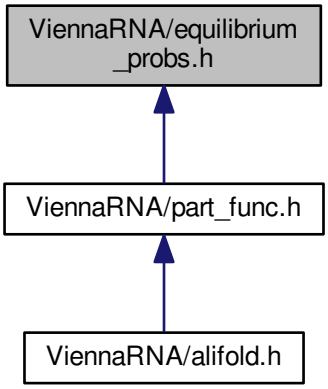
15.22 ViennaRNA/equilibrium_probs.h File Reference

Equilibrium Probability implementations.

Include dependency graph for equilibrium_probs.h:



This graph shows which files directly or indirectly include this file:



Functions

- double vrna_mean_bp_distance_pr (int length, FLT_OR_DBL *pr)

  Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.
- double vrna_mean_bp_distance (vrna_fold_compound_t *vc)

  Get the mean base pair distance in the thermodynamic ensemble.
- vrna_plist_t * vrna_stack_prob (vrna_fold_compound_t *vc, double cutoff)

  Compute stacking probabilities.

15.22.1 Detailed Description

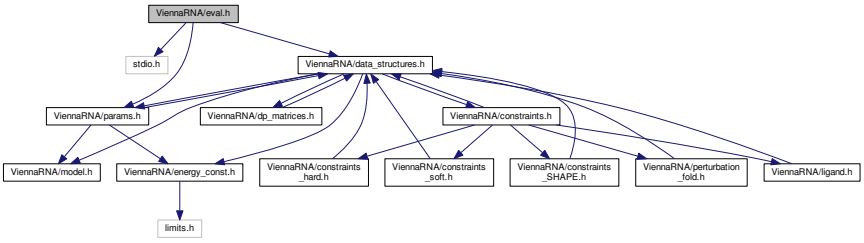
Equilibrium Probability implementations.

This file includes various implementations for equilibrium probability computations based on the partition function of an RNA sequence, two concatenated sequences, or a sequence alignment.

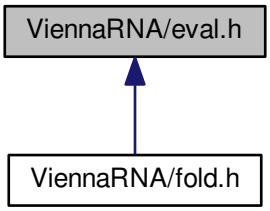
15.23 ViennaRNA/eval.h File Reference

Functions and variables related to energy evaluation of sequence/structure pairs.

Include dependency graph for eval.h:



This graph shows which files directly or indirectly include this file:



- int cut_point

  set to first pos of second seq for cofolding
- int eos_debug

  verbose info from energy_of_struct
- float vrna_eval_structure (vrna_fold_compound_t *vc, const char *structure)

  Calculate the free energy of an already folded RNA.
- float vrna_eval_covar_structure (vrna_fold_compound_t *vc, const char *structure)

  Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.
- float vrna_eval_structure_simple (const char *string, const char *structure)

  Calculate the free energy of an already folded RNA.
- float vrna_eval_structure_verbose (vrna_fold_compound_t *vc, const char *structure, FILE *file)

  Calculate the free energy of an already folded RNA and print contributions on a per-loop base.
- float vrna_eval_structure_v (vrna_fold_compound_t *vc, const char *structure, int verbosity_level, FILE *file)

  Calculate the free energy of an already folded RNA and print contributions on a per-loop base.
- float vrna_eval_structure_simple_verbose (const char *string, const char *structure, FILE *file)

  Calculate the free energy of an already folded RNA and print contributions per loop.
- float vrna_eval_structure_simple_v (const char *string, const char *structure, int verbosity_level, FILE *file)

  Calculate the free energy of an already folded RNA and print contributions per loop.
- int vrna_eval_structure_pt (vrna_fold_compound_t *vc, const short *pt)

  Calculate the free energy of an already folded RNA.
- int vrna_eval_structure_pt_simple (const char *string, const short *pt)

  Calculate the free energy of an already folded RNA.
- int vrna_eval_structure_pt_verbose (vrna_fold_compound_t *vc, const short *pt, FILE *file)

  Calculate the free energy of an already folded RNA.
- int vrna_eval_structure_pt_v (vrna_fold_compound_t *vc, const short *pt, int verbosity_level, FILE *file)

  Calculate the free energy of an already folded RNA.
- int vrna_eval_structure_pt_simple_verbose (const char *string, const short *pt, FILE *file)

  Calculate the free energy of an already folded RNA.
- int vrna_eval_structure_pt_simple_v (const char *string, const short *pt, int verbosity_level, FILE *file)

  Calculate the free energy of an already folded RNA.
- int vrna_eval_loop_pt (vrna_fold_compound_t *vc, int i, const short *pt)

  Calculate energy of a loop.
- float vrna_eval_move (vrna_fold_compound_t *vc, const char *structure, int m1, int m2)

  Calculate energy of a move (closing or opening of a base pair)
- int vrna_eval_move_pt (vrna_fold_compound_t *vc, short *pt, int m1, int m2)

Calculate energy of a move (closing or opening of a base pair)

- float energy_of_structure (const char *string, const char *structure, int verbosity_level)

  Calculate the free energy of an already folded RNA using global model detail settings.
- float energy_of_struct_par (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)

  Calculate the free energy of an already folded RNA.
- float energy_of_circ_structure (const char *string, const char *structure, int verbosity_level)

  Calculate the free energy of an already folded circular RNA.
- float energy_of_circ_struct_par (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)

  Calculate the free energy of an already folded circular RNA.
- int energy_of_structure_pt (const char *string, short *ptable, short *s, short *s1, int verbosity_level)

  Calculate the free energy of an already folded RNA.
- int energy_of_struct_pt_par (const char *string, short *ptable, short *s, short *s1, vrna_param_t *parameters, int verbosity_level)

  Calculate the free energy of an already folded RNA.
- float energy_of_move (const char *string, const char *structure, int m1, int m2)

  Calculate energy of a move (closing or opening of a base pair)
- int energy_of_move_pt (short *pt, short *s, short *s1, int m1, int m2)

  Calculate energy of a move (closing or opening of a base pair)
- int loop_energy (short *ptable, short *s, short *s1, int i)

  Calculate energy of a loop.
- float energy_of_struct (const char *string, const char *structure)
- int energy_of_struct_pt (const char *string, short *ptable, short *s, short *s1)
- float energy_of_circ_struct (const char *string, const char *structure)

15.23.1 Detailed Description

Functions and variables related to energy evaluation of sequence/structure pairs.

15.23.2 Function Documentation

15.23.2.1 float vrna_eval_structure (vrna_fold_compound_t * *vc*, const char * *structure*)

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given pair of structure and sequence (alignment). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not need to contain any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

Note

Accepts vrna_fold_compound_t of type VRNA_VC_TYPE_SINGLE and VRNA_VC_TYPE_ALIGNMENT

See also

vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(), vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_eval_covar_structure()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
structure	Secondary structure in dot-bracket notation

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.2 float vrna_eval_covar_structure (vrna_fold_compound_t * *vc*, const char * *structure*)

Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.

Consensus structure prediction is driven by covariance scores of base pairs in rows of the provided alignment. This function allows to retrieve the total amount of this covariance pseudo energy scores. The vrna_fold_compound_t does not need to contain any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound_comparative(alignment, NULL,
      VRNA_OPTION_EVAL_ONLY);
```

Note

Accepts vrna_fold_compound_t of type VRNA_VC_TYPE_ALIGNMENT only!

See also

vrna_fold_compound_comparative(), vrna_eval_structure()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
structure	Secondary (consensus) structure in dot-bracket notation

Returns

The covariance pseudo energy score of the input structure given the input sequence alignment in kcal/mol

15.23.2.3 float vrna_eval_structure_simple (const char * *string*, const char * *structure*)

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair. In contrast to vrna_eval_structure() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

vrna_eval_structure(), vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_↩
verbose(),

Parameters

string	RNA sequence in uppercase letters
structure	Secondary structure in dot-bracket notation

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.4 float vrna_eval_structure_verbose (vrna_fold_compound_t * *vc*, const char * *structure*, FILE * *file*)

Calculate the free energy of an already folded RNA and print contributions on a per-loop base.

This function is a simplyfied version of vrna_eval_structure_v() that uses the *default* verbosity level. (

See also

vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(),

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
structure	Secondary structure in dot-bracket notation
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.5 float vrna_eval_structure_v (vrna_fold_compound_t * *vc*, const char * *structure*, int *verbosity_level*, FILE * *file*)

Calculate the free energy of an already folded RNA and print contributions on a per-loop base.

This function allows for detailed energy evaluation of a given sequence/structure pair. In contrast to vrna_eval_structure() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluting functions, while values ≥ 1 allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

See also

vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(),

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
structure	Secondary structure in dot-bracket notation
verbosity_level	The level of verbosity of this function
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.6 float vrna_eval_structure_simple_verbose (const char * *string*, const char * *structure*, FILE * *file*)

Calculate the free energy of an already folded RNA and print contributions per loop.

This function is a simplyfied version of vrna_eval_structure_simple_v() that uses the *default* verbosity level.

See also

vrna_eval_structure_simple_v(), vrna_eval_structure_verbose(), vrna_eval_structure_pt(), vrna_eval_↩ structure_verbose(), vrna_eval_structure_pt_verbose()

Parameters

string	RNA sequence in uppercase letters
structure	Secondary structure in dot-bracket notation
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.7 float vrna_eval_structure_simple_v (const char * *string*, const char * *structure*, int *verbosity_level*, FILE * *file*)

Calculate the free energy of an already folded RNA and print contributions per loop.

This function allows for detailed energy evaluation of a given sequence/structure pair. In contrast to vrna_eval_↩ structure() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluting functions, while values ≥ 1 allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

In contrast to vrna_eval_structure_verbose() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Threefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

vrna_eval_structure_verbose(), vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_↩ structure_pt_verbose(),

Parameters

string	RNA sequence in uppercase letters
structure	Secondary structure in dot-bracket notation
verbosity_level	The level of verbosity of this function
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.8 int vrna_eval_structure_pt (vrna_fold_compound_t * *vc*, const short * *pt*)

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from vrna_ptable(). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

See also

vrna_ptable(), vrna_eval_structure(), vrna_eval_structure_pt_verbose()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
pt	Secondary structure as pair_table

Returns

The free energy of the input structure given the input sequence in 10cal/mol

15.23.2.9 int vrna_eval_structure_pt_simple (const char * *string*, const short * *pt*)

Calculate the free energy of an already folded RNA.

In contrast to vrna_eval_structure_pt() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Threefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

vrna_ptable(), vrna_eval_structure_simple(), vrna_eval_structure_pt()

Parameters

string	RNA sequence in uppercase letters
pt	Secondary structure as pair_table

Returns

The free energy of the input structure given the input sequence in 10cal/mol

15.23.2.10 int vrna_eval_structure_pt_verbose (vrna_fold_compound_t * *vc*, const short * *pt*, FILE * *file*)

Calculate the free energy of an already folded RNA.

This function is a simplyfied version of vrna_eval_structure_simple_v() that uses the *default* verbosity level.

See also

vrna_eval_structure_pt_v(), vrna_ptable(), vrna_eval_structure_pt(), vrna_eval_structure_verbose()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
pt	Secondary structure as pair_table
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in 10cal/mol

15.23.2.11 int vrna_eval_structure_pt_v (vrna_fold_compound_t * *vc*, const short * *pt*, int *verbosity_level*, FILE * *file*)

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from vrna_ptable(). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

In contrast to vrna_eval_structure_pt() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluting functions, while values ≥ 1 allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

See also

vrna_ptable(), vrna_eval_structure_pt(), vrna_eval_structure_verbose()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
pt	Secondary structure as pair_table
verbosity_level	The level of verbosity of this function
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in 10cal/mol

15.23.2.12 int vrna_eval_structure_pt_simple_verbose (const char * *string*, const short * *pt*, FILE * *file*)

Calculate the free energy of an already folded RNA.

This function is a simplyfied version of vrna_eval_structure_pt_simple_v() that uses the *default* verbosity level.

See also

vrna_eval_structure_pt_simple_v(), vrna_ptable(), vrna_eval_structure_pt_verbose(), vrna_eval_structure_↩ simple()

Parameters

string	RNA sequence in uppercase letters
pt	Secondary structure as pair_table
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in 10cal/mol

15.23.2.13 int vrna_eval_structure_pt_simple_v (const char * *string*, const short * *pt*, int *verbosity_level*, FILE * *file*)

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from vrna_ptable(). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

In contrast to vrna_eval_structure_pt_verbose() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Threefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

vrna_ptable(), vrna_eval_structure_pt_v(), vrna_eval_structure_simple()

Parameters

string	RNA sequence in uppercase letters
pt	Secondary structure as pair_table
verbosity_level	The level of verbosity of this function
file	A file handle where this function should print to (may be NULL).

Returns

The free energy of the input structure given the input sequence in 10cal/mol

15.23.2.14 int vrna_eval_loop_pt (vrna_fold_compound_t * *vc*, int *i*, const short * *pt*)

Calculate energy of a loop.

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
i	position of covering base pair
pt	the pair table of the secondary structure

Returns

free energy of the loop in 10cal/mol

15.23.2.15 float vrna_eval_move (vrna_fold_compound_t * *vc*, const char * *structure*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

See also

vrna_eval_move_pt()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
structure	secondary structure in dot-bracket notation
m1	first coordinate of base pair
m2	second coordinate of base pair

Returns

energy change of the move in kcal/mol

15.23.2.16 int vrna_eval_move_pt (vrna_fold_compound_t * *vc*, short * *pt*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

See also

vrna_eval_move()

Parameters

vc	A vrna_fold_compound_t containing the energy parameters and model details
pt	the pair table of the secondary structure
m1	first coordinate of base pair
m2	second coordinate of base pair

Returns

energy change of the move in 10cal/mol

15.23.2.17 float energy_of_structure (const char * *string*, const char * *structure*, int *verbosity_level*)

Calculate the free energy of an already folded RNA using global model detail settings.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See energy_of_struct_par() for a completely threadsafe implementation.

Deprecated Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

See also

vrna_eval_structure()

Parameters

string	RNA sequence
structure	secondary structure in dot-bracket notation
verbosity_level	a flag to turn verbose output on/off

Returns

the free energy of the input structure given the input sequence in kcal/mol

15.23.2.18 float energy_of_struct_par (const char * *string*, const char * *structure*, vrna_param_t * *parameters*, int *verbosity_level*)

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Deprecated Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

See also

vrna_eval_structure()

Parameters

string	RNA sequence in uppercase letters
structure	Secondary structure in dot-bracket notation
parameters	A data structure containing the prescaled energy contributions and the model details.
verbosity_level	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.19 float energy_of_circ_structure (const char * *string*, const char * *structure*, int *verbosity_level*)

Calculate the free energy of an already folded circular RNA.

Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See energy_of_circ_struct_par() for a completely threadsafe implementation.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Deprecated Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

See also

vrna_eval_structure()

Parameters

string	RNA sequence
structure	Secondary structure in dot-bracket notation
verbosity_level	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.20 float energy_of_circ_struct_par (const char * *string*, const char * *structure*, vrna_param_t * *parameters*, int *verbosity_level*)

Calculate the free energy of an already folded circular RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Deprecated Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

See also

vrna_eval_structure()

Parameters

string	RNA sequence
structure	Secondary structure in dot-bracket notation
parameters	A data structure containing the prescaled energy contributions and the model details.
verbosity_level	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in kcal/mol

15.23.2.21 int energy_of_structure_pt (const char * *string*, short * *ptable*, short * *s*, short * *s1*, int *verbosity_level*)

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See energy_of_struct_pt_par() for a completely threadsafe implementation.

Deprecated Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

See also

vrna_eval_structure_pt()

Parameters

string	RNA sequence
ptable	the pair table of the secondary structure
s	encoded RNA sequence
s1	encoded RNA sequence
verbosity_level	a flag to turn verbose output on/off

Returns

the free energy of the input structure given the input sequence in 10kcal/mol

15.23.2.22 int energy_of_struct_pt_par (const char * *string*, short * *ptable*, short * *s*, short * *s1*, vrna_param_t * *parameters*, int *verbosity_level*)

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Deprecated Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

See also

vrna_eval_structure_pt()

Parameters

string	RNA sequence in uppercase letters
ptable	The pair table of the secondary structure
s	Encoded RNA sequence
s1	Encoded RNA sequence
parameters	A data structure containing the prescaled energy contributions and the model details.
verbosity_level	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in 10kcal/mol

15.23.2.23 float energy_of_move (const char * *string*, const char * *structure*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

Deprecated Use vrna_eval_move() instead!

See also

vrna_eval_move()

Parameters

string	RNA sequence
structure	secondary structure in dot-bracket notation
m1	first coordinate of base pair
m2	second coordinate of base pair

Returns

energy change of the move in kcal/mol

15.23.2.24 int energy_of_move_pt (short * *pt*, short * *s*, short * *s1*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

Deprecated Use vrna_eval_move_pt() instead!

See also

vrna_eval_move_pt()

Parameters

pt	the pair table of the secondary structure
s	encoded RNA sequence
s1	encoded RNA sequence
m1	first coordinate of base pair
m2	second coordinate of base pair

Returns

energy change of the move in 10cal/mol

15.23.2.25 int loop_energy (short * *ptable*, short * *s*, short * *s1*, int *i*)

Calculate energy of a loop.

Deprecated Use vrna_eval_loop_pt() instead!

See also

vrna_eval_loop_pt()

Parameters

ptable	the pair table of the secondary structure
s	encoded RNA sequence
s1	encoded RNA sequence
i	position of covering base pair

Returns

free energy of the loop in 10cal/mol

15.23.2.26 float energy_of_struct (const char * *string*, const char * *structure*)

Calculate the free energy of an already folded RNA

Note

This function is not entirely threadsafe! Depending on the state of the global variable eos_debug it prints energy information to stdout or not...

Deprecated This function is deprecated and should not be used in future programs! Use energy_of_structure() instead!

See also

energy_of_structure, energy_of_circ_struct(), energy_of_struct_pt()

Parameters

string	RNA sequence
structure	secondary structure in dot-bracket notation

Returns

the free energy of the input structure given the input sequence in kcal/mol

15.23.2.27 int energy_of_struct_pt (const char * *string*, short * *ptable*, short * *s*, short * *s1*)

Calculate the free energy of an already folded RNA

Note

This function is not entirely threadsafe! Depending on the state of the global variable eos_debug it prints energy information to stdout or not...

Deprecated This function is deprecated and should not be used in future programs! Use energy_of_structure_pt() instead!

See also

make_pair_table(), energy_of_structure()

Parameters

string	RNA sequence
ptable	the pair table of the secondary structure
s	encoded RNA sequence
s1	encoded RNA sequence

Returns

the free energy of the input structure given the input sequence in 10kcal/mol

15.23.2.28 float energy_of_circ_struct (const char * *string*, const char * *structure*)

Calculate the free energy of an already folded circular RNA

Note

This function is not entirely threadsafe! Depending on the state of the global variable eos_debug it prints energy information to stdout or not...

Deprecated This function is deprecated and should not be used in future programs Use energy_of_circ_structure() instead!

See also

energy_of_circ_structure(), energy_of_struct(), energy_of_struct_pt()

Parameters

string	RNA sequence
structure	secondary structure in dot-bracket notation

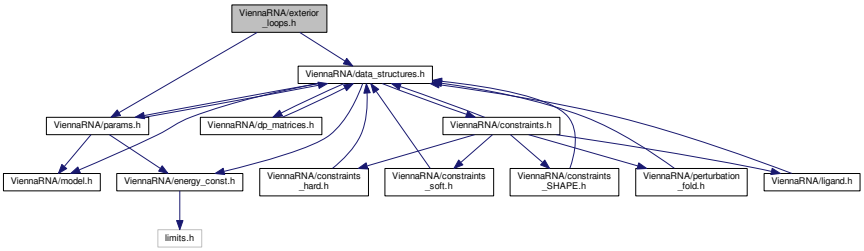
Returns

the free energy of the input structure given the input sequence in kcal/mol

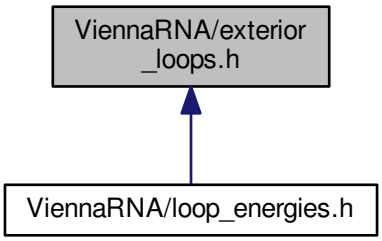
15.24 ViennaRNA/exterior_loops.h File Reference

Energy evaluation of exterior loops for MFE and partition function calculations.

Include dependency graph for exterior_loops.h:



This graph shows which files directly or indirectly include this file:



Functions

- int E_ExtLoop (int type, int si1, int sj1, vrna_param_t ∗P)
- FLT_OR_DBL exp_E_ExtLoop (int type, int si1, int sj1, vrna_exp_param_t ∗P)
- int E_Stem (int type, int si1, int sj1, int extLoop, vrna_param_t ∗P)
- FLT_OR_DBL exp_E_Stem (int type, int si1, int sj1, int extLoop, vrna_exp_param_t ∗P)

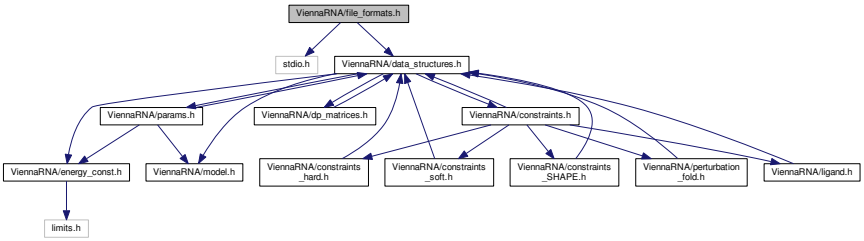
15.24.1 Detailed Description

Energy evaluation of exterior loops for MFE and partition function calculations.

15.25 ViennaRNA/file_formats.h File Reference

Functions dealing with file formats for RNA sequences, structures, and alignments.

Include dependency graph for file_formats.h:



Macros

- #define VRNA_OPTION_MULTILINE 32U

  Tell a function that an input is assumed to span several lines.
- #define VRNA_CONSTRAINT_MULTILINE 32U

  parse multiline constraint

Functions

- void vrna_file_helixlist (const char *seq, const char *db, float energy, FILE *file)

  Print a secondary structure as helix list.
- void vrna_file_connect (const char *seq, const char *db, float energy, const char *identifier, FILE *file)

  Print a secondary structure as connect table.
- void vrna_file_bpseq (const char *seq, const char *db, FILE *file)

  Print a secondary structure in bpseq format.
- void vrna_file_json (const char *seq, const char *db, double energy, const char *identifier, FILE *file)

  Print a secondary structure in jsonformat.
- unsigned int vrna_file_fasta_read_record (char **header, char **sequence, char ***rest, FILE *file, unsigned int options)

  Get a (fasta) data set from a file or stdin.
- char * vrna_extract_record_rest_structure (const char **lines, unsigned int length, unsigned int option)

  Extract a dot-bracket structure string from (multiline)character array.
- int vrna_file_SHAPE_read (const char *file_name, int length, double default_value, char *sequence, double *values)

  Read data from a given SHAPE reactivity input file.
- vrna_plist_t * vrna_file_constraints_read (const char *filename, unsigned int length, unsigned int options)

  Read constraints from an input file.
- void vrna_extract_record_rest_constraint (char **cstruc, const char **lines, unsigned int option)

  Extract a hard constraint encoded as pseudo dot-bracket string.
- unsigned int read_record (char **header, char **sequence, char ***rest, unsigned int options)

  Get a data record from stdin.

15.25.1 Detailed Description

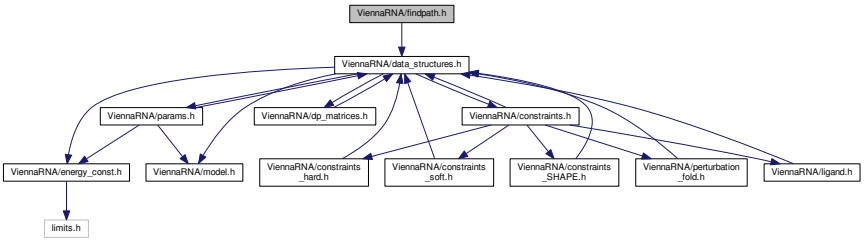
Functions dealing with file formats for RNA sequences, structures, and alignments.

See also

Constraints Definition File

15.26 ViennaRNA/findpath.h File Reference

Include dependency graph for findpath.h:



Data Structures

- struct vrna_path_s

  An element of a refolding path list. More...

Typedefs

- typedef struct vrna_path_s vrna_path_t

  Typename for the refolding path data structure vrna_path_s.
- typedef struct vrna_path_s path_t

  Old typename of vrna_path_s.

Functions

- int vrna_path_findpath_saddle (vrna_fold_compound_t *vc, const char *struc1, const char *struc2, int max)

  Find energy of a saddle point between 2 structures (search only direct path)
- vrna_path_t * vrna_path_findpath (vrna_fold_compound_t *vc, const char *s1, const char *s2, int maxkeep)

  Find refolding path between 2 structures (search only direct path)
- int find_saddle (const char *seq, const char *struc1, const char *struc2, int max)

  Find energy of a saddle point between 2 structures (search only direct path)
- void free_path (vrna_path_t *path)

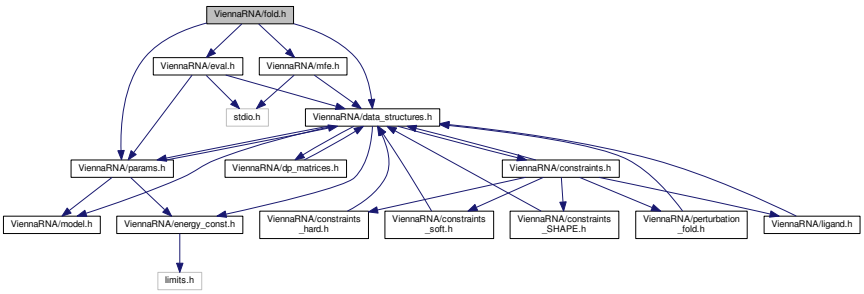
  Free memory allocated by get_path() function.
- vrna_path_t * get_path (const char *seq, const char *s1, const char *s2, int maxkeep)

  Find refolding path between 2 structures (search only direct path)

15.27 ViennaRNA/fold.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for fold.h:



Functions

- float vrna_fold (const char *sequence, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.
- float vrna_circfold (const char *sequence, char *structure)

  Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.
- float fold_par (const char *sequence, char *structure, vrna_param_t *parameters, int is_constrained, int is↩ _circular)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.
- float fold (const char *sequence, char *structure)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.
- float circfold (const char *sequence, char *structure)

  Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.
- void free_arrays (void)

  Free arrays for mfe folding.
- void update_fold_params (void)

  Recalculate energy parameters.
- void update_fold_params_par (vrna_param_t *parameters)

  Recalculate energy parameters.
- void export_fold_arrays (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)
- void export_fold_arrays_par (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p)
- void export_circfold_arrays (int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)
- void export_circfold_arrays_par (int *Fc_p, int *FcH_p, int *FcI_p, int *FcM_p, int **fM2_p, int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p, vrna_param_t **P_p)
- int LoopEnergy (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1)
- int HairpinE (int size, int type, int si1, int sj1, const char *string)
- void initialize_fold (int length)

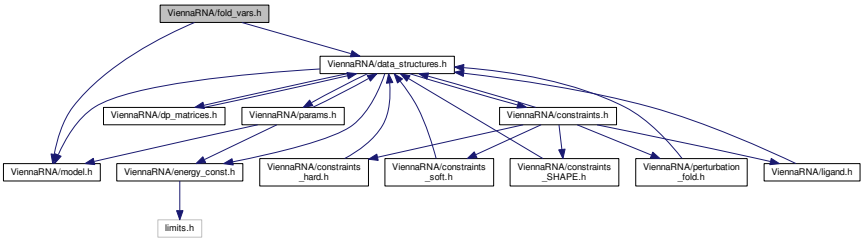
15.27.1 Detailed Description

MFE calculations for single RNA sequences.

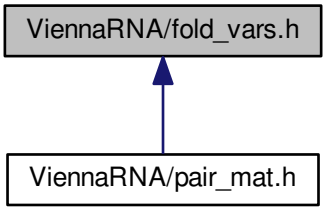
15.28 ViennaRNA/fold_vars.h File Reference

Here all all declarations of the global variables used throughout RNAlib.

Include dependency graph for fold_vars.h:



This graph shows which files directly or indirectly include this file:



Variables

- int fold_constrained

  Global switch to activate/deactivate folding with structure constraints.
- int csv

  generate comma seperated output
- char * RibosumFile
- int james_rule
- int logML
- int cut_point

  Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.
- bondT * base_pair

  Contains a list of base pairs after a call to fold().
- FLT_OR_DBL * pr

  A pointer to the base pair probability matrix.
- int * iindx

  index array to move through pr.

15.28.1 Detailed Description

Here all all declarations of the global variables used throughout RNAlib.

15.28.2 Variable Documentation

15.28.2.1 char* RibosumFile

warning this variable will vanish in the future ribosums will be compiled in instead

15.28.2.2 int james_rule

interior loops of size 2 get energy 0.8Kcal and no mismatches, default 1

15.28.2.3 int logML

use logarithmic multiloop energy function

15.28.2.4 int cut_point

Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.

To evaluate the energy of a duplex structure (a structure formed by two strands), concatenate the to sequences and set it to the first base of the second strand in the concatenated sequence. The default value of -1 stands for single molecule folding. The cut_point variable is also used by vrna_file_PS_rnaplot() and PS_dot_plot() to mark the chain break in postscript plots.

15.28.2.5 bondT* base_pair

Contains a list of base pairs after a call to fold().

base_pair[0].i contains the total number of pairs.

Deprecated Do not use this variable anymore!

15.28.2.6 FLT_OR_DBL* pr

A pointer to the base pair probability matrix.

Deprecated Do not use this variable anymore!

15.28.2.7 int* iindx

index array to move through pr.

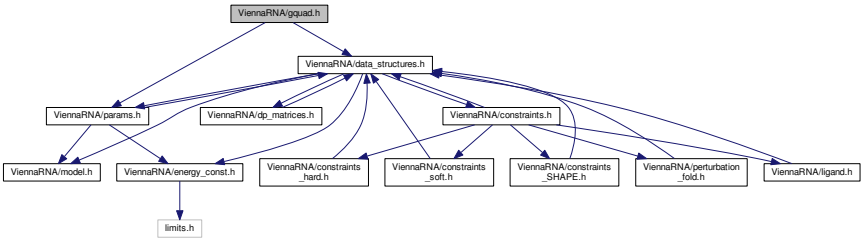
The probability for base i and j to form a pair is in pr[iindx[i]-j].

Deprecated Do not use this variable anymore!

15.29 ViennaRNA/gquad.h File Reference

Various functions related to G-quadruplex computations.

Include dependency graph for gquad.h:



Functions

- int * get_gquad_matrix (short *S, vrna_param_t *P)

  Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.
- int parse_gquad (const char *struc, int *L, int l[3])
- PRIVATE int backtrack_GQuad_IntLoop (int c, int i, int j, int type, short *S, int *ggg, int *index, int *p, int *q, vrna_param_t *P)
- PRIVATE int backtrack_GQuad_IntLoop_L (int c, int i, int j, int type, short *S, int **ggg, int maxdist, int *p, int *q, vrna_param_t *P)

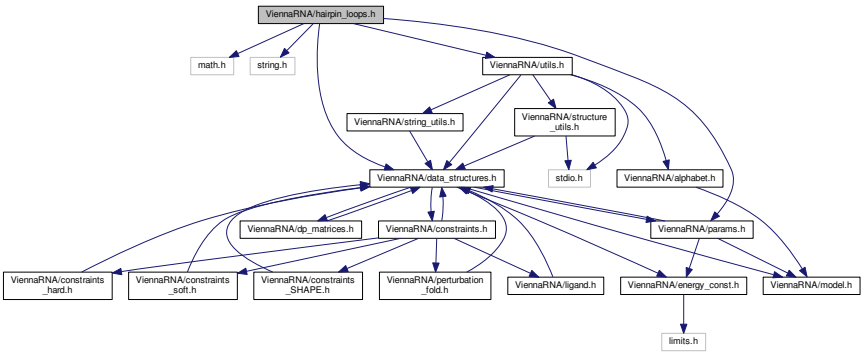
15.29.1 Detailed Description

Various functions related to G-quadruplex computations.

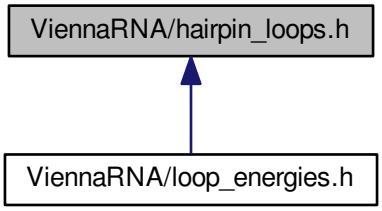
15.30 ViennaRNA/hairpin_loops.h File Reference

Energy evaluation of hairpin loops for MFE and partition function calculations.

Include dependency graph for hairpin_loops.h:



This graph shows which files directly or indirectly include this file:



Functions

- PRIVATE int E_Hairpin (int size, int type, int si1, int sj1, const char *string, vrna_param_t *P)

  Compute the Energy of a hairpin-loop.
- PRIVATE FLT_OR_DBL exp_E_Hairpin (int u, int type, short si1, short sj1, const char *string, vrna_exp_↩ param_t *P)

  Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop.
- int vrna_eval_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate free energy of a hairpin loop.
- int vrna_eval_ext_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate free energy of an exterior hairpin loop.
- int vrna_E_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.
- int vrna_E_ext_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.

- FLT_OR_DBL vrna_exp_E_hp_loop (vrna_fold_compound_t *vc, int i, int j)

  High-Level function for hairpin loop energy evaluation (partition function variant)

- int vrna_BT_hp_loop (vrna_fold_compound_t *vc, int i, int j, int en, vrna_bp_stack_t *bp_stack, int *stack_count)

  Backtrack a hairpin loop closed by (i, j).

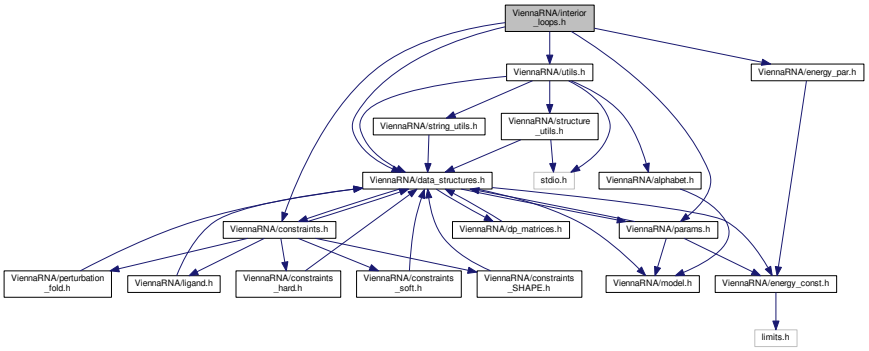
15.30.1 Detailed Description

Energy evaluation of hairpin loops for MFE and partition function calculations.

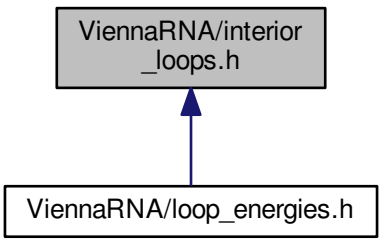
15.31 ViennaRNA/interior_loops.h File Reference

Energy evaluation of interior loops for MFE and partition function calculations.

Include dependency graph for interior_loops.h:



This graph shows which files directly or indirectly include this file:



Functions

- PRIVATE int E_IntLoop (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t *P)
- PRIVATE FLT_OR_DBL exp_E_IntLoop (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, vrna_exp_param_t *P)
- int E_stack (int i, int j, vrna_fold_compound_t *vc)

  Evaluate energy of a base pair stack closed by (i,j)
- int vrna_BT_stack (vrna_fold_compound_t *vc, int *i, int *j, int *en, vrna_bp_stack_t *bp_stack, int *stack_count)

  Backtrack a stacked pair closed by (i, j).
- int vrna_BT_int_loop (vrna_fold_compound_t *vc, int *i, int *j, int en, vrna_bp_stack_t *bp_stack, int *stack_count)

  Backtrack an interior loop closed by (i, j).

15.31.1 Detailed Description

Energy evaluation of interior loops for MFE and partition function calculations.

15.32 ViennaRNA/inverse.h File Reference

Inverse folding routines.

Functions

- float inverse_fold (char *start, const char *target)

  Find sequences with predefined structure.
- float inverse_pf_fold (char *start, const char *target)

  Find sequence that maximizes probability of a predefined structure.

Variables

- char * symbolset

  This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.
- float final_cost
- int give_up
- int inv_verbose

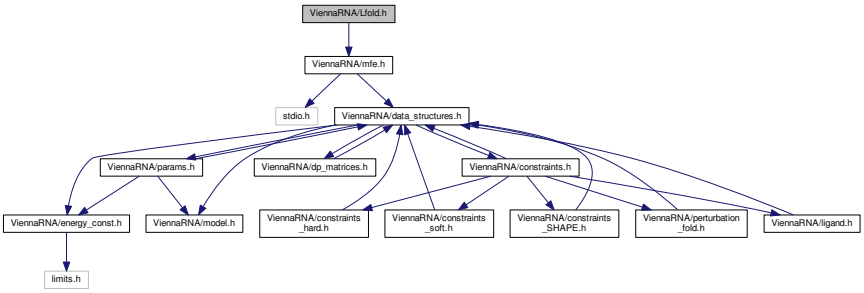
15.32.1 Detailed Description

Inverse folding routines.

15.33 ViennaRNA/Lfold.h File Reference

Predicting local MFE structures of large sequences.

Include dependency graph for Lfold.h:



Functions

- float vrna_Lfold (const char *string, int window_size, FILE *file)

  Local MFE prediction using a sliding window approach (simplified interface)
- float vrna_Lfoldz (const char *string, int window_size, double min_z, FILE *file)

  Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)
- float aliLfold (const char **strings, char *structure, int maxdist)
- float Lfold (const char *string, char *structure, int maxdist)

  The local analog to fold().
- float Lfoldz (const char *string, char *structure, int maxdist, int zsc, double min_z)

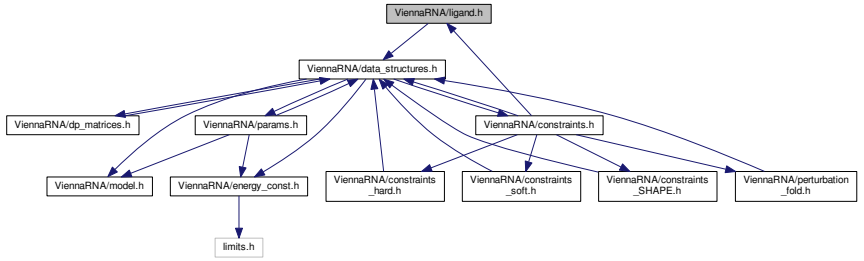
15.33.1 Detailed Description

Predicting local MFE structures of large sequences.

15.34 ViennaRNA/ligand.h File Reference

Functions for incorporation of ligands binding to haipirn and interior loop motifs.

Include dependency graph for ligand.h:



This graph shows which files directly or indirectly include this file:

Functions

- int vrna_sc_add_hi_motif (vrna_fold_compound_t *vc, const char *seq, const char *structure, FLT_OR_DBL energy, unsigned int options)

  Add soft constraints for hairpin or interior loop binding motif.

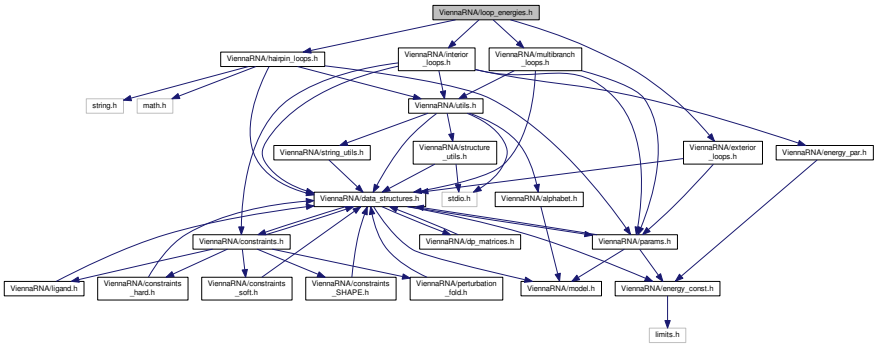
15.34.1 Detailed Description

Functions for incorporation of ligands binding to haipirn and interior loop motifs.

15.35 ViennaRNA/loop_energies.h File Reference

Energy evaluation for MFE and partition function calculations.

Include dependency graph for loop_energies.h:



15.35.1 Detailed Description

Energy evaluation for MFE and partition function calculations.

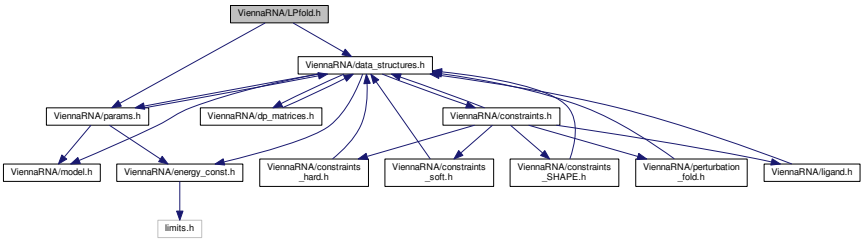
This file contains functions for the calculation of the free energy ΔG of a hairpin- [E_Hairpin()] or interior-loop [E_IntLoop()] .
The unit of the free energy returned is $10^{-2} * \mathrm{kcal/mol}$

In case of computing the partition function, this file also supplies functions which return the Boltzmann weights $e^{-\Delta G/kT}$ for a hairpin- [exp_E_Hairpin()] or interior-loop [exp_E_IntLoop()].

15.36 ViennaRNA/LPfold.h File Reference

Function declarations of partition function variants of the Lfold algorithm.

Include dependency graph for LPfold.h:



Functions

- void update_pf_paramsLP (int length)
- plist * pfl_fold (char *sequence, int winSize, int pairSize, float cutoffb, double **pU, plist **dpp2, FILE *pUfp, FILE *spup)

  Compute partition functions for locally stable secondary structures.
- plist * pfl_fold_par (char *sequence, int winSize, int pairSize, float cutoffb, double **pU, plist **dpp2, FILE *pUfp, FILE *spup, vrna_exp_param_t *parameters)

  Compute partition functions for locally stable secondary structures.
- void putoutpU_prob (double **pU, int length, int ulength, FILE *fp, int energies)

  Writes the unpaired probabilities (pU) or opening energies into a file.
- void putoutpU_prob_bin (double **pU, int length, int ulength, FILE *fp, int energies)

  Writes the unpaired probabilities (pU) or opening energies into a binary file.
- void init_pf_foldLP (int length)

15.36.1 Detailed Description

Function declarations of partition function variants of the Lfold algorithm.

15.36.2 Function Documentation

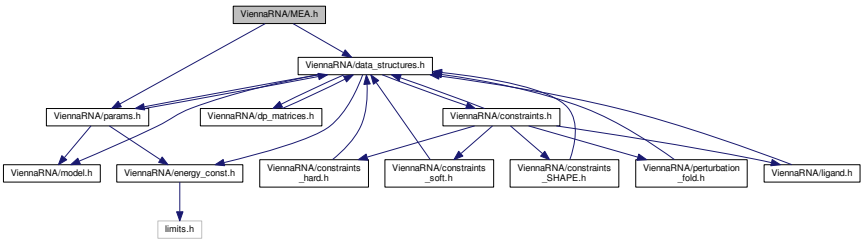
15.36.2.1 void init_pf_foldLP (int *length*)

Dunno if this function was ever used by external programs linking to RNAlib, but it was declared PUBLIC before. Anyway, never use this function as it will be removed soon and does nothing at all

15.37 ViennaRNA/MEA.h File Reference

Computes a MEA (maximum expected accuracy) structure.

Include dependency graph for MEA.h:



Functions

- float MEA (plist *p, char *structure, double gamma)

  Computes a MEA (maximum expected accuracy) structure.

15.37.1 Detailed Description

Computes a MEA (maximum expected accuracy) structure.

15.37.2 Function Documentation

15.37.2.1 float MEA (plist * *p*, char * *structure*, double *gamma*)

Computes a MEA (maximum expected accuracy) structure.

The algorithm maximizes the expected accuracy

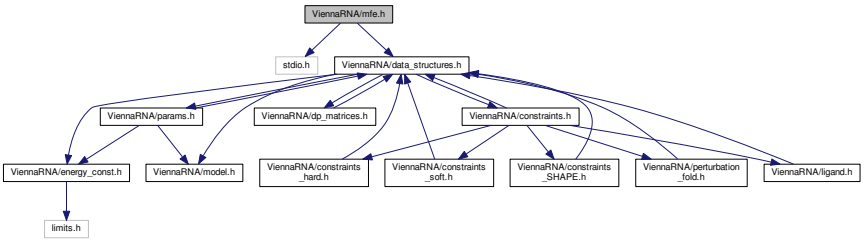
$$A(S) = \sum_{(i,j)\in S} 2\gamma p_{ij} + \sum_{i\notin S} p_i^u$$

Higher values of γ result in more base pairs of lower probability and thus higher sensitivity. Low values of γ result in structures containing only highly likely pairs (high specificity). The code of the MEA function also demonstrates the use of sparse dynamic programming scheme to reduce the time and memory complexity of folding.

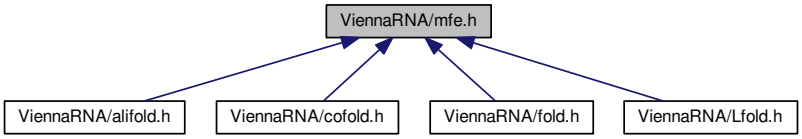
15.38 ViennaRNA/mfe.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for mfe.h:



This graph shows which files directly or indirectly include this file:



Functions

- float vrna_mfe (vrna_fold_compound_t *vc, char *structure)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.
- float vrna_mfe_dimer (vrna_fold_compound_t *vc, char *structure)

  Compute the minimum free energy of two interacting RNA molecules.
- float vrna_mfe_window (vrna_fold_compound_t *vc, FILE *file)

  Local MFE prediction using a sliding window approach.
- float vrna_mfe_window_zscore (vrna_fold_compound_t *vc, double min_z, FILE *file)

  Local MFE prediction using a sliding window approach (with z-score cut-off)

15.38.1 Detailed Description

MFE calculations for single RNA sequences.

This file includes (almost) all function declarations within the RNAlib that are related to MFE folding...

15.39 ViennaRNA/mm.h File Reference

Several Maximum Matching implementations.

15.39.1 Detailed Description

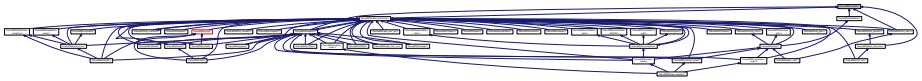
Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

15.40 ViennaRNA/model.h File Reference

The model details data structure and its corresponding modifiers.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_md_s

  The data structure that contains the complete model details used throughout the calculations. More...

Macros

- #define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0

  Default temperature for structure prediction and free energy evaluation in $^\circ C$
- #define VRNA_MODEL_DEFAULT_PF_SCALE -1

  Default scaling factor for partition function computations.
- #define VRNA_MODEL_DEFAULT_BETA_SCALE 1.

  Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.
- #define VRNA_MODEL_DEFAULT_DANGLES 2

  Default dangling end model.
- #define VRNA_MODEL_DEFAULT_SPECIAL_HP 1

  Default model behavior for lookup of special tri-, tetra-, and hexa-loops.
- #define VRNA_MODEL_DEFAULT_NO_LP 0

  Default model behavior for so-called 'lonely pairs'.
- #define VRNA_MODEL_DEFAULT_NO_GU 0

  Default model behavior for G-U base pairs.
- #define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0

  Default model behavior for G-U base pairs closing a loop.
- #define VRNA_MODEL_DEFAULT_CIRC 0

  Default model behavior to treat a molecule as a circular RNA (DNA)
- #define VRNA_MODEL_DEFAULT_GQUAD 0

  Default model behavior regarding the treatment of G-Quadruplexes.
- #define VRNA_MODEL_DEFAULT_UNIQ_ML 0

  Default behavior of the model regarding unique multibranch loop decomposition.
- #define VRNA_MODEL_DEFAULT_ENERGY_SET 0

  Default model behavior on which energy set to use.
- #define VRNA_MODEL_DEFAULT_BACKTRACK 1

Default model behavior with regards to backtracking of structures.

- #define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'

  Default model behavior on what type of backtracking to perform.
- #define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1

  Default model behavior with regards to computing base pair probabilities.
- #define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1

  Default model behavior for the allowed maximum base pair span.
- #define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1

  Default model behavior for the sliding window approach.
- #define VRNA_MODEL_DEFAULT_LOG_ML 0

  Default model behavior on how to evaluate the energy contribution of multibranch loops.
- #define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0

  Default model behavior for consensus structure energy evaluation.
- #define VRNA_MODEL_DEFAULT_ALI_RIBO 0

  Default model behavior for consensus structure covariance contribution assessment.
- #define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.

  Default model behavior for weighting the covariance score in consensus structure prediction.
- #define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

  Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.
- #define MAXALPHA 20

  Maximal length of alphabet.

Typedefs

- typedef struct vrna_md_s vrna_md_t

  Typename for the model details data structure vrna_md_s.

Functions

- void vrna_md_set_default (vrna_md_t *md)

  Apply default model details to a provided vrna_md_t data structure.
- void vrna_md_update (vrna_md_t *md)

  Update the model details data structure.
- char * vrna_md_option_string (vrna_md_t *md)

  Get a corresponding commandline parameter string of the options in a vrna_md_t.
- void vrna_md_defaults_reset (vrna_md_t *md_p)

  Reset the global default model details to a specific set of parameters, or their initial values.
- void vrna_md_defaults_temperature (double T)

  Set default temperature for energy evaluation of loops.
- double vrna_md_defaults_temperature_get (void)

  Get default temperature for energy evaluation of loops.
- void vrna_md_defaults_betaScale (double b)

  Set default scaling factor of thermodynamic temperature in Boltzmann factors.
- double vrna_md_defaults_betaScale_get (void)

  Get default scaling factor of thermodynamic temperature in Boltzmann factors.
- void vrna_md_defaults_dangles (int d)

  Set default dangle model for structure prediction.
- int vrna_md_defaults_dangles_get (void)

  Get default dangle model for structure prediction.

- void vrna_md_defaults_special_hp (int flag)

  Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.
- int vrna_md_defaults_special_hp_get (void)

  Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.
- void vrna_md_defaults_noLP (int flag)

  Set default behavior for prediction of canonical secondary structures.
- int vrna_md_defaults_noLP_get (void)

  Get default behavior for prediction of canonical secondary structures.
- void vrna_md_defaults_noGU (int flag)

  Set default behavior for treatment of G-U wobble pairs.
- int vrna_md_defaults_noGU_get (void)

  Get default behavior for treatment of G-U wobble pairs.
- void vrna_md_defaults_noGUclosure (int flag)

  Set default behavior for G-U pairs as closing pair for loops.
- int vrna_md_defaults_noGUclosure_get (void)

  Get default behavior for G-U pairs as closing pair for loops.
- void vrna_md_defaults_logML (int flag)

  Set default behavior recomputing free energies of multibranch loops using a logarithmic model.
- int vrna_md_defaults_logML_get (void)

  Get default behavior recomputing free energies of multibranch loops using a logarithmic model.
- void vrna_md_defaults_circ (int flag)

  Set default behavior whether input sequences are circularized.
- int vrna_md_defaults_circ_get (void)

  Get default behavior whether input sequences are circularized.
- void vrna_md_defaults_gquad (int flag)

  Set default behavior for treatment of G-Quadruplexes.
- int vrna_md_defaults_gquad_get (void)

  Get default behavior for treatment of G-Quadruplexes.
- void vrna_md_defaults_uniq_ML (int flag)

  Set default behavior for creating additional matrix for unique multibranch loop prediction.
- int vrna_md_defaults_uniq_ML_get (void)

  Get default behavior for creating additional matrix for unique multibranch loop prediction.
- void vrna_md_defaults_energy_set (int e)

  Set default energy set.
- int vrna_md_defaults_energy_set_get (void)

  Get default energy set.
- void vrna_md_defaults_backtrack (int flag)

  Set default behavior for whether to backtrack secondary structures.
- int vrna_md_defaults_backtrack_get (void)

  Get default behavior for whether to backtrack secondary structures.
- void vrna_md_defaults_backtrack_type (char t)

  Set default backtrack type, i.e. which DP matrix is used.
- char vrna_md_defaults_backtrack_type_get (void)

  Get default backtrack type, i.e. which DP matrix is used.
- void vrna_md_defaults_compute_bpp (int flag)

  Set the default behavior for whether to compute base pair probabilities after partition function computation.
- int vrna_md_defaults_compute_bpp_get (void)

  Get the default behavior for whether to compute base pair probabilities after partition function computation.
- void vrna_md_defaults_max_bp_span (int span)

  Set default maximal base pair span.
- int vrna_md_defaults_max_bp_span_get (void)

Get default maximal base pair span.

- void vrna_md_defaults_min_loop_size (int size)

  Set default minimal loop size.
- int vrna_md_defaults_min_loop_size_get (void)

  Get default minimal loop size.
- void vrna_md_defaults_window_size (int size)

  Set default window size for sliding window structure prediction approaches.
- int vrna_md_defaults_window_size_get (void)

  Get default window size for sliding window structure prediction approaches.
- void vrna_md_defaults_oldAliEn (int flag)

  Set default behavior for whether to use old energy model for comparative structure prediction.
- int vrna_md_defaults_oldAliEn_get (void)

  Get default behavior for whether to use old energy model for comparative structure prediction.
- void vrna_md_defaults_ribo (int flag)

  Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.
- int vrna_md_defaults_ribo_get (void)

  Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.
- void vrna_md_defaults_cv_fact (double factor)

  Set the default covariance scaling factor used in comparative structure prediction.
- double vrna_md_defaults_cv_fact_get (void)

  Get the default covariance scaling factor used in comparative structure prediction.
- void vrna_md_defaults_nc_fact (double factor)
- double vrna_md_defaults_nc_fact_get (void)
- void vrna_md_defaults_sfact (double factor)

  Set the default scaling factor used to avoid under-/overflows in partition function computation.
- double vrna_md_defaults_sfact_get (void)

  Get the default scaling factor used to avoid under-/overflows in partition function computation.
- void set_model_details (vrna_md_t *md)

  Set default model details.

Variables

- double temperature

  Rescale energy parameters to a temperature in degC.
- double pf_scale

  A scaling factor used by pf_fold() to avoid overflows.
- int dangles

  Switch the energy model for dangling end contributions (0, 1, 2, 3)
- int tetra_loop

  Include special stabilizing energies for some tri-, tetra- and hexa-loops;.
- int noLonelyPairs

  Global switch to avoid/allow helices of length 1.
- int noGU

  Global switch to forbid/allow GU base pairs at all.
- int no_closingGU

  GU allowed only inside stacks if set to 1.
- int circ

  backward compatibility variable.. this does not effect anything
- int gquad

  Allow G-quadruplex formation.

- int canonicalBPonly
- int uniq_ML

  do ML decomposition uniquely (for subopt)
- int energy_set

  0 = BP; 1=any mit GC; 2=any mit AU-parameter
- int do_backtrack

  do backtracking, i.e. compute secondary structures or base pair probabilities
- char backtrack_type

  A backtrack array marker for inverse_fold()
- char * nonstandards

  contains allowed non standard base pairs
- int max_bp_span

  Maximum allowed base pair span.
- int oldAliEn

  use old alifold energies (with gaps)
- int ribo

  use ribosum matrices
- int logML

  if nonzero use logarithmic ML energy in energy_of_struct

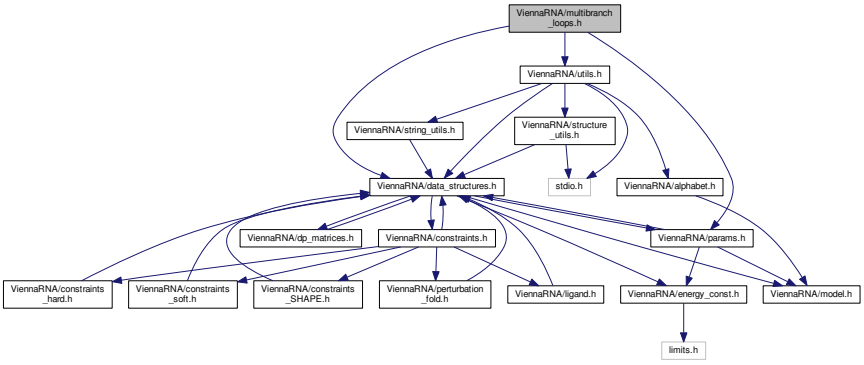
15.40.1 Detailed Description

The model details data structure and its corresponding modifiers.

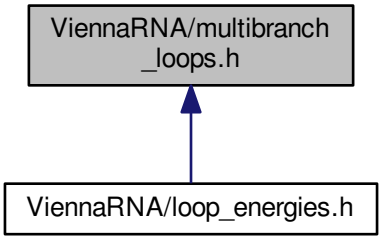
15.41 ViennaRNA/multibranch_loops.h File Reference

Energy evaluation of multibranch loops for MFE and partition function calculations.

Include dependency graph for multibranch_loops.h:



This graph shows which files directly or indirectly include this file:



Functions

- int E_mb_loop_stack (int i, int j, vrna_fold_compound_t ∗vc)

  Evaluate energy of a multi branch helices stacking onto closing pair (i,j)

- int vrna_BT_mb_loop (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int ∗k, int en, int ∗component1, int ∗component2)

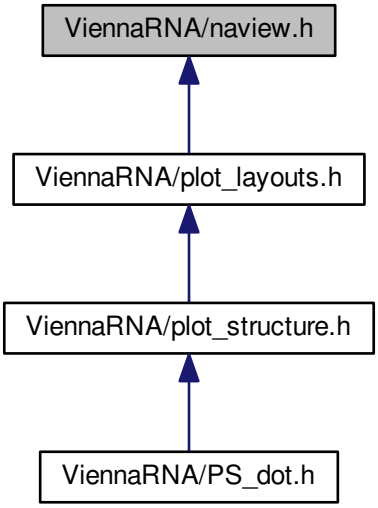
  Backtrack the decomposition of a multi branch loop closed by (i, j).

15.41.1 Detailed Description

Energy evaluation of multibranch loops for MFE and partition function calculations.

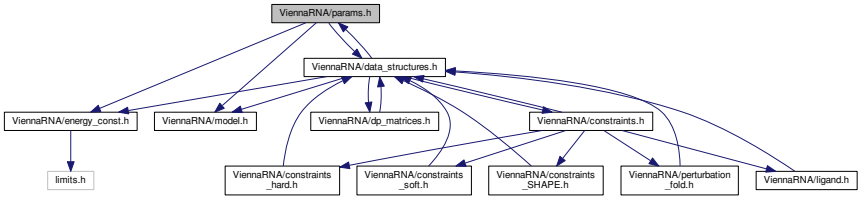
15.42 ViennaRNA/naview.h File Reference

This graph shows which files directly or indirectly include this file:



15.43 ViennaRNA/params.h File Reference

Include dependency graph for params.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct vrna_param_s

  The datastructure that contains temperature scaled energy parameters. More...
- struct vrna_exp_param_s

  The datastructure that contains temperature scaled Boltzmann weights of the energy parameters. More...

Typedefs

- typedef struct vrna_param_s vrna_param_t

  Typename for the free energy parameter data structure vrna_params.
- typedef struct vrna_exp_param_s vrna_exp_param_t

  Typename for the Boltzmann factor data structure vrna_exp_params.
- typedef struct vrna_param_s paramT

  Old typename of vrna_param_s.
- typedef struct vrna_exp_param_s pf_paramT

  Old typename of vrna_exp_param_s.

Functions

- vrna_param_t * vrna_params (vrna_md_t *md)

  Get a data structure containing prescaled free energy parameters.
- vrna_param_t * vrna_params_copy (vrna_param_t *par)

  Get a copy of the provided free energy parameters.
- vrna_exp_param_t * vrna_exp_params (vrna_md_t *md)

  Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.
- vrna_exp_param_t * vrna_exp_params_comparative (unsigned int n_seq, vrna_md_t *md)

  Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)
- vrna_exp_param_t * vrna_exp_params_copy (vrna_exp_param_t *par)

  Get a copy of the provided free energy parameters (provided as Boltzmann factors)
- void vrna_params_subst (vrna_fold_compound_t *vc, vrna_param_t *par)

  Update/Reset energy parameters data structure within a vrna_fold_compound_t.
- void vrna_exp_params_subst (vrna_fold_compound_t *vc, vrna_exp_param_t *params)

  Update the energy parameters for subsequent partition function computations.
- void vrna_exp_params_rescale (vrna_fold_compound_t *vc, double *mfe)

  Rescale Boltzmann factors for partition function computations.
- void vrna_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)

  Reset free energy parameters within a vrna_fold_compound_t according to provided, or default model details.
- void vrna_exp_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)

  Reset Boltzmann factors for partition function computations within a vrna_fold_compound_t according to provided, or default model details.
- vrna_exp_param_t * get_scaled_pf_parameters (void)
- vrna_exp_param_t * get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.
- vrna_exp_param_t * get_boltzmann_factor_copy (vrna_exp_param_t *parameters)

  Get a copy of already precomputed Boltzmann factors.
- vrna_exp_param_t * get_scaled_alipf_parameters (unsigned int n_seq)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)
- vrna_exp_param_t * get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.
- vrna_param_t * scale_parameters (void)

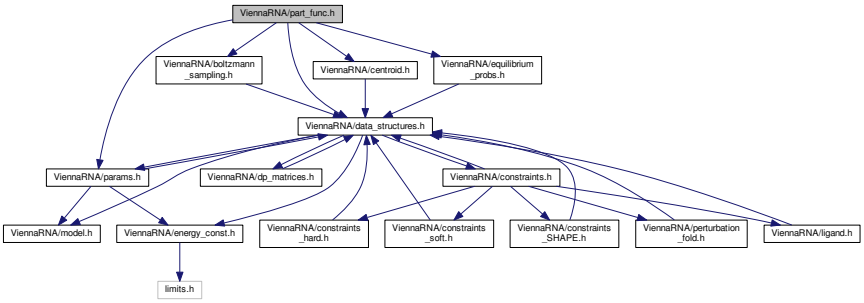
  Get precomputed energy contributions for all the known loop types.
- vrna_param_t * get_scaled_parameters (double temperature, vrna_md_t md)

  Get precomputed energy contributions for all the known loop types.

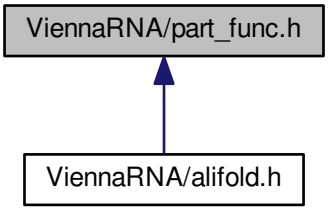
15.44 ViennaRNA/part_func.h File Reference

Partition function of single RNA sequences.

Include dependency graph for part_func.h:



This graph shows which files directly or indirectly include this file:



Functions

- float vrna_pf (vrna_fold_compound_t *vc, char *structure)

  Compute the partition function Q for a given RNA sequence, or sequence alignment.
- float vrna_pf_fold (const char *sequence, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for an RNA sequence using a comparative method.
- float vrna_pf_circfold (const char *sequence, char *structure, vrna_plist_t **pl)

  Compute Partition function Q (and base pair probabilities) for a circular RNA sequences using a comparative method.
- int vrna_pf_float_precision (void)

  Find out whether partition function computations are using single precision floating points.
- float pf_fold_par (const char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_↩ bppm, int is_constrained, int is_circular)

  Compute the partition function Q for a given RNA sequence.
- float pf_fold (const char *sequence, char *structure)

  Compute the partition function Q of an RNA sequence.
- float pf_circ_fold (const char *sequence, char *structure)

Compute the partition function of a circular RNA sequence.

- char * pbacktrack (char *sequence)

  Sample a secondary structure from the Boltzmann ensemble according its probability.

- char * pbacktrack_circ (char *sequence)

  Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

- void free_pf_arrays (void)

  Free arrays for the partition function recursions.

- void update_pf_params (int length)

  Recalculate energy parameters.

- void update_pf_params_par (int length, vrna_exp_param_t *parameters)

  Recalculate energy parameters.

- FLT_OR_DBL * export_bppm (void)

  Get a pointer to the base pair probability array
  Accessing the base pair probabilities for a pair (i,j) is achieved by.

- int get_pf_arrays (short **S_p, short **S1_p, char **ptype_p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm↩ _p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **qln_p)

  Get the pointers to (almost) all relavant computation arrays used in partition function computation.

- double get_subseq_F (int i, int j)

  Get the free energy of a subsequence from the q[] array.

- double mean_bp_distance (int length)

  Get the mean base pair distance of the last partition function computation.

- double mean_bp_distance_pr (int length, FLT_OR_DBL *pr)

  Get the mean base pair distance in the thermodynamic ensemble.

- vrna_plist_t * stackProb (double cutoff)

  Get the probability of stacks.

- void init_pf_fold (int length)

  Allocate space for pf_fold()

- char * centroid (int length, double *dist)
- char * get_centroid_struct_gquad_pr (int length, double *dist)
- double mean_bp_dist (int length)
- double expLoopEnergy (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)
- double expHairpinEnergy (int u, int type, short si1, short sj1, const char *string)

Variables

- int st_back

  Flag indicating that auxilary arrays are needed throughout the computations. This is essential for stochastic backtracking.

15.44.1 Detailed Description

Partition function of single RNA sequences.

This file includes (almost) all function declarations within the **RNAlib** that are related to Partion function folding...

15.44.2 Function Documentation

15.44.2.1 int vrna_pf_float_precision (void)

Find out whether partition function computations are using single precision floating points.

See also

FLT_OR_DBL

Returns

1 if single precision is used, 0 otherwise

15.44.2.2 vrna_plist_t* stackProb (double *cutoff*)

Get the probability of stacks.

Deprecated Use vrna_stack_prob() instead!

15.44.2.3 void init_pf_fold (int *length*)

Allocate space for pf_fold()

Deprecated This function is obsolete and will be removed soon!

15.44.2.4 char* centroid (int *length,* double * *dist*)

Deprecated This function is deprecated and should not be used anymore as it is not threadsafe!

See also

get_centroid_struct_pl(), get_centroid_struct_pr()

15.44.2.5 char* get_centroid_struct_gquad_pr (int *length,* double * *dist*)

Deprecated This function is deprecated and should not be used anymore as it is not threadsafe!

See also

vrna_centroid(), vrna_centroid_from_probs(), vrna_centroid_from_plist()

15.44.2.6 double mean_bp_dist (int *length*)

get the mean pair distance of ensemble

Deprecated This function is not threadsafe and should not be used anymore. Use mean_bp_distance() instead!

15.44.2.7 double expLoopEnergy (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*)

Deprecated Use exp_E_IntLoop() from loop_energies.h instead

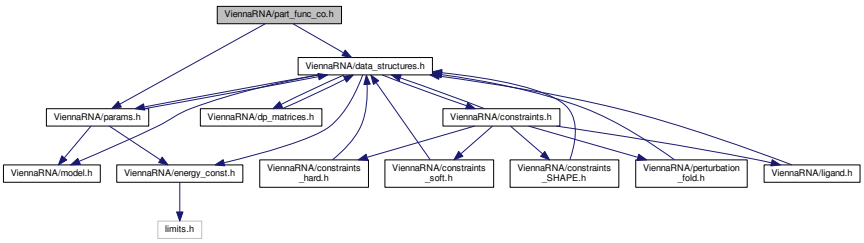
15.44.2.8 double expHairpinEnergy (int *u*, int *type*, short *si1*, short *sj1*, const char * *string*)

Deprecated Use exp_E_Hairpin() from loop_energies.h instead

15.45 ViennaRNA/part_func_co.h File Reference

Partition function for two RNA sequences.

Include dependency graph for part_func_co.h:



Data Structures

- struct vrna_dimer_pf_s

  Data structure returned by vrna_pf_dimer() More...
- struct vrna_dimer_conc_s

  Data structure for concentration dependency computations. More...

Typedefs

- typedef struct vrna_dimer_pf_s vrna_dimer_pf_t

  Typename for the data structure that stores the dimer partition functions, vrna_dimer_pf_s, as returned by vrna_pf_dimer()
- typedef struct vrna_dimer_conc_s vrna_dimer_conc_t

  Typename for the data structure that stores the dimer concentrations, vrna_dimer_conc_s, as required by vrna_pf_dimer_concentration()
- typedef struct vrna_dimer_pf_s cofoldF

  Backward compatibility typedef for vrna_dimer_pf_s.
- typedef struct vrna_dimer_conc_s ConcEnt

  Backward compatibility typedef for vrna_dimer_conc_s.

Functions

- vrna_dimer_pf_t vrna_pf_dimer (vrna_fold_compound_t *vc, char *structure)

  Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.
- void vrna_pf_dimer_probs (double FAB, double FA, double FB, vrna_plist_t *prAB, const vrna_plist_t *prA, const vrna_plist_t *prB, int Alength, const vrna_exp_param_t *exp_params)

  Compute Boltzmann probabilities of dimerization without homodimers.
- vrna_dimer_conc_t * vrna_pf_dimer_concentrations (double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double *startconc, const vrna_exp_param_t *exp_params)

  Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.
- vrna_dimer_pf_t co_pf_fold (char *sequence, char *structure)

  Calculate partition function and base pair probabilities.
- vrna_dimer_pf_t co_pf_fold_par (char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained)

  Calculate partition function and base pair probabilities.
- vrna_plist_t * get_plist (vrna_plist_t *pl, int length, double cut_off)
- void compute_probabilities (double FAB, double FEA, double FEB, vrna_plist_t *prAB, vrna_plist_t *prA, vrna_plist_t *prB, int Alength)

  Compute Boltzmann probabilities of dimerization without homodimers.
- vrna_dimer_conc_t * get_concentrations (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double *startconc)

  Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.
- void init_co_pf_fold (int length)
- FLT_OR_DBL * export_co_bppm (void)

  Get a pointer to the base pair probability array.
- void free_co_pf_arrays (void)

  Free the memory occupied by co_pf_fold()
- void update_co_pf_params (int length)

  Recalculate energy parameters.
- void update_co_pf_params_par (int length, vrna_exp_param_t *parameters)

  Recalculate energy parameters.

Variables

- int mirnatog

  Toggles no intrabp in 2nd mol.
- double F_monomer [2]

  Free energies of the two monomers.

15.45.1 Detailed Description

Partition function for two RNA sequences.

As for folding one RNA molecule, this computes the partition function of all possible structures and the base pair probabilities. Uses the same global pf_scale variable to avoid overflows.

To simplify the implementation the partition function computation is done internally in a null model that does not include the duplex initiation energy, i.e. the entropic penalty for producing a dimer from two monomers). The resulting free energies and pair probabilities are initially relative to that null model. In a second step the free

energies can be corrected to include the dimerization penalty, and the pair probabilities can be divided into the conditional pair probabilities given that a re dimer is formed or not formed.

After computing the partition functions of all possible dimeres one can compute the probabilities of base pairs, the concentrations out of start concentrations and sofar and soaway.

Dimer formation is inherently concentration dependent. Given the free energies of the monomers A and B and dimers AB, AA, and BB one can compute the equilibrium concentrations, given input concentrations of A and B, see e.g. Dimitrov & Zuker (2004)

15.45.2 Function Documentation

15.45.2.1 vrna_dimer_pf_t co_pf_fold (char * *sequence*, char * *structure*)

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the cut_point nucleotide.

Note

OpenMP: Since this function relies on the global parameters do_backtrack, dangles, temperature and pf_↩ scale it is not threadsafe according to concurrent changes in these variables! Use co_pf_fold_par() instead to circumvent this issue.

Deprecated {Use vrna_pf_dimer() instead!}

Parameters

sequence	Concatenated RNA sequences
structure	Will hold the structure or constraints

Returns

vrna_dimer_pf_t structure containing a set of energies needed for concentration computations.

15.45.2.2 vrna_dimer_pf_t co_pf_fold_par (char * *sequence*, char * *structure*, vrna_exp_param_t * *parameters*, int *calculate_bppm*, int *is_constrained*)

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the cut_point nucleotide.

Deprecated Use vrna_pf_dimer() instead!

See also

get_boltzmann_factors(), co_pf_fold()

Parameters

sequence	Concatenated RNA sequences
structure	Pointer to the structure constraint
parameters	Data structure containing the precalculated Boltzmann factors
calculate_bppm	Switch to turn Base pair probability calculations on/off (0==off)
is_constrained	Switch to indicate that a structure contraint is passed via the structure argument (0==off)

Returns

vrna_dimer_pf_t structure containing a set of energies needed for concentration computations.

15.45.2.3 vrna_plist_t* get_plist (vrna_plist_t * *pl*, int *length*, double *cut_off*)

DO NOT USE THIS FUNCTION ANYMORE

Deprecated { This function is deprecated and will be removed soon!} use assign_plist_from_pr() instead!

15.45.2.4 void compute_probabilities (double *FAB*, double *FEA*, double *FEB*, vrna_plist_t * *prAB*, vrna_plist_t * *prA*, vrna_plist_t * *prB*, int *Alength*)

Compute Boltzmann probabilities of dimerization without homodimers.

Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by assign_plist_from_pr(), the dimer probabilities 'prAB' are modified in place.

Deprecated { Use vrna_pf_dimer_probs() instead!}

Parameters

FAB	free energy of dimer AB
FEA	free energy of monomer A
FEB	free energy of monomer B
prAB	pair probabilities for dimer
prA	pair probabilities monomer
prB	pair probabilities monomer
Alength	Length of molecule A

15.45.2.5 vrna_dimer_conc_t* get_concentrations (double *FEAB*, double *FEAA*, double *FEBB*, double *FEA*, double *FEB*, double * *startconc*)

Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

This function takes an array 'startconc' of input concentrations with alternating entries for the initial concentrations of molecules A and B (terminated by two zeroes), then computes the resulting equilibrium concentrations from the free energies for the dimers. Dimer free energies should be the dimer-only free energies, i.e. the FcAB entries from the vrna_dimer_pf_t struct.

Deprecated { Use vrna_pf_dimer_concentrations() instead!}

Parameters

FEAB	Free energy of AB dimer (FcAB entry)
FEAA	Free energy of AA dimer (FcAB entry)
FEBB	Free energy of BB dimer (FcAB entry)
FEA	Free energy of monomer A
FEB	Free energy of monomer B
startconc	List of start concentrations [a0],[b0],[a1],[b1],...,[an][bn],[0],[0]

Returns

vrna_dimer_conc_t array containing the equilibrium energies and start concentrations

15.45.2.6 void init_co_pf_fold (int *length*)

DO NOT USE THIS FUNCTION ANYMORE

Deprecated { This function is deprecated and will be removed soon!}

15.45.2.7 FLT_OR_DBL* export_co_bppm (void)

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

Deprecated This function is deprecated and will be removed soon! The base pair probability array is available through the vrna_fold_compound_t data structure, and its associated vrna_mx_pf_t member.

See also

vrna_idx_row_wise()

Returns

A pointer to the base pair probability array

15.45.2.8 void free_co_pf_arrays (void)

Free the memory occupied by co_pf_fold()

Deprecated This function will be removed for the new API soon! See vrna_pf_dimer(), vrna_fold_compound(), and vrna_fold_compound_free() for an alternative

15.45.2.9 void update_co_pf_params (int *length*)

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings.

Deprecated Use vrna_exp_params_subst() instead!

Parameters

length	Length of the current RNA sequence

15.45.2.10 void update_co_pf_params_par (int *length*, vrna_exp_param_t * *parameters*)

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings. It's second argument can either be NULL or a data structure containing the precomputed Boltzmann factors. In the first scenario, the necessary data structure will be created automatically according to the current global model settings, i.e. this mode might not be threadsafe. However, if the provided data structure is not NULL, threadsafety for the model parameters dangles, pf_scale and temperature is regained, since their values are taken from this data structure during subsequent calculations.

Deprecated Use vrna_exp_params_subst() instead!

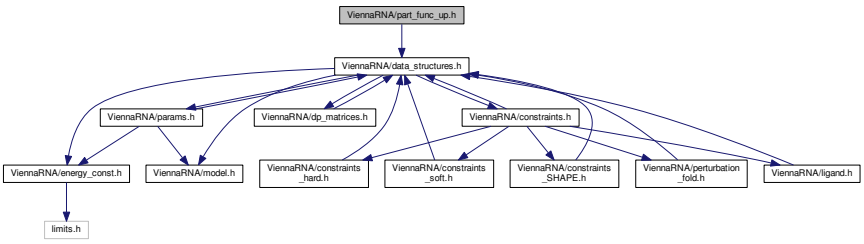
Parameters

length	Length of the current RNA sequence
parameters	data structure containing the precomputed Boltzmann factors

15.46 ViennaRNA/part_func_up.h File Reference

Partition Function Cofolding as stepwise process.

Include dependency graph for part_func_up.h:



Functions

- pu_contrib ∗ pf_unstru (char ∗sequence, int max_w)

  Calculate the partition function over all unpaired regions of a maximal length.
- interact ∗ pf_interact (const char ∗s1, const char ∗s2, pu_contrib ∗p_c, pu_contrib ∗p_c2, int max_w, char ∗cstruc, int incr3, int incr5)

  Calculates the probability of a local interaction between two sequences.
- void free_interact (interact ∗pin)

  Frees the output of function pf_interact().
- void free_pu_contrib_struct (pu_contrib ∗pu)

  Frees the output of function pf_unstru().

15.46.1 Detailed Description

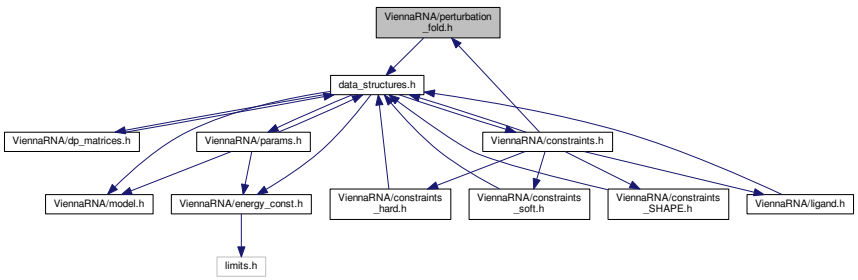
Partition Function Cofolding as stepwise process.

In this approach to cofolding the interaction between two RNA molecules is seen as a stepwise process. In a first step, the target molecule has to adopt a structure in which a binding site is accessible. In a second step, the ligand molecule will hybridize with a region accessible to an interaction. Consequently the algorithm is designed as a two step process: The first step is the calculation of the probability that a region within the target is unpaired, or equivalently, the calculation of the free energy needed to expose a region. In the second step we compute the free energy of an interaction for every possible binding site.

15.47 ViennaRNA/perturbation_fold.h File Reference

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Include dependency graph for perturbation_fold.h:



This graph shows which files directly or indirectly include this file:

Macros

- #define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

  Use the sum of squared aberrations as objective function.
- #define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

  Use the sum of absolute aberrations as objective function.
- #define VRNA_MINIMIZER_DEFAULT 0

  Use a custom implementation of the gradient descent algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_CONJUGATE_FR 1

  Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_CONJUGATE_PR 2

  Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_VECTOR_BFGS 3

  Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_VECTOR_BFGS2 4

  Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.
- #define VRNA_MINIMIZER_STEEPEST_DESCENT 5

  Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.

Typedefs

- typedef void(∗ progress_callback) (int iteration, double score, double ∗epsilon)

  Callback for following the progress of the minimization process.

Functions

- void vrna_sc_minimize_pertubation (vrna_fold_compound_t ∗vc, const double ∗q_prob_unpaired, int objective_function, double sigma_squared, double tau_squared, int algorithm, int sample_size, double ∗epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback)

  Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

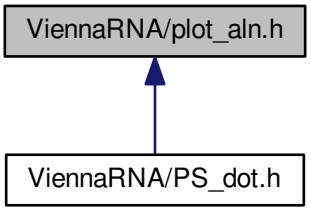
15.47.1 Detailed Description

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

15.48 ViennaRNA/plot_aln.h File Reference

Various functions for plotting Sequence / Structure Alignments.

This graph shows which files directly or indirectly include this file:



Functions

- int PS_color_aln (const char *structure, const char *filename, const char *seqs[ ], const char *names[ ])

  Produce PostScript sequence alignment color-annotated by consensus structure.
- int aliPS_color_aln (const char *structure, const char *filename, const char *seqs[ ], const char *names[ ])

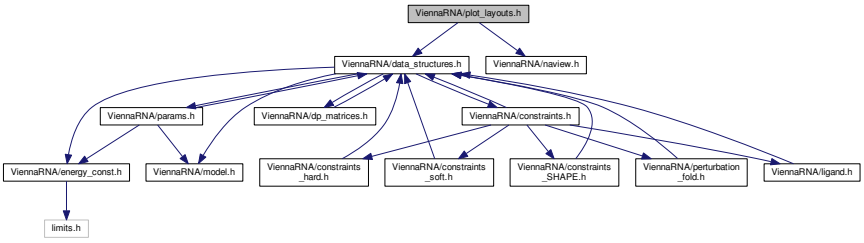
15.48.1 Detailed Description

Various functions for plotting Sequence / Structure Alignments.

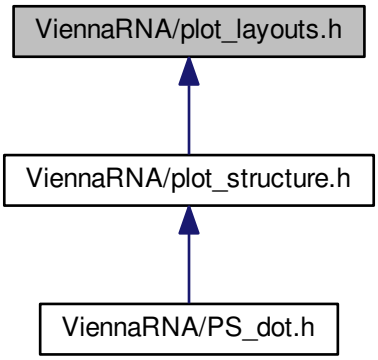
15.49 ViennaRNA/plot_layouts.h File Reference

Secondary structure plot layout algorithms.

Include dependency graph for plot_layouts.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct COORDINATE

  this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE More...

Macros

- #define VRNA_PLOT_TYPE_SIMPLE 0

  Definition of Plot type simple
- #define VRNA_PLOT_TYPE_NAVIEW 1

  Definition of Plot type Naview
- #define VRNA_PLOT_TYPE_CIRCULAR 2

  Definition of Plot type Circular

Functions

- int simple_xy_coordinates (short ∗pair_table, float ∗X, float ∗Y)

  Calculate nucleotide coordinates for secondary structure plot the Simple way
- int simple_circplot_coordinates (short ∗pair_table, float ∗x, float ∗y)

  Calculate nucleotide coordinates for Circular Plot

Variables

- int rna_plot_type

  Switch for changing the secondary structure layout algorithm.

15.49.1 Detailed Description

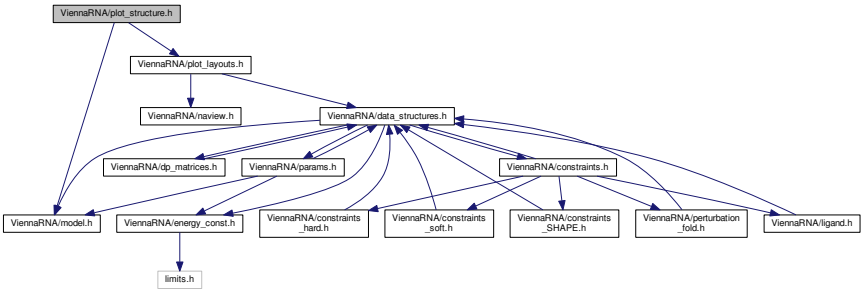
Secondary structure plot layout algorithms.

c Ronny Lorenz The ViennaRNA Package

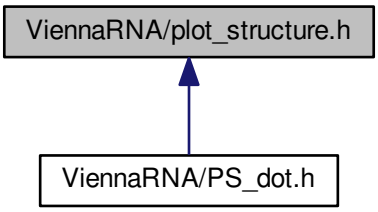
15.50 ViennaRNA/plot_structure.h File Reference

Various functions for plotting RNA secondary structures.

Include dependency graph for plot_structure.h:



This graph shows which files directly or indirectly include this file:



Functions

- int vrna_file_PS_rnaplot (const char *seq, const char *structure, const char *file, vrna_md_t *md_p)

  Produce a secondary structure graph in PostScript and write it to 'filename'.
- int vrna_file_PS_rnaplot_a (const char *seq, const char *structure, const char *file, const char *pre, const char *post, vrna_md_t *md_p)

  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.
- int gmlRNA (char *string, char *structure, char *ssfile, char option)

  Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.
- int ssv_rna_plot (char *string, char *structure, char *ssfile)

Produce a secondary structure graph in SStructView format.

- int svg_rna_plot (char *string, char *structure, char *ssfile)

  Produce a secondary structure plot in SVG format and write it to a file.

- int xrna_plot (char *string, char *structure, char *ssfile)

  Produce a secondary structure plot for further editing in XRNA.

- int PS_rna_plot (char *string, char *structure, char *file)

  Produce a secondary structure graph in PostScript and write it to 'filename'.

- int PS_rna_plot_a (char *string, char *structure, char *file, char *pre, char *post)

  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

- int PS_rna_plot_a_gquad (char *string, char *structure, char *ssfile, char *pre, char *post)

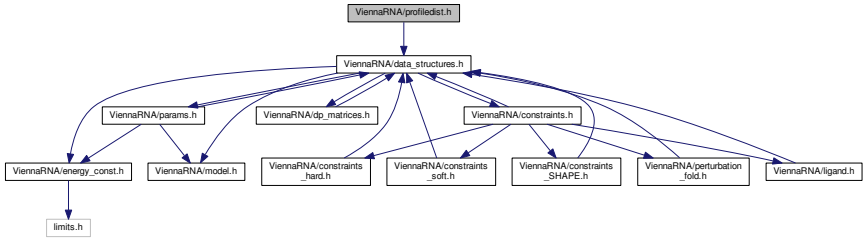
  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)

15.50.1 Detailed Description

Various functions for plotting RNA secondary structures.

15.51 ViennaRNA/profiledist.h File Reference

Include dependency graph for profiledist.h:



Functions

- float profile_edit_distance (const float *T1, const float *T2)

  Align the 2 probability profiles T1, T2

  .

- float * Make_bp_profile_bppm (FLT_OR_DBL *bppm, int length)

  condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

- void print_bppm (const float *T)

  print string representation of probability profile

- void free_profile (float *T)

  free space allocated in Make_bp_profile

- float * Make_bp_profile (int length)

15.51.1 Function Documentation

15.51.1.1 float profile_edit_distance (const float * *T1*, const float * *T2*)

Align the 2 probability profiles T1, T2
.

This is like a Needleman-Wunsch alignment, we should really use affine gap-costs ala Gotoh

15.51.1.2 float* Make_bp_profile_bppm (FLT_OR_DBL * *bppm*, int *length*)

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

This resulting probability profile is used as input for profile_edit_distance

Parameters

bppm	A pointer to the base pair probability matrix
length	The length of the sequence

Returns

The bp profile

15.51.1.3 void free_profile (float * *T*)

free space allocated in Make_bp_profile

Backward compatibility only. You can just use plain free()

15.51.1.4 float* Make_bp_profile (int *length*)

Note

This function is NOT threadsafe

See also

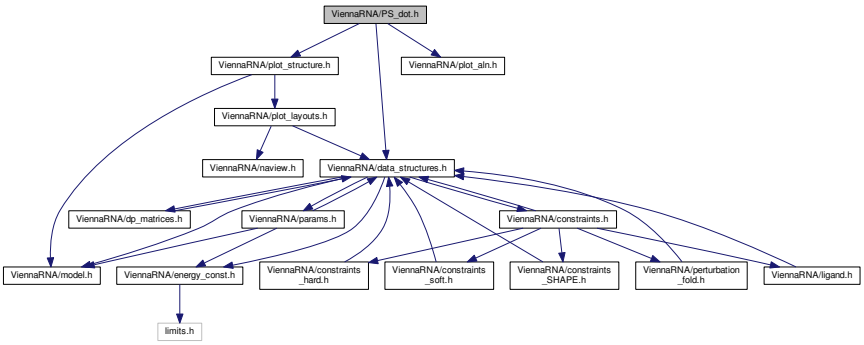
Make_bp_profile_bppm()

Deprecated This function is deprecated and will be removed soon! See Make_bp_profile_bppm() for a replacement

15.52 ViennaRNA/PS_dot.h File Reference

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Include dependency graph for PS_dot.h:



Functions

- int PS_dot_plot_list (char *seq, char *filename, plist *pl, plist *mf, char *comment)

  Produce a postscript dot-plot from two pair lists.
- int PS_dot_plot (char *string, char *file)

  Produce postscript dot-plot.

15.52.1 Detailed Description

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

15.53 ViennaRNA/read_epars.h File Reference

Functions

- void read_parameter_file (const char fname[ ])

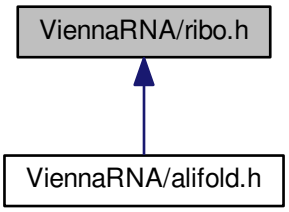
  Read energy parameters from a file.
- void write_parameter_file (const char fname[ ])

  Write energy parameters to a file.

15.54 ViennaRNA/ribo.h File Reference

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

This graph shows which files directly or indirectly include this file:



Functions

- float ** get_ribosum (const char **Alseq, int n_seq, int length)

  Retrieve a RiboSum Scoring Matrix for a given Alignment.
- float ** readribosum (char *name)

  Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.

15.54.1 Detailed Description

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

15.55 ViennaRNA/RNAstruct.h File Reference

Parsing and Coarse Graining of Structures.

Functions

- char * b2HIT (const char *structure)

  Converts the full structure from bracket notation to the HIT notation including root.
- char * b2C (const char *structure)

  Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.
- char * b2Shapiro (const char *structure)

  Converts the full structure from bracket notation to the weighted coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.
- char * add_root (const char *structure)

  Adds a root to an un-rooted tree in any except bracket notation.
- char * expand_Shapiro (const char *coarse)

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

- char * expand_Full (const char *structure)

  Convert the full structure from bracket notation to the expanded notation including root.

- char * unexpand_Full (const char *ffull)

  Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

- char * unweight (const char *wcoarse)

  Strip weights from any weighted tree.

- void unexpand_aligned_F (char *align[2])

  Converts two aligned structures in expanded notation.

- void parse_structure (const char *structure)

  Collects a statistic of structure elements of the full structure in bracket notation.

Variables

- int loop_size [STRUC]

  contains a list of all loop sizes. loop_size[0] contains the number of external bases.

- int helix_size [STRUC]

  contains a list of all stack sizes.

- int loop_degree [STRUC]

  contains the corresponding list of loop degrees.

- int loops

  contains the number of loops (and therefore of stacks).

- int unpaired

  contains the number of unpaired bases.

- int pairs

  contains the number of base pairs in the last parsed structure.

15.55.1 Detailed Description

Parsing and Coarse Graining of Structures.

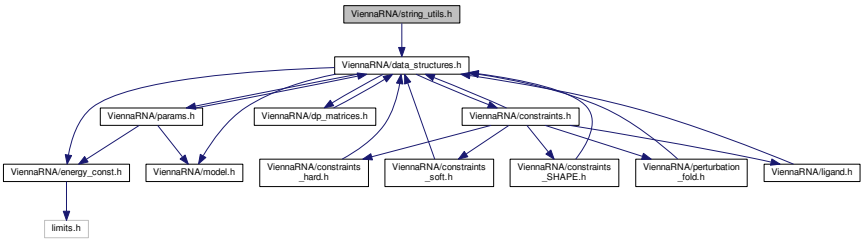
Example:

```
*   .((..(((...)))..((..)))).   is the bracket or full tree
*   becomes expanded:   - expand_Full() -
*   ((U)(((U)(U)((((U)(U)(U)P)P)P)(U)(U)(((U)(U)P)P)P)P)(U)R)
*   HIT:                - b2HIT() -
*   ((U1)((U2)((U3)P3)(U2)((U2)P2)P2)(U1)R)
*   Coarse:             - b2C() -
*   ((H)((H)M)R)
*   becomes expanded:   - expand_Shapiro() -
*   (((((H)S)((H)S)M)S)R)
*   weighted Shapiro:   - b2Shapiro() -
*   ((((((H3)S3)((H2)S2)M4)S2)E2)R)
*
```

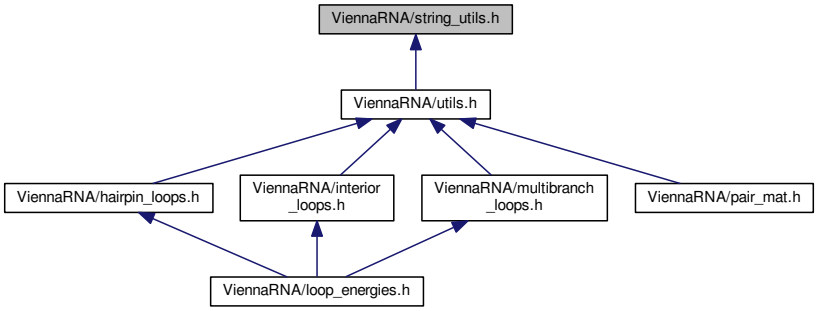
15.56 ViennaRNA/string_utils.h File Reference

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

Include dependency graph for string_utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define XSTR(s) STR(s)

  Stringify a macro after expansion.

- #define STR(s) #s

  Stringify a macro argument.

- #define FILENAME_MAX_LENGTH 80

  Maximum length of filenames that are generated by our programs.

- #define FILENAME_ID_LENGTH 42

  Maximum length of id taken from fasta header for filename generation.

Functions

- char * vrna_random_string (int l, const char symbols[])
  Create a random string using characters from a specified symbol set.
- int vrna_hamming_distance (const char *s1, const char *s2)
  Calculate hamming distance between two sequences.
- int vrna_hamming_distance_bound (const char *s1, const char *s2, int n)
  Calculate hamming distance between two sequences up to a specified length.
- void vrna_seq_toRNA (char *sequence)
  Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.
- void vrna_seq_toupper (char *sequence)
  Convert an input sequence to uppercase.
- char * vrna_cut_point_insert (const char *string, int cp)
  Add a separating '&' character into a string according to cut-point position.
- char * vrna_cut_point_remove (const char *string, int *cp)
  Remove a separating '&' character from a string.
- void str_uppercase (char *sequence)
  Convert an input sequence to uppercase.
- void str_DNA2RNA (char *sequence)
  Convert a DNA input sequence to RNA alphabet.
- char * random_string (int l, const char symbols[])
  Create a random string using characters from a specified symbol set.
- int hamming (const char *s1, const char *s2)
  Calculate hamming distance between two sequences.
- int hamming_bound (const char *s1, const char *s2, int n)
  Calculate hamming distance between two sequences up to a specified length.

15.56.1 Detailed Description

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

15.56.2 Function Documentation

15.56.2.1 void str_uppercase (char * *sequence*)

Convert an input sequence to uppercase.

Deprecated Use vrna_seq_toupper() instead!

15.56.2.2 void str_DNA2RNA (char * *sequence*)

Convert a DNA input sequence to RNA alphabet.

Deprecated Use vrna_seq_toRNA() instead!

15.56.2.3 char* random_string (int *l*, const char *symbols[ ]*)

Create a random string using characters from a specified symbol set.

Deprecated Use vrna_random_string() instead!

15.56.2.4 int hamming (const char * *s1*, const char * *s2*)

Calculate hamming distance between two sequences.

Deprecated Use vrna_hamming_distance() instead!

15.56.2.5 int hamming_bound (const char * *s1*, const char * *s2*, int *n*)

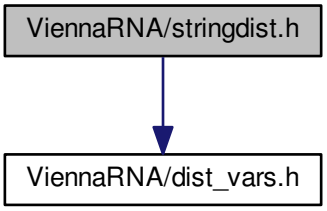
Calculate hamming distance between two sequences up to a specified length.

Deprecated Use vrna_hamming_distance_bound() instead!

15.57 ViennaRNA/stringdist.h File Reference

Functions for String Alignment.

Include dependency graph for stringdist.h:



Functions

- swString * Make_swString (char *string)

  Convert a structure into a format suitable for string_edit_distance().
- float string_edit_distance (swString *T1, swString *T2)

  Calculate the string edit distance of T1 and T2.

15.57.1 Detailed Description

Functions for String Alignment.

15.57.2 Function Documentation

15.57.2.1 swString* Make_swString (char * *string*)

Convert a structure into a format suitable for string_edit_distance().

Parameters

string	

Returns

15.57.2.2 float string_edit_distance (swString * *T1*, swString * *T2*)

Calculate the string edit distance of T1 and T2.

Parameters

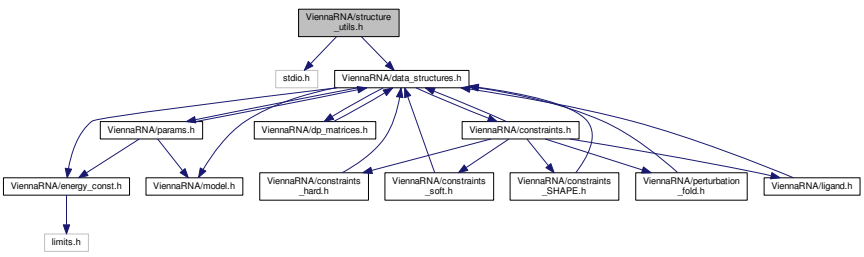
T1	
T2	

Returns

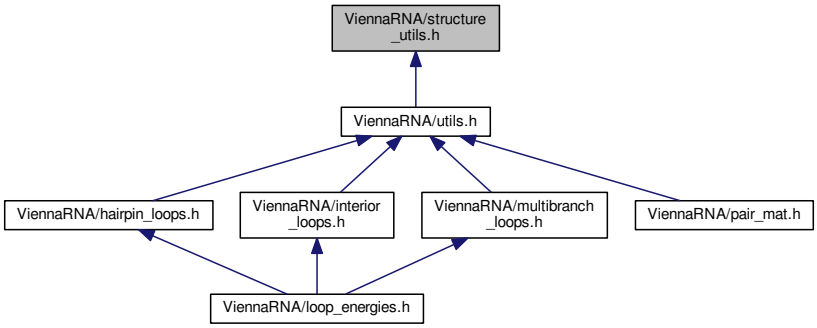
15.58 ViennaRNA/structure_utils.h File Reference

Various utility- and helper-functions for secondary structure parsing, converting, etc.

Include dependency graph for structure_utils.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct vrna_hx_s

  Data structure representing an entry of a helix list. More...

Typedefs

- typedef struct vrna_hx_s vrna_hx_t

  Convenience typedef for data structure vrna_hx_s.

Functions

- char * vrna_db_pack (const char *struc)

  Pack secondary secondary structure, 5:1 compression using base 3 encoding.
- char * vrna_db_unpack (const char *packed)

  Unpack secondary structure previously packed with vrna_db_pack()
- short * vrna_ptable (const char *structure)

  Create a pair table of a secondary structure.
- short * vrna_pt_pk_get (const char *structure)

  Create a pair table of a secondary structure (pseudo-knot version)
- short * vrna_ptable_copy (const short *pt)

  Get an exact copy of a pair table.
- short * vrna_pt_ali_get (const char *structure)

  Create a pair table of a secondary structure (snoop align version)
- short * vrna_pt_snoop_get (const char *structure)

  Create a pair table of a secondary structure (snoop version)
- int * vrna_loopidx_from_ptable (const short *pt)

  Get a loop index representation of a structure.
- char * vrna_db_from_ptable (short *pt)

  Convert a pair table into dot-parenthesis notation.
- int vrna_bp_distance (const char *str1, const char *str2)

  Compute the "base pair" distance between two secondary structures s1 and s2.
- unsigned int * vrna_refBPcnt_matrix (const short *reference_pt, unsigned int turn)

Make a reference base pair count matrix.

- unsigned int * vrna_refBPdist_matrix (const short *pt1, const short *pt2, unsigned int turn)

  Make a reference base pair distance matrix.
- char * vrna_db_from_probs (const FLT_OR_DBL *pr, unsigned int length)

  Create a dot-bracket like structure string from base pair probability matrix.
- char vrna_bpp_symbol (const float *x)

  Get a pseudo dot bracket notation for a given probability information.
- char * vrna_db_from_bp_stack (vrna_bp_stack_t *bp, unsigned int length)

  Create a dot-backet/parenthesis structure from backtracking stack.
- vrna_plist_t * vrna_plist (const char *struc, float pr)

  Create a vrna_plist_t from a dot-bracket string.
- vrna_plist_t * vrna_plist_from_probs (vrna_fold_compound_t *vc, double cut_off)

  Create a vrna_plist_t from base pair probability matrix.
- char * vrna_db_from_plist (vrna_plist_t *pairs, unsigned int n)

  Convert a list of base pairs into dot-bracket notation.
- void assign_plist_from_db (vrna_plist_t **pl, const char *struc, float pr)

  Create a vrna_plist_t from a dot-bracket string.
- char * pack_structure (const char *struc)

  Pack secondary secondary structure, 5:1 compression using base 3 encoding.
- char * unpack_structure (const char *packed)

  Unpack secondary structure previously packed with pack_structure()
- short * make_pair_table (const char *structure)

  Create a pair table of a secondary structure.
- short * copy_pair_table (const short *pt)

  Get an exact copy of a pair table.
- short * alimake_pair_table (const char *structure)
- short * make_pair_table_snoop (const char *structure)
- int bp_distance (const char *str1, const char *str2)

  Compute the "base pair" distance between two secondary structures s1 and s2.
- unsigned int * make_referenceBP_array (short *reference_pt, unsigned int turn)

  Make a reference base pair count matrix.
- unsigned int * compute_BPdifferences (short *pt1, short *pt2, unsigned int turn)

  Make a reference base pair distance matrix.
- void assign_plist_from_pr (vrna_plist_t **pl, FLT_OR_DBL *probs, int length, double cutoff)

  Create a vrna_plist_t from a probability matrix.
- void parenthesis_structure (char *structure, vrna_bp_stack_t *bp, int length)

  Create a dot-backet/parenthesis structure from backtracking stack.
- void parenthesis_zuker (char *structure, vrna_bp_stack_t *bp, int length)

  Create a dot-backet/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.
- void bppm_to_structure (char *structure, FLT_OR_DBL *pr, unsigned int length)

  Create a dot-bracket like structure string from base pair probability matrix.
- char bppm_symbol (const float *x)

  Get a pseudo dot bracket notation for a given probability information.

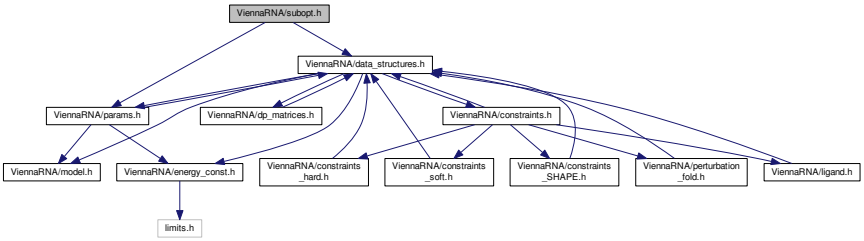
15.58.1 Detailed Description

Various utility- and helper-functions for secondary structure parsing, converting, etc.

15.59 ViennaRNA/subopt.h File Reference

RNAsubopt and density of states declarations.

Include dependency graph for subopt.h:



Data Structures

- struct vrna_subopt_sol_s

  Solution element from subopt.c.

Macros

- #define MAXDOS 1000

  Maximum density of states discretization for subopt.

Typedefs

- typedef struct vrna_subopt_sol_s vrna_subopt_solution_t

  Typename for the subopt solution list repesenting data structure vrna_subopt_sol_s.
- typedef struct vrna_subopt_sol_s SOLUTION

  Backward compatibility typedef for vrna_subopt_sol_s.

Functions

- vrna_subopt_solution_t * vrna_subopt (vrna_fold_compound_t *vc, int delta, int sorted, FILE *fp)

  Returns list of subopt structures or writes to fp.
- vrna_subopt_solution_t * vrna_subopt_zuker (vrna_fold_compound_t *vc)

  Compute Zuker type suboptimal structures.
- SOLUTION * subopt (char *seq, char *structure, int delta, FILE *fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION * subopt_par (char *seq, char *structure, vrna_param_t *parameters, int delta, int is_↩ constrained, int is_circular, FILE *fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION * subopt_circ (char *seq, char *sequence, int delta, FILE *fp)

  Returns list of circular subopt structures or writes to fp.
- SOLUTION * zukersubopt (const char *string)

  Compute Zuker type suboptimal structures.
- SOLUTION * zukersubopt_par (const char *string, vrna_param_t *parameters)

  Compute Zuker type suboptimal structures.

Variables

- double print_energy

  printing threshold for use with logML
- int subopt_sorted

  Sort output by energy.
- int density_of_states [MAXDOS+1]

  The Density of States.

15.59.1 Detailed Description

RNAsubopt and density of states declarations.

15.59.2 Typedef Documentation

15.59.2.1 typedef struct vrna_subopt_sol_s SOLUTION

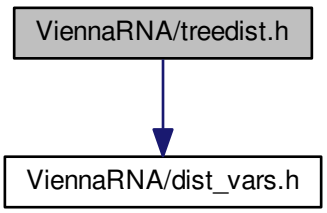
Backward compatibility typedef for vrna_subopt_sol_s.

Deprecated Use vrna_subopt_solution_t instead!

15.60 ViennaRNA/treedist.h File Reference

Functions for Tree Edit Distances.

Include dependency graph for treedist.h:



Functions

- Tree * make_tree (char *struc)

  Constructs a Tree (essentially the postorder list) of the structure 'struc', for use in tree_edit_distance().
- float tree_edit_distance (Tree *T1, Tree *T2)

  Calculates the edit distance of the two trees.
- void print_tree (Tree *t)

  Print a tree (mainly for debugging)
- void free_tree (Tree *t)

  Free the memory allocated for Tree t.

15.60.1 Detailed Description

Functions for Tree Edit Distances.

15.60.2 Function Documentation

15.60.2.1 Tree* make_tree (char * *struc*)

Constructs a Tree (essentially the postorder list) of the structure 'struc', for use in tree_edit_distance().

Parameters

struc	may be any rooted structure representation.

Returns

15.60.2.2 float tree_edit_distance (Tree * *T1*, Tree * *T2*)

Calculates the edit distance of the two trees.

Parameters

T1	
T2	

Returns

15.60.2.3 void free_tree (Tree * *t*)

Free the memory allocated for Tree t.

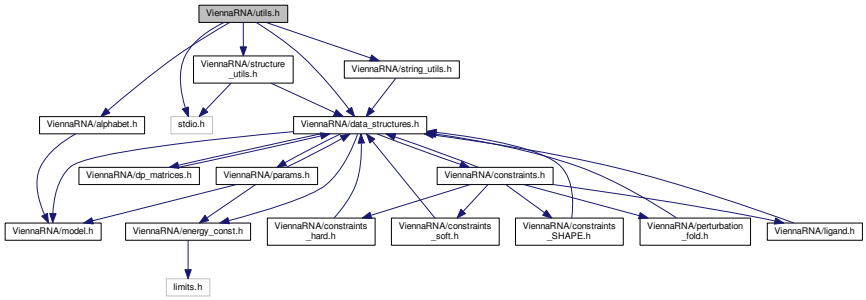
Parameters

t	

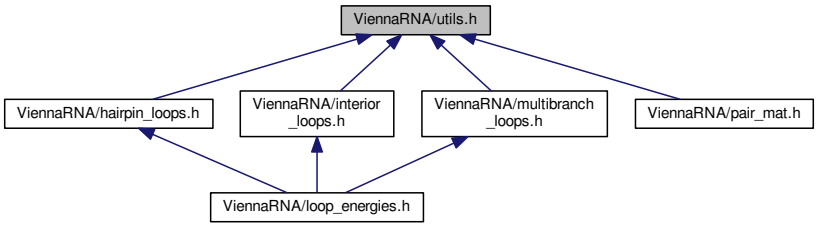
15.61 ViennaRNA/utils.h File Reference

General utility- and helper-functions used throughout the *ViennaRNA Package*.

Include dependency graph for utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VRNA_INPUT_ERROR 1U

  Output flag of get_input_line(): "An ERROR has occured, maybe EOF".
- #define VRNA_INPUT_QUIT 2U

  Output flag of get_input_line(): "the user requested quitting the program".
- #define VRNA_INPUT_MISC 4U

  Output flag of get_input_line(): "something was read".
- #define VRNA_INPUT_FASTA_HEADER 8U

  Input/Output flag of get_input_line():
  if used as input option this tells get_input_line() that the data to be read should comply with the FASTA format.
- #define VRNA_INPUT_CONSTRAINT 32U

  Input flag for get_input_line():
  Tell get_input_line() that we assume to read a structure constraint.
- #define VRNA_INPUT_NO_TRUNCATION 256U

  Input switch for get_input_line(): "do not trunkate the line by eliminating white spaces at end of line".
- #define VRNA_INPUT_NO_REST 512U

  Input switch for vrna_file_fasta_read_record(): "do fill rest array".
- #define VRNA_INPUT_NO_SPAN 1024U

  Input switch for vrna_file_fasta_read_record(): "never allow data to span more than one line".
- #define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U

  Input switch for vrna_file_fasta_read_record(): "do not skip empty lines".
- #define VRNA_INPUT_BLANK_LINE 4096U

Output flag for vrna_file_fasta_read_record(): "read an empty line".

- #define VRNA_INPUT_NOSKIP_COMMENTS 128U

  Input switch for get_input_line(): "do not skip comment lines".

- #define VRNA_INPUT_COMMENT 8192U

  Output flag for vrna_file_fasta_read_record(): "read a comment".

- #define MIN2(A, B) ((A) < (B) ? (A) : (B))

  Get the minimum of two comparable values.

- #define MAX2(A, B) ((A) > (B) ? (A) : (B))

  Get the maximum of two comparable values.

- #define MIN3(A, B, C) (MIN2((MIN2((A),(B))) ,(C)))

  Get the minimum of three comparable values.

- #define MAX3(A, B, C) (MAX2((MAX2((A),(B))) ,(C)))

  Get the maximum of three comparable values.

Functions

- void * vrna_alloc (unsigned size)

  Allocate space safely.

- void * vrna_realloc (void *p, unsigned size)

  Reallocate space safely.

- void vrna_message_error (const char message[])

  Die with an error message.

- void vrna_message_warning (const char message[])

  Print a warning message.

- void vrna_init_rand (void)

  Initialize seed for random number generator.

- double vrna_urn (void)

  get a random number from [0..1]

- int vrna_int_urn (int from, int to)

  Generates a pseudo random integer in a specified range.

- void vrna_file_copy (FILE *from, FILE *to)

  Inefficient 'cp'.

- char * vrna_time_stamp (void)

  Get a timestamp.

- char * get_line (FILE *fp)

  Read a line of arbitrary length from a stream.

- unsigned int get_input_line (char **string, unsigned int options)

- void vrna_message_input_seq_simple (void)

  Print a line to stdout that asks for an input sequence.

- void vrna_message_input_seq (const char *s)

  Print a line with a user defined string and a ruler to stdout.

- int * vrna_idx_row_wise (unsigned int length)

  Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.

- int * vrna_idx_col_wise (unsigned int length)

  Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.

- void print_tty_input_seq (void)

  Print a line to stdout that asks for an input sequence.

- void print_tty_input_seq_str (const char *s)

  Print a line with a user defined string and a ruler to stdout.

- void warn_user (const char message[])

Print a warning message.

- void nrerror (const char message[])

  Die with an error message.

- void * space (unsigned size)

  Allocate space safely.

- void * xrealloc (void *p, unsigned size)

  Reallocate space safely.

- void init_rand (void)

  Make random number seeds.

- double urn (void)

  get a random number from [0..1]

- int int_urn (int from, int to)

  Generates a pseudo random integer in a specified range.

- void filecopy (FILE *from, FILE *to)

  Inefficient `cp`

- char * time_stamp (void)

  Get a timestamp.

Variables

- unsigned short xsubi [3]

  Current 48 bit random number.

15.61.1 Detailed Description

General utility- and helper-functions used throughout the *ViennaRNA Package*.

15.61.2 Function Documentation

15.61.2.1 void print_tty_input_seq (void)

Print a line to *stdout* that asks for an input sequence.

There will also be a ruler (scale line) printed that helps orientation of the sequence positions

Deprecated Use vrna_message_input_seq_simple() instead!

15.61.2.2 void print_tty_input_seq_str (const char * *s*)

Print a line with a user defined string and a ruler to stdout.

(usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

Deprecated Use vrna_message_input_seq() instead!

15.61.2.3 void warn_user (const char *message[ ]*)

Print a warning message.

Print a warning message to *stderr*

Deprecated Use vrna_message_warning() instead!

15.61.2.4 void nrerror (const char *message[ ]*)

Die with an error message.

Deprecated Use vrna_message_error() instead!

15.61.2.5 void* space (unsigned *size*)

Allocate space safely.

Deprecated Use vrna_alloc() instead!

15.61.2.6 void* xrealloc (void * *p,* unsigned *size*)

Reallocate space safely.

Deprecated Use vrna_realloc() instead!

15.61.2.7 void init_rand (void)

Make random number seeds.

Deprecated Use vrna_init_rand() instead!

15.61.2.8 double urn (void)

get a random number from [0..1]

Deprecated Use vrna_urn() instead!

15.61.2.9 int int_urn (int *from*, int *to*)

Generates a pseudo random integer in a specified range.

Deprecated Use vrna_int_urn() instead!

15.61.2.10 void filecopy (FILE * *from*, FILE * *to*)

Inefficient `cp`

Deprecated Use vrna_file_copy() instead!

15.61.2.11 char* time_stamp (void)

Get a timestamp.

Deprecated Use vrna_time_stamp() instead!

Bibliography

[1] S.H. Bernhart, I.L. Hofacker, S. Will, A.R. Gruber, and P.F. Stadler. RNAalifold: Improved consensus structure prediction for RNA alignments. *BMC bioinformatics*, 9(1):474, 2008. 180

[2] S.H. Bernhart, H. Tafer, U. Mückstein, C. Flamm, P.F. Stadler, and I.L. Hofacker. Partition function and base pairing probabilities of RNA heterodimers. *Algorithms for Molecular Biology*, 1(1):3, 2006. 173

[3] R.E. Bruccoleri and G. Heinrich. An improved algorithm for nucleic acid secondary structure display. *Computer applications in the biosciences: CABIOS*, 4(1):167–173, 1988. 318

[4] Katherine E. Deigan, Tian W. Li, David H. Mathews, and Kevin M. Weeks. Accurate SHAPE-directed RNA structure determination. *PNAS*, 106:97–102, 2009. 246

[5] Christoph Flamm, Ivo L Hofacker, Sebastian Maurer-Stroh, Peter F Stadler, and Martin Zehl. Design of multistable RNA molecules. *RNA*, 7(02):254–265, 2001. 278, 279

[6] W. Fontana, P.F. Stadler, E.G. Bornberg-Bauer, T. Griesmacher, I.L. Hofacker, M. Tacker, P. Tarazona, E.D. Weinberger, and P. Schuster. RNA folding and combinatory landscapes. *Physical review E*, 47(3):2083, 1993. 3

[7] I.L. Hofacker, M. Fekete, and P.F. Stadler. Secondary structure prediction for aligned RNA sequences. *Journal of molecular biology*, 319(5):1059–1066, 2002. 180

[8] I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994. 2

[9] I.L. Hofacker and P.F. Stadler. Memory efficient folding algorithms for circular RNA secondary structures. *Bioinformatics*, 22(10):1172–1176, 2006. 133, 139, 161, 187, 191

[10] Ronny Lorenz, Stephan H. Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011. 2

[11] Ronny Lorenz, Christoph Flamm, and Ivo L. Hofacker. 2d projections of RNA folding landscapes. In Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, and Peter F. Stadler, editors, *German Conference on Bioinformatics 2009*, volume 157 of *Lecture Notes in Informatics*, pages 11–20, Bonn, September 2009. Gesellschaft f. Informatik. 213, 215

[12] D.H. Mathews, M.D. Disney, J.L. Childs, S.J. Schroeder, M. Zuker, and D.H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287, 2004. 205

[13] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. 137

[14] B.A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer applications in the biosciences: CABIOS*, 4(3):387–393, 1988. 3

[15] B.A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer applications in the biosciences: CABIOS*, 6(4):309–318, 1990. 5

[16] D.H. Turner and D.H. Mathews. NNDB: The nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Research*, 38(suppl 1):D280–D282, 2010. 205

[17] Stefan Washietl, Ivo L. Hofacker, Peter F. Stadler, and Manolis Kellis. RNA folding with soft constraints: reconciliation of probing data and thermodynamics secondary structure prediction. *Nucleic Acids Research*, 40(10):4261–4272, 2012. 254

[18] S. Wuchty, W. Fontana, I. L. Hofacker, and P. Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49(2):145–165, February 1999. 153

[19] Kourosh Zarringhalam, Michelle M. Meyer, Ivan Dotu, Jeffrey H. Chuang, and Peter Clote. Integrating chemical footprinting data into RNA secondary structure prediction. *PLOS ONE*, 7(10), 2012. 247

[20] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244(4900):48–52, April 1989. 151

[21] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981. 133, 160

Index

_struct_en, 319
2Dpfold.h
    destroy_TwoDpfold_variables, 326
    get_TwoDpfold_variables, 326
    TwoDpfold_pbacktrack, 327
    TwoDpfold_pbacktrack5, 328
    TwoDpfoldList, 326

add_root
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 288
aliLfold
    Local MFE consensus structures for Sequence Alignments, 204
aliPS_color_aln
    Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 313
alifold
    MFE Consensus Structures for Sequence Alignment(s), 187
alimake_pair_table
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 298
alipbacktrack
    Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), 194
alipf_circ_fold
    Partition Function and Base Pair Probabilities for Sequence Alignment(s), 192
alipf_fold
    Partition Function and Base Pair Probabilities for Sequence Alignment(s), 192
alipf_fold_par
    Partition Function and Base Pair Probabilities for Sequence Alignment(s), 192
alloc_sequence_arrays
    Predicting Consensus Structures from Alignment(s), 183
alpha
    vrna_exp_param_s, 65
alphabet.h
    vrna_nucleotide_decode, 333
    vrna_nucleotide_encode, 332
    vrna_ptypes, 332
assign_plist_from_db
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 295
assign_plist_from_pr
    Computing Partition Functions and Pair Probabilities, 146
auxdata
    vrna_fc_s, 260

b2HIT
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 287
b2Shapiro
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 288
b2C
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 288
BONUS
    energy_const.h, 353
backtrack_GQuad_IntLoop
    Processing and Evaluating Decomposed Loops, 55
backtrack_GQuad_IntLoop_L
    Processing and Evaluating Decomposed Loops, 55
backtrack_type
    Manipulation of the Prediction Models, 104
base_pair
    fold_vars.h, 377
bondT
    Data Structures and Preprocessor Macros, 124
bp_distance
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 298
bppm_symbol
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 300
bppm_to_structure
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 299
bt
    vrna_sc_s, 239

COORDINATE, 312
Calculate Partition Functions of a Distance Based Partitioning, 221
    vrna_pf_TwoD, 222
    vrna_sol_TwoD_pf_t, 222
Calculate Secondary Structures of two RNAs upon Dimerization, 159
Calculating MFE representatives of a Distance Based Partitioning, 214
    destroy_TwoDfold_variables, 219
    get_TwoDfold_variables, 218
    TwoDfold_backtrack_f5, 220
    TwoDfold_vars, 217
    TwoDfoldList, 219

vrna_backtrack5_TwoD, 218
vrna_mfe_TwoD, 217
vrna_sol_TwoD_t, 217
canonicalBPonly
Manipulation of the Prediction Models, 104
centroid
part_func.h, 397
centroid.h
get_centroid_struct_pl, 336
get_centroid_struct_pr, 336
circalifold
MFE Consensus Structures for Sequence Alignment(s), 188
circfold
MFE Structures of single Nucleic Acid Sequences, 163
Classified Dynamic Programming, 212
co_pf_fold
part_func_co.h, 400
co_pf_fold_par
part_func_co.h, 400
cofold
MFE Structures of two hybridized Sequences, 167
cofold_par
MFE Structures of two hybridized Sequences, 168
Compute the centroid structure, 148
vrna_centroid, 148
vrna_centroid_from_plist, 148
vrna_centroid_from_probs, 149
Compute the Density of States, 226
density_of_states, 226
Compute the structure with maximum expected accuracy (MEA), 147
compute_BPdifferences
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 299
compute_probabilities
part_func_co.h, 401
Computing Minimum Free Energy (MFE) Structures, 133
vrna_mfe, 134
Computing Partition Functions and Pair Probabilities, 135
assign_plist_from_pr, 146
export_bppm, 143
free_pf_arrays, 142
get_pf_arrays, 144
mean_bp_distance, 145
mean_bp_distance_pr, 145
pf_circ_fold, 142
pf_fold, 141
pf_fold_par, 140
update_pf_params, 143
update_pf_params_par, 143
vrna_mean_bp_distance, 137
vrna_mean_bp_distance_pr, 137
vrna_pf, 138
vrna_pf_circfold, 139
vrna_pf_fold, 138
vrna_plist_from_probs, 145
vrna_stack_prob, 137
cons_seq
vrna_fc_s, 262
constrain, 122
constrain_ptypes
constraints_hard.h, 342
Constraining the Secondary Structure Predictions and Evaluations, 105
VRNA_CONSTRAINT_FILE, 109
VRNA_CONSTRAINT_SOFT_MFE, 109
VRNA_CONSTRAINT_SOFT_PF, 109
VRNA_DECOMP_EXT_EXT_EXT, 115
VRNA_DECOMP_EXT_EXT_STEM1, 116
VRNA_DECOMP_EXT_EXT_STEM, 115
VRNA_DECOMP_EXT_EXT, 114
VRNA_DECOMP_EXT_STEM_EXT, 115
VRNA_DECOMP_EXT_STEM, 114
VRNA_DECOMP_EXT_UP, 114
VRNA_DECOMP_ML_COAXIAL, 113
VRNA_DECOMP_ML_ML_ML, 111
VRNA_DECOMP_ML_ML_STEM, 113
VRNA_DECOMP_ML_ML, 112
VRNA_DECOMP_ML_STEM, 111
VRNA_DECOMP_ML_UP, 112
VRNA_DECOMP_PAIR_HP, 109
VRNA_DECOMP_PAIR_IL, 110
VRNA_DECOMP_PAIR_ML, 110
vrna_constraints_add, 116
vrna_message_constraint_options, 117
vrna_message_constraint_options_all, 118
constraints_SHAPE.h
vrna_sc_SHAPE_parse_method, 343
constraints_hard.h
constrain_ptypes, 342
print_tty_constraint, 342
print_tty_constraint_full, 342
VRNA_CONSTRAINT_DB_ANG_BRACK, 341
VRNA_CONSTRAINT_NO_HEADER, 341
convert_parameter_file
Converting Energy Parameter Files, 211
Converting Energy Parameter Files, 207
convert_parameter_file, 211
VRNA_CONVERT_OUTPUT_ALL, 208
VRNA_CONVERT_OUTPUT_BULGE, 209
VRNA_CONVERT_OUTPUT_DANGLE3, 209
VRNA_CONVERT_OUTPUT_DANGLE5, 209
VRNA_CONVERT_OUTPUT_DUMP, 210
VRNA_CONVERT_OUTPUT_HP, 208
VRNA_CONVERT_OUTPUT_INT_11, 209
VRNA_CONVERT_OUTPUT_INT_21, 209
VRNA_CONVERT_OUTPUT_INT_22, 209
VRNA_CONVERT_OUTPUT_INT, 210
VRNA_CONVERT_OUTPUT_MISC, 210
VRNA_CONVERT_OUTPUT_MM_EXT, 209
VRNA_CONVERT_OUTPUT_MM_HP, 208
VRNA_CONVERT_OUTPUT_MM_INT_1N, 208
VRNA_CONVERT_OUTPUT_MM_INT_23, 208

VRNA_CONVERT_OUTPUT_MM_INT, 208
VRNA_CONVERT_OUTPUT_MM_MULTI, 209
VRNA_CONVERT_OUTPUT_ML, 210
VRNA_CONVERT_OUTPUT_NINIO, 210
VRNA_CONVERT_OUTPUT_SPECIAL_HP, 210
VRNA_CONVERT_OUTPUT_STACK, 208
VRNA_CONVERT_OUTPUT_VANILLA, 210
copy_pair_table
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 296
cost_matrix
dist_vars.h, 349
cpair
Data Structures and Preprocessor Macros, 123
cut_point
fold_vars.h, 377
cv_fact
Predicting Consensus Structures from Alignment(s), 184

dangles
Manipulation of the Prediction Models, 103
vrna_md_s, 81
Data Structures and Preprocessor Macros, 119
bondT, 124
cpair, 123
PAIR, 123
plist, 123
sect, 124
density_of_states
Compute the Density of States, 226
destroy_TwoDfold_variables
Calculating MFE representatives of a Distance Based Partitioning, 219
destroy_TwoDpfold_variables
2Dpfold.h, 326
Direct refolding paths between two secondary structures, 277
find_saddle, 279
free_path, 280
get_path, 280
path_t, 278
vrna_path_findpath, 279
vrna_path_findpath_saddle, 278
dist_vars.h
cost_matrix, 349
edit_backtrack, 349
Distance based partitioning of the Secondary Structure Space, 213
do_backtrack
Manipulation of the Prediction Models, 104
dupVar, 123
duplexT, 122

E_ExtLoop
Processing and Evaluating Decomposed Loops, 52
E_Hairpin
Processing and Evaluating Decomposed Loops, 56
E_IntLoop
Processing and Evaluating Decomposed Loops, 58
E_Stem
Processing and Evaluating Decomposed Loops, 53
E_mb_loop_stack
Processing and Evaluating Decomposed Loops, 60
edit_backtrack
dist_vars.h, 349
encode_ali_sequence
Predicting Consensus Structures from Alignment(s), 183
Energy Parameter Sets and Boltzmann Factors, 62
get_boltzmann_factor_copy, 72
get_boltzmann_factors, 71
get_boltzmann_factors_ali, 72
get_scaled_alipf_parameters, 72
get_scaled_parameters, 73
get_scaled_pf_parameters, 71
paramT, 65
pf_paramT, 65
scale_parameters, 72
vrna_exp_params, 66
vrna_exp_params_comparative, 67
vrna_exp_params_copy, 67
vrna_exp_params_rescale, 68
vrna_exp_params_reset, 70
vrna_exp_params_subst, 68
vrna_params, 65
vrna_params_copy, 66
vrna_params_reset, 69
vrna_params_subst, 68
energy_const.h
BONUS, 353
FORBIDDEN, 353
GASCONST, 352
INF, 352
K0, 352
MAXLOOP, 353
NBPAIRS, 353
TURN, 353
energy_of_alistruct
Predicting Consensus Structures from Alignment(s), 180
energy_of_circ_struct
eval.h, 371
energy_of_circ_struct_par
eval.h, 367
energy_of_circ_structure
eval.h, 366
energy_of_move
eval.h, 368
energy_of_move_pt
eval.h, 369
energy_of_struct
eval.h, 370
energy_of_struct_par
eval.h, 365
energy_of_struct_pt
eval.h, 370

energy_of_struct_pt_par
eval.h, 368
energy_of_structure
eval.h, 365
energy_of_structure_pt
eval.h, 367
energy_set
Manipulation of the Prediction Models, 104
Enumerating Suboptimal Structures, 150
eval.h
energy_of_circ_struct, 371
energy_of_circ_struct_par, 367
energy_of_circ_structure, 366
energy_of_move, 368
energy_of_move_pt, 369
energy_of_struct, 370
energy_of_struct_par, 365
energy_of_struct_pt, 370
energy_of_struct_pt_par, 368
energy_of_structure, 365
energy_of_structure_pt, 367
loop_energy, 369
vrna_eval_covar_structure, 357
vrna_eval_loop_pt, 364
vrna_eval_move, 364
vrna_eval_move_pt, 364
vrna_eval_structure, 356
vrna_eval_structure_pt, 361
vrna_eval_structure_pt_simple, 361
vrna_eval_structure_pt_simple_v, 363
vrna_eval_structure_pt_simple_verbose, 363
vrna_eval_structure_pt_v, 362
vrna_eval_structure_pt_verbose, 362
vrna_eval_structure_simple, 357
vrna_eval_structure_simple_v, 360
vrna_eval_structure_simple_verbose, 360
vrna_eval_structure_v, 359
vrna_eval_structure_verbose, 359
exp_E_ExtLoop
Processing and Evaluating Decomposed Loops, 52
exp_E_Hairpin
Processing and Evaluating Decomposed Loops, 57
exp_E_IntLoop
Processing and Evaluating Decomposed Loops, 60
exp_E_Stem
Processing and Evaluating Decomposed Loops, 54
exp_f
vrna_sc_s, 239
expHairpinEnergy
part_func.h, 398
expLoopEnergy
part_func.h, 398
expand_Full
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 289
expand_Shapiro
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 289
export_ali_bppm
Partition Function and Base Pair Probabilities for Sequence Alignment(s), 193
export_bppm
Computing Partition Functions and Pair Probabilities, 143
export_circfold_arrays
MFE Structures of single Nucleic Acid Sequences, 165
export_circfold_arrays_par
MFE Structures of single Nucleic Acid Sequences, 165
export_co_bppm
part_func_co.h, 402
export_cofold_arrays
MFE Structures of two hybridized Sequences, 169
export_cofold_arrays_gq
MFE Structures of two hybridized Sequences, 168
export_fold_arrays
MFE Structures of single Nucleic Acid Sequences, 164
export_fold_arrays_par
MFE Structures of single Nucleic Acid Sequences, 164

f
vrna_sc_s, 239
FILENAME_ID_LENGTH
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 282
FILENAME_MAX_LENGTH
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 282
FORBIDDEN
energy_const.h, 353
filecopy
utils.h, 428
final_cost
Inverse Secondary Structure Prediction, 48
find_saddle
Direct refolding paths between two secondary structures, 279
fold
MFE Structures of single Nucleic Acid Sequences, 163
fold_par
MFE Structures of single Nucleic Acid Sequences, 162
fold_vars.h
base_pair, 377
cut_point, 377
iindx, 377
james_rule, 377
logML, 377
pr, 377
RibosumFile, 377

Free Energy Evaluation for given Sequence / Structure Pairs, 50
vrna_eval_hp_loop, 50
free_alifold_arrays
MFE Consensus Structures for Sequence Alignment(s), 188
free_alipf_arrays
Partition Function and Base Pair Probabilities for Sequence Alignment(s), 193
free_arrays
MFE Structures of single Nucleic Acid Sequences, 164
free_auxdata
vrna_fc_s, 260
free_co_arrays
MFE Structures of two hybridized Sequences, 168
free_co_pf_arrays
part_func_co.h, 402
free_data
vrna_hc_s, 230
free_path
Direct refolding paths between two secondary structures, 280
free_pf_arrays
Computing Partition Functions and Pair Probabilities, 142
free_profile
profiledist.h, 410
free_sequence_arrays
Predicting Consensus Structures from Alignment(s), 183
free_tree
treedist.h, 423
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 311
aliPS_color_aln, 313
gmlRNA, 315
PS_dot_plot, 318
PS_dot_plot_list, 317
PS_rna_plot, 317
PS_rna_plot_a, 317
PS_rna_plot_a_gquad, 317
rna_plot_type, 318
simple_circplot_coordinates, 314
simple_xy_coordinates, 313
ssv_rna_plot, 316
svg_rna_plot, 316
VRNA_PLOT_TYPE_CIRCULAR, 313
VRNA_PLOT_TYPE_NAVIEW, 313
VRNA_PLOT_TYPE_SIMPLE, 313
vrna_file_PS_rnaplot, 314
vrna_file_PS_rnaplot_a, 315
xrna_plot, 316
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 304
read_record, 310
VRNA_CONSTRAINT_MULTILINE, 305
VRNA_OPTION_MULTILINE, 305
vrna_extract_record_rest_constraint, 309
vrna_extract_record_rest_structure, 308
vrna_file_SHAPE_read, 309
vrna_file_bpseq, 306
vrna_file_connect, 306
vrna_file_constraints_read, 309
vrna_file_fasta_read_record, 307
vrna_file_helixlist, 305
vrna_file_json, 306

GASCONST
energy_const.h, 352
Generate soft constraints from data, 251
progress_callback, 253
VRNA_MINIMIZER_CONJUGATE_FR, 252
VRNA_MINIMIZER_CONJUGATE_PR, 252
VRNA_MINIMIZER_STEEPEST_DESCENT, 253
VRNA_MINIMIZER_VECTOR_BFGS2, 253
VRNA_MINIMIZER_VECTOR_BFGS, 252
VRNA_OBJECTIVE_FUNCTION_ABSOLUTE, 252
VRNA_OBJECTIVE_FUNCTION_QUADRATIC, 252
vrna_sc_minimize_pertubation, 253
get_TwoDfold_variables
Calculating MFE representatives of a Distance Based Partitioning, 218
get_TwoDpfold_variables
2Dpfold.h, 326
get_alipf_arrays
Predicting Consensus Structures from Alignment(s), 181
get_boltzmann_factor_copy
Energy Parameter Sets and Boltzmann Factors, 72
get_boltzmann_factors
Energy Parameter Sets and Boltzmann Factors, 71
get_boltzmann_factors_ali
Energy Parameter Sets and Boltzmann Factors, 72
get_centroid_struct_gquad_pr
part_func.h, 397
get_centroid_struct_pl
centroid.h, 336
get_centroid_struct_pr
centroid.h, 336
get_concentrations
part_func_co.h, 401
get_gquad_matrix
Processing and Evaluating Decomposed Loops, 54
get_input_line
Utilities, 130
get_line
Utilities, 130
get_monomere_mfes
MFE Structures of two hybridized Sequences, 170
get_mpi
Predicting Consensus Structures from Alignment(s), 182
get_path

Direct refolding paths between two secondary structures, 280
get_pf_arrays
Computing Partition Functions and Pair Probabilities, 144
get_plist
part_func_co.h, 401
get_scaled_alipf_parameters
Energy Parameter Sets and Boltzmann Factors, 72
get_scaled_parameters
Energy Parameter Sets and Boltzmann Factors, 73
get_scaled_pf_parameters
Energy Parameter Sets and Boltzmann Factors, 71
give_up
Inverse Secondary Structure Prediction, 48
gmlRNA
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 315

HairpinE
MFE Structures of single Nucleic Acid Sequences, 165
hamming
string_utils.h, 416
hamming_bound
string_utils.h, 416
Hard Constraints, 227
VRNA_CONSTRAINT_DB_DEFAULT, 232
VRNA_CONSTRAINT_DB_DOT, 231
VRNA_CONSTRAINT_DB_ENFORCE_BP, 230
VRNA_CONSTRAINT_DB_GQUAD, 232
VRNA_CONSTRAINT_DB_INTERMOL, 232
VRNA_CONSTRAINT_DB_INTRAMOL, 231
VRNA_CONSTRAINT_DB_PIPE, 230
VRNA_CONSTRAINT_DB_RND_BRACK, 231
VRNA_CONSTRAINT_DB_X, 231
VRNA_CONSTRAINT_DB, 230
vrna_callback_hc_evaluate, 233
vrna_hc_add_bp, 235
vrna_hc_add_bp_nonspecific, 235
vrna_hc_add_from_db, 236
vrna_hc_add_up, 234
vrna_hc_add_up_batch, 234
vrna_hc_free, 235
vrna_hc_init, 234

INF
energy_const.h, 352
id
vrna_exp_param_s, 65
iindx
fold_vars.h, 377
Incorporating ligands binding to specific sequence/structure motifs, 249
vrna_sc_add_hi_motif, 249
Incorporating SHAPE reactivity data, 245
vrna_sc_SHAPE_to_pr, 247
vrna_sc_add_SHAPE_deigan, 246
vrna_sc_add_SHAPE_deigan_ali, 246
vrna_sc_add_SHAPE_zarringhalam, 247
init_co_pf_fold
part_func_co.h, 402
init_pf_fold
part_func.h, 397
init_pf_foldLP
LPfold.h, 384
init_rand
utils.h, 427
initialize_cofold
MFE Structures of two hybridized Sequences, 170
initialize_fold
MFE Structures of single Nucleic Acid Sequences, 165
int_urn
utils.h, 427
interact, 121
inv_verbose
Inverse Secondary Structure Prediction, 48
Inverse Secondary Structure Prediction, 47
final_cost, 48
give_up, 48
inv_verbose, 48
inverse_fold, 47
inverse_pf_fold, 48
inverse_fold
Inverse Secondary Structure Prediction, 47
inverse_pf_fold
Inverse Secondary Structure Prediction, 48

james_rule
fold_vars.h, 377

K0
energy_const.h, 352

LPfold.h
init_pf_foldLP, 384
Lfold
Local MFE structure Prediction and Z-scores, 198
Lfoldz
Local MFE structure Prediction and Z-scores, 198
Local MFE consensus structures for Sequence Alignments, 204
aliLfold, 204
Local MFE structure Prediction and Z-scores, 196
Lfold, 198
Lfoldz, 198
vrna_Lfold, 196
vrna_Lfoldz, 198
vrna_mfe_window, 199
vrna_mfe_window_zscore, 199
logML
fold_vars.h, 377
loop_energy
eval.h, 369
LoopEnergy
MFE Structures of single Nucleic Acid Sequences, 165

MAXLOOP
energy_const.h, 353
MEA.h
MEA, 385
MEA
MEA.h, 385
MFE Consensus Structures for Sequence Alignment(s), 185
alifold, 187
circalifold, 188
free_alifold_arrays, 188
vrna_alifold, 185
vrna_circalifold, 187
MFE Structures of single Nucleic Acid Sequences, 160
circfold, 163
export_circfold_arrays, 165
export_circfold_arrays_par, 165
export_fold_arrays, 164
export_fold_arrays_par, 164
fold, 163
fold_par, 162
free_arrays, 164
HairpinE, 165
initialize_fold, 165
LoopEnergy, 165
update_fold_params, 164
update_fold_params_par, 164
vrna_circfold, 161
vrna_fold, 161
MFE Structures of two hybridized Sequences, 166
cofold, 167
cofold_par, 168
export_cofold_arrays, 169
export_cofold_arrays_gq, 168
free_co_arrays, 168
get_monomere_mfes, 170
initialize_cofold, 170
update_cofold_params, 168
update_cofold_params_par, 168
vrna_cofold, 167
vrna_mfe_dimer, 170
Make_bp_profile
profiledist.h, 410
Make_bp_profile_bppm
profiledist.h, 410
make_pair_table
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 296
make_pair_table_snoop
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 298
make_referenceBP_array
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 298
Make_swString
stringdist.h, 417
make_tree
treedist.h, 423
Manipulation of the Prediction Models, 74
backtrack_type, 104
canonicalBPonly, 104
dangles, 103
do_backtrack, 104
energy_set, 104
max_bp_span, 104
noLonelyPairs, 103
nonstandards, 104
pf_scale, 103
set_model_details, 102
temperature, 102
tetra_loop, 103
VRNA_MODEL_DEFAULT_ALI_CV_FACT, 85
VRNA_MODEL_DEFAULT_ALI_NC_FACT, 85
VRNA_MODEL_DEFAULT_ALI_OLD_EN, 85
VRNA_MODEL_DEFAULT_ALI_RIBO, 85
VRNA_MODEL_DEFAULT_BACKTRACK_TYPE, 84
VRNA_MODEL_DEFAULT_BACKTRACK, 83
VRNA_MODEL_DEFAULT_BETA_SCALE, 81
VRNA_MODEL_DEFAULT_CIRC, 83
VRNA_MODEL_DEFAULT_COMPUTE_BPP, 84
VRNA_MODEL_DEFAULT_DANGLES, 82
VRNA_MODEL_DEFAULT_ENERGY_SET, 83
VRNA_MODEL_DEFAULT_GQUAD, 83
VRNA_MODEL_DEFAULT_LOG_ML, 84
VRNA_MODEL_DEFAULT_MAX_BP_SPAN, 84
VRNA_MODEL_DEFAULT_NO_GU_CLOSURE, 82
VRNA_MODEL_DEFAULT_NO_GU, 82
VRNA_MODEL_DEFAULT_NO_LP, 82
VRNA_MODEL_DEFAULT_PF_SCALE, 81
VRNA_MODEL_DEFAULT_SPECIAL_HP, 82
VRNA_MODEL_DEFAULT_TEMPERATURE, 81
VRNA_MODEL_DEFAULT_UNIQ_ML, 83
VRNA_MODEL_DEFAULT_WINDOW_SIZE, 84
vrna_md_defaults_backtrack, 95
vrna_md_defaults_backtrack_get, 96
vrna_md_defaults_backtrack_type, 96
vrna_md_defaults_backtrack_type_get, 96
vrna_md_defaults_betaScale, 87
vrna_md_defaults_betaScale_get, 88
vrna_md_defaults_circ, 92
vrna_md_defaults_circ_get, 92
vrna_md_defaults_compute_bpp, 96
vrna_md_defaults_compute_bpp_get, 97
vrna_md_defaults_cv_fact, 100
vrna_md_defaults_cv_fact_get, 101
vrna_md_defaults_dangles, 88
vrna_md_defaults_dangles_get, 88
vrna_md_defaults_energy_set, 95
vrna_md_defaults_energy_set_get, 95
vrna_md_defaults_gquad, 92
vrna_md_defaults_gquad_get, 94
vrna_md_defaults_logML_get, 91
vrna_md_defaults_logML, 91
vrna_md_defaults_max_bp_span, 97

vrna_md_defaults_max_bp_span_get, 97
vrna_md_defaults_min_loop_size, 98
vrna_md_defaults_min_loop_size_get, 98
vrna_md_defaults_nc_fact, 101
vrna_md_defaults_nc_fact_get, 101
vrna_md_defaults_noGU_get, 90
vrna_md_defaults_noGUclosure, 90
vrna_md_defaults_noGUclosure_get, 91
vrna_md_defaults_noGU, 90
vrna_md_defaults_noLP_get, 89
vrna_md_defaults_noLP, 89
vrna_md_defaults_oldAliEn, 99
vrna_md_defaults_oldAliEn_get, 99
vrna_md_defaults_reset, 86
vrna_md_defaults_ribo, 100
vrna_md_defaults_ribo_get, 100
vrna_md_defaults_sfact, 101
vrna_md_defaults_sfact_get, 102
vrna_md_defaults_special_hp, 88
vrna_md_defaults_special_hp_get, 89
vrna_md_defaults_temperature, 87
vrna_md_defaults_temperature_get, 87
vrna_md_defaults_uniq_ML_get, 94
vrna_md_defaults_uniq_ML, 94
vrna_md_defaults_window_size, 98
vrna_md_defaults_window_size_get, 99
vrna_md_option_string, 86
vrna_md_set_default, 85
vrna_md_update, 86
max_bp_span
Manipulation of the Prediction Models, 104
mean_bp_dist
part_func.h, 397
mean_bp_distance
Computing Partition Functions and Pair Probabilities, 145
mean_bp_distance_pr
Computing Partition Functions and Pair Probabilities, 145
min_loop_size
vrna_md_s, 81

n_seq
vrna_fc_s, 261
NBPAIRS
energy_const.h, 353
nc_fact
Predicting Consensus Structures from Alignment(s), 184
noLonelyPairs
Manipulation of the Prediction Models, 103
node, 122
nonstandards
Manipulation of the Prediction Models, 104
nrerror
utils.h, 427

PAIR
Data Structures and Preprocessor Macros, 123
PS_dot_plot
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 318
PS_dot_plot_list
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 317
PS_rna_plot
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 317
PS_rna_plot_a
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 317
PS_rna_plot_a_gquad
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 317
pack_structure
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 295
pair_info
Various utilities for Sequence Alignments, and Comparative Structure Prediction, 302
paramT
Energy Parameter Sets and Boltzmann Factors, 65
parenthesis_structure
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 299
parenthesis_zuker
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 299
parse_gquad
Processing and Evaluating Decomposed Loops, 54
parse_structure
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 290
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 285
add_root, 288
alimake_pair_table, 298
assign_plist_from_db, 295
b2HIT, 287
b2Shapiro, 288
b2C, 288
bp_distance, 298
bppm_symbol, 300
bppm_to_structure, 299
compute_BPdifferences, 299
copy_pair_table, 296
expand_Full, 289
expand_Shapiro, 289
make_pair_table, 296
make_pair_table_snoop, 298
make_referenceBP_array, 298
pack_structure, 295
parenthesis_structure, 299
parenthesis_zuker, 299
parse_structure, 290
unexpand_Full, 289
unexpand_aligned_F, 290

unpack_structure, 296
unweight, 289
vrna_bp_distance, 293
vrna_db_from_bp_stack, 293
vrna_db_from_plist, 294
vrna_db_from_ptable, 292
vrna_db_pack, 290
vrna_db_unpack, 291
vrna_plist, 294
vrna_pt_pk_get, 292
vrna_pt_snoop_get, 292
vrna_ptable, 291
vrna_ptable_copy, 292
vrna_refBPcnt_matrix, 293
vrna_refBPdist_matrix, 293
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 281
FILENAME_ID_LENGTH, 282
FILENAME_MAX_LENGTH, 282
vrna_cut_point_insert, 283
vrna_cut_point_remove, 284
vrna_hamming_distance, 282
vrna_hamming_distance_bound, 283
vrna_random_string, 282
vrna_seq_toRNA, 283
vrna_seq_toupper, 283
part_func.h
centroid, 397
expHairpinEnergy, 398
expLoopEnergy, 398
get_centroid_struct_gquad_pr, 397
init_pf_fold, 397
mean_bp_dist, 397
stackProb, 397
vrna_pf_float_precision, 397
part_func_co.h
co_pf_fold, 400
co_pf_fold_par, 400
compute_probabilities, 401
export_co_bppm, 402
free_co_pf_arrays, 402
get_concentrations, 401
get_plist, 401
init_co_pf_fold, 402
update_co_pf_params, 403
update_co_pf_params_par, 403
Partition Function and Base Pair Probabilities for Sequence Alignment(s), 190
alipf_circ_fold, 192
alipf_fold, 192
alipf_fold_par, 192
export_ali_bppm, 193
free_alipf_arrays, 193
vrna_pf_alifold, 190
vrna_pf_circalifold, 191
Partition Function for two hybridized Sequences, 172
vrna_pf_dimer, 174
vrna_pf_dimer_concentrations, 175
vrna_pf_dimer_probs, 174
Partition Function for two hybridized Sequences as a stepwise Process, 176
pf_interact, 177
pf_unstru, 177
Partition functions for locally stable secondary structures, 201
pfl_fold, 202
putoutpU_prob, 202
putoutpU_prob_bin, 203
update_pf_paramsLP, 201
path_t
Direct refolding paths between two secondary structures, 278
pbacktrack
Stochastic backtracking in the Ensemble, 157
pbacktrack_circ
Stochastic backtracking in the Ensemble, 158
pf_circ_fold
Computing Partition Functions and Pair Probabilities, 142
pf_fold
Computing Partition Functions and Pair Probabilities, 141
pf_fold_par
Computing Partition Functions and Pair Probabilities, 140
pf_interact
Partition Function for two hybridized Sequences as a stepwise Process, 177
pf_paramT
Energy Parameter Sets and Boltzmann Factors, 65
pf_scale
Manipulation of the Prediction Models, 103
pf_unstru
Partition Function for two hybridized Sequences as a stepwise Process, 177
pfl_fold
Partition functions for locally stable secondary structures, 202
plist
Data Structures and Preprocessor Macros, 123
Postorder_list, 319
pr
fold_vars.h, 377
Predicting Consensus Structures from Alignment(s), 179
alloc_sequence_arrays, 183
cv_fact, 184
encode_ali_sequence, 183
energy_of_alistruct, 180
free_sequence_arrays, 183
get_alipf_arrays, 181
get_mpi, 182
nc_fact, 184
update_alifold_params, 181
vrna_aln_mpi, 182
Predicting Locally stable structures of large sequences, 195

print_tty_constraint
constraints_hard.h, 342
print_tty_constraint_full
constraints_hard.h, 342
print_tty_input_seq
utils.h, 426
print_tty_input_seq_str
utils.h, 426
Processing and Evaluating Decomposed Loops, 51
backtrack_GQuad_IntLoop, 55
backtrack_GQuad_IntLoop_L, 55
E_ExtLoop, 52
E_Hairpin, 56
E_IntLoop, 58
E_Stem, 53
E_mb_loop_stack, 60
exp_E_ExtLoop, 52
exp_E_Hairpin, 57
exp_E_IntLoop, 60
exp_E_Stem, 54
get_gquad_matrix, 54
parse_gquad, 54
vrna_BT_hp_loop, 58
vrna_BT_mb_loop, 60
vrna_E_ext_hp_loop, 58
vrna_E_hp_loop, 57
vrna_exp_E_hp_loop, 58
profile_edit_distance
profiledist.h, 410
profiledist.h
free_profile, 410
Make_bp_profile, 410
Make_bp_profile_bppm, 410
profile_edit_distance, 410
progress_callback
Generate soft constraints from data, 253
pscore
vrna_fc_s, 263
pscore_pf_compat
vrna_fc_s, 263
ptype
vrna_fc_s, 260
ptype_pf_compat
vrna_fc_s, 261
pu_contrib, 121
pu_out, 122
putoutpU_prob
Partition functions for locally stable secondary structures, 202
putoutpU_prob_bin
Partition functions for locally stable secondary structures, 203

RNA Secondary Structure Prediction, 45
random_string
string_utils.h, 415
read_parameter_file
Reading/Writing Energy Parameter Sets from/to File, 205
read_record
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 310
Reading/Writing Energy Parameter Sets from/to File, 205
read_parameter_file, 205
write_parameter_file, 206
Refolding paths between secondary structues, 49
RibosumFile
fold_vars.h, 377
rna_plot_type
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 318

S
vrna_fc_s, 262
S3
vrna_fc_s, 262
S5
vrna_fc_s, 262
S_cons
vrna_fc_s, 262
SOLUTION
subopt.h, 422
sc
vrna_fc_s, 261
scale_parameters
Energy Parameter Sets and Boltzmann Factors, 72
scs
vrna_fc_s, 263
sect
Data Structures and Preprocessor Macros, 124
sequence
vrna_fc_s, 260
sequence_encoding
vrna_fc_s, 260
sequences
vrna_fc_s, 261
set_model_details
Manipulation of the Prediction Models, 102
simple_circplot_coordinates
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 314
simple_xy_coordinates
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 313
snoopT, 123
Soft Constraints, 237
vrna_callback_sc_backtrack, 240
vrna_callback_sc_energy, 239
vrna_callback_sc_exp_energy, 240
vrna_sc_add_bp, 241
vrna_sc_add_bt, 243
vrna_sc_add_data, 243
vrna_sc_add_exp_f, 244
vrna_sc_add_f, 243
vrna_sc_add_up, 242
vrna_sc_free, 242

vrna_sc_init, 241
vrna_sc_remove, 242
space
utils.h, 427
ssv_rna_plot
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 316
st_back
Stochastic backtracking in the Ensemble, 158
stackProb
part_func.h, 397
stat_cb
vrna_fc_s, 259
Stochastic backtracking in the Ensemble, 156
pbacktrack, 157
pbacktrack_circ, 158
st_back, 158
vrna_pbacktrack, 157
vrna_pbacktrack5, 156
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), 194
alipbacktrack, 194
Stochastic Backtracking of Structures from Distance Based Partitioning, 224
vrna_pbacktrack5_TwoD, 225
vrna_pbacktrack_TwoD, 224
str_DNA2RNA
string_utils.h, 415
str_uppercase
string_utils.h, 415
string_edit_distance
stringdist.h, 418
string_utils.h
hamming, 416
hamming_bound, 416
random_string, 415
str_DNA2RNA, 415
str_uppercase, 415
stringdist.h
Make_swString, 417
string_edit_distance, 418
subopt
Suboptimal structures within an energy band arround the MFE, 154
subopt.h
SOLUTION, 422
subopt_circ
Suboptimal structures within an energy band arround the MFE, 154
Suboptimal structures according to Zuker et al. 1989, 151
vrna_subopt_zuker, 151
zukersubopt, 152
zukersubopt_par, 152
Suboptimal structures within an energy band arround the MFE, 153
subopt, 154
subopt_circ, 154
vrna_subopt, 153
svg_rna_plot
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 316
swString, 319

TURN
energy_const.h, 353
temperature
Manipulation of the Prediction Models, 102
tetra_loop
Manipulation of the Prediction Models, 103
The Dynamic Programming Matrices, 270
VRNA_MX_2DFOLD, 274
VRNA_MX_DEFAULT, 274
VRNA_MX_WINDOW, 274
vrna_mx_add, 275
vrna_mx_mfe_free, 275
vrna_mx_pf_free, 276
vrna_mx_type_e, 274
The Fold Compound, 255
VRNA_OPTION_EVAL_ONLY, 264
VRNA_OPTION_MFE, 264
VRNA_OPTION_PF, 264
VRNA_STATUS_MFE_POST, 263
VRNA_STATUS_MFE_PRE, 263
VRNA_STATUS_PF_POST, 264
VRNA_STATUS_PF_PRE, 264
VRNA_VC_TYPE_ALIGNMENT, 266
VRNA_VC_TYPE_SINGLE, 266
vrna_callback_free_auxdata, 265
vrna_callback_recursion_status, 265
vrna_fc_type_e, 265
vrna_fold_compound, 266
vrna_fold_compound_add_auxdata, 268
vrna_fold_compound_add_callback, 269
vrna_fold_compound_comparative, 267
vrna_fold_compound_free, 268
time_stamp
utils.h, 428
Tree, 320
tree_edit_distance
treedist.h, 423
treedist.h
free_tree, 423
make_tree, 423
tree_edit_distance, 423
TwoDfold_backtrack_f5
Calculating MFE representatives of a Distance Based Partitioning, 220
TwoDfold_vars, 215
Calculating MFE representatives of a Distance Based Partitioning, 217
TwoDfoldList
Calculating MFE representatives of a Distance Based Partitioning, 219
TwoDpfold_pbacktrack
2Dpfold.h, 327
TwoDpfold_pbacktrack5

2Dpfold.h, 328
TwoDpfold_vars, 320
TwoDpfoldList
2Dpfold.h, 326
type
vrna_fc_s, 259

unexpand_Full
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 289
unexpand_aligned_F
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 290
unpack_structure
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 296
unweight
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 289
update_alifold_params
Predicting Consensus Structures from Alignment(s), 181
update_co_pf_params
part_func_co.h, 403
update_co_pf_params_par
part_func_co.h, 403
update_cofold_params
MFE Structures of two hybridized Sequences, 168
update_cofold_params_par
MFE Structures of two hybridized Sequences, 168
update_fold_params
MFE Structures of single Nucleic Acid Sequences, 164
update_fold_params_par
MFE Structures of single Nucleic Acid Sequences, 164
update_pf_params
Computing Partition Functions and Pair Probabilities, 143
update_pf_params_par
Computing Partition Functions and Pair Probabilities, 143
update_pf_paramsLP
Partition functions for locally stable secondary structures, 201
urn
utils.h, 427
Utilities, 125
get_input_line, 130
get_line, 130
VRNA_INPUT_CONSTRAINT, 127
VRNA_INPUT_FASTA_HEADER, 127
vrna_alloc, 127
vrna_idx_col_wise, 132
vrna_idx_row_wise, 131
vrna_int_urn, 129
vrna_message_error, 128
vrna_message_input_seq, 131
vrna_message_input_seq_simple, 131
vrna_message_warning, 128
vrna_realloc, 128
vrna_time_stamp, 129
vrna_urn, 129
xsubi, 132
utils.h
filecopy, 428
init_rand, 427
int_urn, 427
nrerror, 427
print_tty_input_seq, 426
print_tty_input_seq_str, 426
space, 427
time_stamp, 428
urn, 427
warn_user, 426
xrealloc, 427

VRNA_CONSTRAINT_DB_ANG_BRACK
constraints_hard.h, 341
VRNA_CONSTRAINT_DB_DEFAULT
Hard Constraints, 232
VRNA_CONSTRAINT_DB_DOT
Hard Constraints, 231
VRNA_CONSTRAINT_DB_ENFORCE_BP
Hard Constraints, 230
VRNA_CONSTRAINT_DB_GQUAD
Hard Constraints, 232
VRNA_CONSTRAINT_DB_INTERMOL
Hard Constraints, 232
VRNA_CONSTRAINT_DB_INTRAMOL
Hard Constraints, 231
VRNA_CONSTRAINT_DB_PIPE
Hard Constraints, 230
VRNA_CONSTRAINT_DB_RND_BRACK
Hard Constraints, 231
VRNA_CONSTRAINT_DB_X
Hard Constraints, 231
VRNA_CONSTRAINT_DB
Hard Constraints, 230
VRNA_CONSTRAINT_FILE
Constraining the Secondary Structure Predictions and Evaluations, 109
VRNA_CONSTRAINT_MULTILINE
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 305
VRNA_CONSTRAINT_NO_HEADER
constraints_hard.h, 341
VRNA_CONSTRAINT_SOFT_MFE
Constraining the Secondary Structure Predictions and Evaluations, 109
VRNA_CONSTRAINT_SOFT_PF
Constraining the Secondary Structure Predictions and Evaluations, 109
VRNA_CONVERT_OUTPUT_ALL
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_BULGE
Converting Energy Parameter Files, 209

VRNA_CONVERT_OUTPUT_DANGLE3
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_DANGLE5
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_DUMP
Converting Energy Parameter Files, 210
VRNA_CONVERT_OUTPUT_HP
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_INT_11
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_INT_21
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_INT_22
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_INT
Converting Energy Parameter Files, 210
VRNA_CONVERT_OUTPUT_MISC
Converting Energy Parameter Files, 210
VRNA_CONVERT_OUTPUT_MM_EXT
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_MM_HP
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_MM_INT_1N
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_MM_INT_23
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_MM_INT
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_MM_MULTI
Converting Energy Parameter Files, 209
VRNA_CONVERT_OUTPUT_ML
Converting Energy Parameter Files, 210
VRNA_CONVERT_OUTPUT_NINIO
Converting Energy Parameter Files, 210
VRNA_CONVERT_OUTPUT_SPECIAL_HP
Converting Energy Parameter Files, 210
VRNA_CONVERT_OUTPUT_STACK
Converting Energy Parameter Files, 208
VRNA_CONVERT_OUTPUT_VANILLA
Converting Energy Parameter Files, 210
VRNA_DECOMP_EXT_EXT_EXT
Constraining the Secondary Structure Predictions and Evaluations, 115
VRNA_DECOMP_EXT_EXT_STEM1
Constraining the Secondary Structure Predictions and Evaluations, 116
VRNA_DECOMP_EXT_EXT_STEM
Constraining the Secondary Structure Predictions and Evaluations, 115
VRNA_DECOMP_EXT_EXT
Constraining the Secondary Structure Predictions and Evaluations, 114
VRNA_DECOMP_EXT_STEM_EXT
Constraining the Secondary Structure Predictions and Evaluations, 115
VRNA_DECOMP_EXT_STEM
Constraining the Secondary Structure Predictions and Evaluations, 114
VRNA_DECOMP_EXT_UP
Constraining the Secondary Structure Predictions and Evaluations, 114
VRNA_DECOMP_ML_COAXIAL
Constraining the Secondary Structure Predictions and Evaluations, 113
VRNA_DECOMP_ML_ML_ML
Constraining the Secondary Structure Predictions and Evaluations, 111
VRNA_DECOMP_ML_ML_STEM
Constraining the Secondary Structure Predictions and Evaluations, 113
VRNA_DECOMP_ML_ML
Constraining the Secondary Structure Predictions and Evaluations, 112
VRNA_DECOMP_ML_STEM
Constraining the Secondary Structure Predictions and Evaluations, 111
VRNA_DECOMP_ML_UP
Constraining the Secondary Structure Predictions and Evaluations, 112
VRNA_DECOMP_PAIR_HP
Constraining the Secondary Structure Predictions and Evaluations, 109
VRNA_DECOMP_PAIR_IL
Constraining the Secondary Structure Predictions and Evaluations, 110
VRNA_DECOMP_PAIR_ML
Constraining the Secondary Structure Predictions and Evaluations, 110
VRNA_INPUT_CONSTRAINT
Utilities, 127
VRNA_INPUT_FASTA_HEADER
Utilities, 127
VRNA_MINIMIZER_CONJUGATE_FR
Generate soft constraints from data, 252
VRNA_MINIMIZER_CONJUGATE_PR
Generate soft constraints from data, 252
VRNA_MINIMIZER_STEEPEST_DESCENT
Generate soft constraints from data, 253
VRNA_MINIMIZER_VECTOR_BFGS2
Generate soft constraints from data, 253
VRNA_MINIMIZER_VECTOR_BFGS
Generate soft constraints from data, 252
VRNA_MODEL_DEFAULT_ALI_CV_FACT
Manipulation of the Prediction Models, 85
VRNA_MODEL_DEFAULT_ALI_NC_FACT
Manipulation of the Prediction Models, 85
VRNA_MODEL_DEFAULT_ALI_OLD_EN
Manipulation of the Prediction Models, 85
VRNA_MODEL_DEFAULT_ALI_RIBO
Manipulation of the Prediction Models, 85
VRNA_MODEL_DEFAULT_BACKTRACK_TYPE
Manipulation of the Prediction Models, 84
VRNA_MODEL_DEFAULT_BACKTRACK
Manipulation of the Prediction Models, 83
VRNA_MODEL_DEFAULT_BETA_SCALE
Manipulation of the Prediction Models, 81

VRNA_MODEL_DEFAULT_CIRC
Manipulation of the Prediction Models, 83
VRNA_MODEL_DEFAULT_COMPUTE_BPP
Manipulation of the Prediction Models, 84
VRNA_MODEL_DEFAULT_DANGLES
Manipulation of the Prediction Models, 82
VRNA_MODEL_DEFAULT_ENERGY_SET
Manipulation of the Prediction Models, 83
VRNA_MODEL_DEFAULT_GQUAD
Manipulation of the Prediction Models, 83
VRNA_MODEL_DEFAULT_LOG_ML
Manipulation of the Prediction Models, 84
VRNA_MODEL_DEFAULT_MAX_BP_SPAN
Manipulation of the Prediction Models, 84
VRNA_MODEL_DEFAULT_NO_GU_CLOSURE
Manipulation of the Prediction Models, 82
VRNA_MODEL_DEFAULT_NO_GU
Manipulation of the Prediction Models, 82
VRNA_MODEL_DEFAULT_NO_LP
Manipulation of the Prediction Models, 82
VRNA_MODEL_DEFAULT_PF_SCALE
Manipulation of the Prediction Models, 81
VRNA_MODEL_DEFAULT_SPECIAL_HP
Manipulation of the Prediction Models, 82
VRNA_MODEL_DEFAULT_TEMPERATURE
Manipulation of the Prediction Models, 81
VRNA_MODEL_DEFAULT_UNIQ_ML
Manipulation of the Prediction Models, 83
VRNA_MODEL_DEFAULT_WINDOW_SIZE
Manipulation of the Prediction Models, 84
VRNA_MX_2DFOLD
The Dynamic Programming Matrices, 274
VRNA_MX_DEFAULT
The Dynamic Programming Matrices, 274
VRNA_MX_WINDOW
The Dynamic Programming Matrices, 274
VRNA_OBJECTIVE_FUNCTION_ABSOLUTE
Generate soft constraints from data, 252
VRNA_OBJECTIVE_FUNCTION_QUADRATIC
Generate soft constraints from data, 252
VRNA_OPTION_EVAL_ONLY
The Fold Compound, 264
VRNA_OPTION_MFE
The Fold Compound, 264
VRNA_OPTION_MULTILINE
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 305
VRNA_OPTION_PF
The Fold Compound, 264
VRNA_PLOT_TYPE_CIRCULAR
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 313
VRNA_PLOT_TYPE_NAVIEW
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 313
VRNA_PLOT_TYPE_SIMPLE
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 313
VRNA_STATUS_MFE_POST
The Fold Compound, 263
VRNA_STATUS_MFE_PRE
The Fold Compound, 263
VRNA_STATUS_PF_POST
The Fold Compound, 264
VRNA_STATUS_PF_PRE
The Fold Compound, 264
VRNA_VC_TYPE_ALIGNMENT
The Fold Compound, 266
VRNA_VC_TYPE_SINGLE
The Fold Compound, 266
Various utilities for Sequence Alignments, and Comparative Structure Prediction, 301
pair_info, 302
vrna_aln_pinfo, 302
ViennaRNA/1.8.4_epars.h, 323
ViennaRNA/1.8.4_intloops.h, 323
ViennaRNA/2Dfold.h, 324
ViennaRNA/2Dpfold.h, 325
ViennaRNA/LPfold.h, 384
ViennaRNA/Lfold.h, 382
ViennaRNA/MEA.h, 385
ViennaRNA/PS_dot.h, 411
ViennaRNA/RNAstruct.h, 412
ViennaRNA/alifold.h, 328
ViennaRNA/aln_util.h, 330
ViennaRNA/alphabet.h, 331
ViennaRNA/boltzmann_sampling.h, 333
ViennaRNA/centroid.h, 335
ViennaRNA/cofold.h, 336
ViennaRNA/constraints.h, 337
ViennaRNA/constraints_SHAPE.h, 342
ViennaRNA/constraints_hard.h, 339
ViennaRNA/constraints_soft.h, 344
ViennaRNA/convert_epars.h, 345
ViennaRNA/data_structures.h, 346
ViennaRNA/dist_vars.h, 349
ViennaRNA/dp_matrices.h, 350
ViennaRNA/duplex.h, 351
ViennaRNA/edit_cost.h, 351
ViennaRNA/energy_const.h, 352
ViennaRNA/equilibrium_probs.h, 353
ViennaRNA/eval.h, 354
ViennaRNA/exterior_loops.h, 371
ViennaRNA/file_formats.h, 372
ViennaRNA/findpath.h, 374
ViennaRNA/fold.h, 375
ViennaRNA/fold_vars.h, 376
ViennaRNA/gquad.h, 378
ViennaRNA/hairpin_loops.h, 379
ViennaRNA/interior_loops.h, 380
ViennaRNA/inverse.h, 381
ViennaRNA/ligand.h, 382
ViennaRNA/loop_energies.h, 383
ViennaRNA/mfe.h, 386

ViennaRNA/mm.h, 386
ViennaRNA/model.h, 387
ViennaRNA/multibranch_loops.h, 391
ViennaRNA/naview.h, 393
ViennaRNA/params.h, 393
ViennaRNA/part_func.h, 395
ViennaRNA/part_func_co.h, 398
ViennaRNA/part_func_up.h, 403
ViennaRNA/perturbation_fold.h, 404
ViennaRNA/plot_aln.h, 406
ViennaRNA/plot_layouts.h, 406
ViennaRNA/plot_structure.h, 408
ViennaRNA/profiledist.h, 409
ViennaRNA/read_epars.h, 411
ViennaRNA/ribo.h, 412
ViennaRNA/string_utils.h, 414
ViennaRNA/stringdist.h, 416
ViennaRNA/structure_utils.h, 418
ViennaRNA/subopt.h, 421
ViennaRNA/treedist.h, 422
ViennaRNA/utils.h, 423
vrna_BT_hp_loop
    Processing and Evaluating Decomposed Loops, 58
vrna_BT_mb_loop
    Processing and Evaluating Decomposed Loops, 60
vrna_E_ext_hp_loop
    Processing and Evaluating Decomposed Loops, 58
vrna_E_hp_loop
    Processing and Evaluating Decomposed Loops, 57
vrna_Lfold
    Local MFE structure Prediction and Z-scores, 196
vrna_Lfoldz
    Local MFE structure Prediction and Z-scores, 198
vrna_alifold
    MFE Consensus Structures for Sequence Alignment(s), 185
vrna_alloc
    Utilities, 127
vrna_aln_mpi
    Predicting Consensus Structures from Alignment(s), 182
vrna_aln_pinfo
    Various utilities for Sequence Alignments, and Comparative Structure Prediction, 302
vrna_backtrack5_TwoD
    Calculating MFE representatives of a Distance Based Partitioning, 218
vrna_basepair_s, 121
vrna_bp_distance
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 293
vrna_bp_stack_s, 121
vrna_callback_free_auxdata
    The Fold Compound, 265
vrna_callback_hc_evaluate
    Hard Constraints, 233
vrna_callback_recursion_status
    The Fold Compound, 265
vrna_callback_sc_backtrack
    Soft Constraints, 240
vrna_callback_sc_energy
    Soft Constraints, 239
vrna_callback_sc_exp_energy
    Soft Constraints, 240
vrna_centroid
    Compute the centroid structure, 148
vrna_centroid_from_plist
    Compute the centroid structure, 148
vrna_centroid_from_probs
    Compute the centroid structure, 149
vrna_circalifold
    MFE Consensus Structures for Sequence Alignment(s), 187
vrna_circfold
    MFE Structures of single Nucleic Acid Sequences, 161
vrna_cofold
    MFE Structures of two hybridized Sequences, 167
vrna_constraints_add
    Constraining the Secondary Structure Predictions and Evaluations, 116
vrna_cpair_s, 121
vrna_cut_point_insert
    Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 283
vrna_cut_point_remove
    Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 284
vrna_db_from_bp_stack
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 293
vrna_db_from_plist
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 294
vrna_db_from_ptable
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 292
vrna_db_pack
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 290
vrna_db_unpack
    Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 291
vrna_dimer_conc_s, 173
vrna_dimer_pf_s, 173
vrna_eval_covar_structure
    eval.h, 357
vrna_eval_hp_loop
    Free Energy Evaluation for given Sequence / Structure Pairs, 50
vrna_eval_loop_pt
    eval.h, 364
vrna_eval_move
    eval.h, 364

vrna_eval_move_pt
eval.h, 364
vrna_eval_structure
eval.h, 356
vrna_eval_structure_pt
eval.h, 361
vrna_eval_structure_pt_simple
eval.h, 361
vrna_eval_structure_pt_simple_v
eval.h, 363
vrna_eval_structure_pt_simple_verbose
eval.h, 363
vrna_eval_structure_pt_v
eval.h, 362
vrna_eval_structure_pt_verbose
eval.h, 362
vrna_eval_structure_simple
eval.h, 357
vrna_eval_structure_simple_v
eval.h, 360
vrna_eval_structure_simple_verbose
eval.h, 360
vrna_eval_structure_v
eval.h, 359
vrna_eval_structure_verbose
eval.h, 359
vrna_exp_E_hp_loop
Processing and Evaluating Decomposed Loops, 58
vrna_exp_param_s, 64
alpha, 65
id, 65
vrna_exp_params
Energy Parameter Sets and Boltzmann Factors, 66
vrna_exp_params_comparative
Energy Parameter Sets and Boltzmann Factors, 67
vrna_exp_params_copy
Energy Parameter Sets and Boltzmann Factors, 67
vrna_exp_params_rescale
Energy Parameter Sets and Boltzmann Factors, 68
vrna_exp_params_reset
Energy Parameter Sets and Boltzmann Factors, 70
vrna_exp_params_subst
Energy Parameter Sets and Boltzmann Factors, 68
vrna_extract_record_rest_constraint
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 309
vrna_extract_record_rest_structure
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 308
vrna_fc_s, 256
auxdata, 260
cons_seq, 262
free_auxdata, 260
n_seq, 261
pscore, 263
pscore_pf_compat, 263
ptype, 260
ptype_pf_compat, 261
S, 262
S3, 262
S5, 262
S_cons, 262
sc, 261
scs, 263
sequence, 260
sequence_encoding, 260
sequences, 261
stat_cb, 259
type, 259
vrna_fc_type_e
The Fold Compound, 265
vrna_file_PS_rnaplot
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 314
vrna_file_PS_rnaplot_a
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 315
vrna_file_SHAPE_read
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 309
vrna_file_bpseq
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 306
vrna_file_connect
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 306
vrna_file_constraints_read
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 309
vrna_file_fasta_read_record
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 307
vrna_file_helixlist
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 305
vrna_file_json
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 306
vrna_fold
MFE Structures of single Nucleic Acid Sequences, 161
vrna_fold_compound
The Fold Compound, 266
vrna_fold_compound_add_auxdata
The Fold Compound, 268
vrna_fold_compound_add_callback
The Fold Compound, 269

vrna_fold_compound_comparative
The Fold Compound, 267
vrna_fold_compound_free
The Fold Compound, 268
vrna_hamming_distance
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 282
vrna_hamming_distance_bound
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 283
vrna_hc_add_bp
Hard Constraints, 235
vrna_hc_add_bp_nonspecific
Hard Constraints, 235
vrna_hc_add_from_db
Hard Constraints, 236
vrna_hc_add_up
Hard Constraints, 234
vrna_hc_add_up_batch
Hard Constraints, 234
vrna_hc_free
Hard Constraints, 235
vrna_hc_init
Hard Constraints, 234
vrna_hc_s, 229
free_data, 230
vrna_hc_up_s, 230
vrna_hx_s, 287
vrna_idx_col_wise
Utilities, 132
vrna_idx_row_wise
Utilities, 131
vrna_int_urn
Utilities, 129
vrna_md_defaults_backtrack
Manipulation of the Prediction Models, 95
vrna_md_defaults_backtrack_get
Manipulation of the Prediction Models, 96
vrna_md_defaults_backtrack_type
Manipulation of the Prediction Models, 96
vrna_md_defaults_backtrack_type_get
Manipulation of the Prediction Models, 96
vrna_md_defaults_betaScale
Manipulation of the Prediction Models, 87
vrna_md_defaults_betaScale_get
Manipulation of the Prediction Models, 88
vrna_md_defaults_circ
Manipulation of the Prediction Models, 92
vrna_md_defaults_circ_get
Manipulation of the Prediction Models, 92
vrna_md_defaults_compute_bpp
Manipulation of the Prediction Models, 96
vrna_md_defaults_compute_bpp_get
Manipulation of the Prediction Models, 97
vrna_md_defaults_cv_fact
Manipulation of the Prediction Models, 100
vrna_md_defaults_cv_fact_get
Manipulation of the Prediction Models, 101
vrna_md_defaults_dangles
Manipulation of the Prediction Models, 88
vrna_md_defaults_dangles_get
Manipulation of the Prediction Models, 88
vrna_md_defaults_energy_set
Manipulation of the Prediction Models, 95
vrna_md_defaults_energy_set_get
Manipulation of the Prediction Models, 95
vrna_md_defaults_gquad
Manipulation of the Prediction Models, 92
vrna_md_defaults_gquad_get
Manipulation of the Prediction Models, 94
vrna_md_defaults_logML_get
Manipulation of the Prediction Models, 91
vrna_md_defaults_logML
Manipulation of the Prediction Models, 91
vrna_md_defaults_max_bp_span
Manipulation of the Prediction Models, 97
vrna_md_defaults_max_bp_span_get
Manipulation of the Prediction Models, 97
vrna_md_defaults_min_loop_size
Manipulation of the Prediction Models, 98
vrna_md_defaults_min_loop_size_get
Manipulation of the Prediction Models, 98
vrna_md_defaults_nc_fact
Manipulation of the Prediction Models, 101
vrna_md_defaults_nc_fact_get
Manipulation of the Prediction Models, 101
vrna_md_defaults_noGU_get
Manipulation of the Prediction Models, 90
vrna_md_defaults_noGUclosure
Manipulation of the Prediction Models, 90
vrna_md_defaults_noGUclosure_get
Manipulation of the Prediction Models, 91
vrna_md_defaults_noGU
Manipulation of the Prediction Models, 90
vrna_md_defaults_noLP_get
Manipulation of the Prediction Models, 89
vrna_md_defaults_noLP
Manipulation of the Prediction Models, 89
vrna_md_defaults_oldAliEn
Manipulation of the Prediction Models, 99
vrna_md_defaults_oldAliEn_get
Manipulation of the Prediction Models, 99
vrna_md_defaults_reset
Manipulation of the Prediction Models, 86
vrna_md_defaults_ribo
Manipulation of the Prediction Models, 100
vrna_md_defaults_ribo_get
Manipulation of the Prediction Models, 100
vrna_md_defaults_sfact
Manipulation of the Prediction Models, 101
vrna_md_defaults_sfact_get
Manipulation of the Prediction Models, 102
vrna_md_defaults_special_hp
Manipulation of the Prediction Models, 88

vrna_md_defaults_special_hp_get
Manipulation of the Prediction Models, 89
vrna_md_defaults_temperature
Manipulation of the Prediction Models, 87
vrna_md_defaults_temperature_get
Manipulation of the Prediction Models, 87
vrna_md_defaults_uniq_ML_get
Manipulation of the Prediction Models, 94
vrna_md_defaults_uniq_ML
Manipulation of the Prediction Models, 94
vrna_md_defaults_window_size
Manipulation of the Prediction Models, 98
vrna_md_defaults_window_size_get
Manipulation of the Prediction Models, 99
vrna_md_option_string
Manipulation of the Prediction Models, 86
vrna_md_s, 78
dangles, 81
min_loop_size, 81
vrna_md_set_default
Manipulation of the Prediction Models, 85
vrna_md_update
Manipulation of the Prediction Models, 86
vrna_mean_bp_distance
Computing Partition Functions and Pair Probabilities, 137
vrna_mean_bp_distance_pr
Computing Partition Functions and Pair Probabilities, 137
vrna_message_constraint_options
Constraining the Secondary Structure Predictions and Evaluations, 117
vrna_message_constraint_options_all
Constraining the Secondary Structure Predictions and Evaluations, 118
vrna_message_error
Utilities, 128
vrna_message_input_seq
Utilities, 131
vrna_message_input_seq_simple
Utilities, 131
vrna_message_warning
Utilities, 128
vrna_mfe
Computing Minimum Free Energy (MFE) Structures, 134
vrna_mfe_TwoD
Calculating MFE representatives of a Distance Based Partitioning, 217
vrna_mfe_dimer
MFE Structures of two hybridized Sequences, 170
vrna_mfe_window
Local MFE structure Prediction and Z-scores, 199
vrna_mfe_window_zscore
Local MFE structure Prediction and Z-scores, 199
vrna_mx_add
The Dynamic Programming Matrices, 275
vrna_mx_mfe_free
The Dynamic Programming Matrices, 275
vrna_mx_mfe_s, 271
vrna_mx_pf_free
The Dynamic Programming Matrices, 276
vrna_mx_pf_s, 272
vrna_mx_type_e
The Dynamic Programming Matrices, 274
vrna_nucleotide_decode
alphabet.h, 333
vrna_nucleotide_encode
alphabet.h, 332
vrna_param_s, 64
vrna_params
Energy Parameter Sets and Boltzmann Factors, 65
vrna_params_copy
Energy Parameter Sets and Boltzmann Factors, 66
vrna_params_reset
Energy Parameter Sets and Boltzmann Factors, 69
vrna_params_subst
Energy Parameter Sets and Boltzmann Factors, 68
vrna_path_findpath
Direct refolding paths between two secondary structures, 279
vrna_path_findpath_saddle
Direct refolding paths between two secondary structures, 278
vrna_path_s, 278
vrna_pbacktrack
Stochastic backtracking in the Ensemble, 157
vrna_pbacktrack5
Stochastic backtracking in the Ensemble, 156
vrna_pbacktrack5_TwoD
Stochastic Backtracking of Structures from Distance Based Partitioning, 225
vrna_pbacktrack_TwoD
Stochastic Backtracking of Structures from Distance Based Partitioning, 224
vrna_pf
Computing Partition Functions and Pair Probabilities, 138
vrna_pf_TwoD
Calculate Partition Functions of a Distance Based Partitioning, 222
vrna_pf_alifold
Partition Function and Base Pair Probabilities for Sequence Alignment(s), 190
vrna_pf_circalifold
Partition Function and Base Pair Probabilities for Sequence Alignment(s), 191
vrna_pf_circfold
Computing Partition Functions and Pair Probabilities, 139
vrna_pf_dimer
Partition Function for two hybridized Sequences, 174
vrna_pf_dimer_concentrations
Partition Function for two hybridized Sequences, 175

vrna_pf_dimer_probs
Partition Function for two hybridized Sequences, 174
vrna_pf_float_precision
part_func.h, 397
vrna_pf_fold
Computing Partition Functions and Pair Probabilities, 138
vrna_pinfo_s, 301
vrna_plist
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 294
vrna_plist_from_probs
Computing Partition Functions and Pair Probabilities, 145
vrna_plist_s, 121
vrna_pt_pk_get
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 292
vrna_pt_snoop_get
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 292
vrna_ptable
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 291
vrna_ptable_copy
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 292
vrna_ptypes
alphabet.h, 332
vrna_random_string
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 282
vrna_realloc
Utilities, 128
vrna_refBPcnt_matrix
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 293
vrna_refBPdist_matrix
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 293
vrna_sc_SHAPE_parse_method
constraints_SHAPE.h, 343
vrna_sc_SHAPE_to_pr
Incorporating SHAPE reactivity data, 247
vrna_sc_add_SHAPE_deigan
Incorporating SHAPE reactivity data, 246
vrna_sc_add_SHAPE_deigan_ali
Incorporating SHAPE reactivity data, 246
vrna_sc_add_SHAPE_zarringhalam
Incorporating SHAPE reactivity data, 247
vrna_sc_add_bp
Soft Constraints, 241
vrna_sc_add_bt
Soft Constraints, 243
vrna_sc_add_data
Soft Constraints, 243
vrna_sc_add_exp_f
Soft Constraints, 244
vrna_sc_add_f
Soft Constraints, 243
vrna_sc_add_hi_motif
Incorporating ligands binding to specific sequence/structure motifs, 249
vrna_sc_add_up
Soft Constraints, 242
vrna_sc_free
Soft Constraints, 242
vrna_sc_init
Soft Constraints, 241
vrna_sc_minimize_pertubation
Generate soft constraints from data, 253
vrna_sc_remove
Soft Constraints, 242
vrna_sc_s, 238
bt, 239
exp_f, 239
f, 239
vrna_sect_s, 121
vrna_seq_toRNA
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 283
vrna_seq_toupper
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 283
vrna_sol_TwoD_pf_t, 222
Calculate Partition Functions of a Distance Based Partitioning, 222
vrna_sol_TwoD_t, 215
Calculating MFE representatives of a Distance Based Partitioning, 217
vrna_stack_prob
Computing Partition Functions and Pair Probabilities, 137
vrna_subopt
Suboptimal structures within an energy band around the MFE, 153
vrna_subopt_sol_s, 322
vrna_subopt_zuker
Suboptimal structures according to Zuker et al. 1989, 151
vrna_time_stamp
Utilities, 129
vrna_urn
Utilities, 129

warn_user
utils.h, 426
write_parameter_file
Reading/Writing Energy Parameter Sets from/to File, 206

xrealloc
utils.h, 427

xrna_plot
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 316
xsubi
Utilities, 132

zukersubopt
Suboptimal structures according to Zuker et al. 1989, 152
zukersubopt_par
Suboptimal structures according to Zuker et al. 1989, 152