

RNAlib-2.2.0-RC3

Generated by Doxygen 1.8.9.1

Mon May 11 2015 11:28:29

Contents

1	ViennaRNA Package core - RNAlib	1
1.1	Introduction	1
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	Input / Output File Formats	15
5.1	File formats for Secondary Structure Constraints	15
5.1.1	Constraints Definition File	15
5.1.1.1	Constraint commands	15
5.1.1.2	Specification of the loop type context	15
5.1.1.3	Controlling the orientation of base pairing	16
5.1.1.4	Sequence coordinates	16
5.1.1.5	Valid constraint commands	16
6	Example - A Small Example Program	19
7	Deprecated List	21
8	Module Index	29
8.1	Modules	29

9	Data Structure Index	31
9.1	Data Structures	31
10	File Index	33
10.1	File List	33
11	Module Documentation	37
11.1	RNA Secondary Structure Prediction	37
11.1.1	Detailed Description	39
11.2	Inverse Secondary Structure Prediction	40
11.2.1	Detailed Description	40
11.2.2	Function Documentation	40
11.2.2.1	inverse_fold	40
11.2.2.2	inverse_pf_fold	40
11.2.3	Variable Documentation	41
11.2.3.1	final_cost	41
11.2.3.2	give_up	41
11.2.3.3	inv_verbose	41
11.3	Free Energy Evaluation for given Sequence / Structure Pairs	42
11.3.1	Detailed Description	43
11.3.2	Function Documentation	43
11.3.2.1	vrna_eval_structure	43
11.3.2.2	vrna_eval_covar_structure	44
11.3.2.3	vrna_eval_structure_simple	44
11.3.2.4	vrna_eval_structure_verbose	44
11.3.2.5	vrna_eval_structure_simple_verbose	45
11.3.2.6	vrna_eval_structure_pt	45
11.3.2.7	vrna_eval_structure_pt_simple	46
11.3.2.8	vrna_eval_structure_pt_verbose	46
11.3.2.9	vrna_eval_structure_pt_simple_verbose	47
11.3.2.10	vrna_eval_loop_pt	47
11.3.2.11	vrna_eval_move	47
11.3.2.12	vrna_eval_move_pt	48
11.3.2.13	energy_of_structure	48
11.3.2.14	energy_of_struct_par	49
11.3.2.15	energy_of_circ_structure	49
11.3.2.16	energy_of_circ_struct_par	50
11.3.2.17	energy_of_structure_pt	50
11.3.2.18	energy_of_struct_pt_par	51
11.3.2.19	energy_of_move	51
11.3.2.20	energy_of_move_pt	52

11.3.2.21	loop_energy	52
11.3.2.22	energy_of_struct	53
11.3.2.23	energy_of_struct_pt	53
11.3.2.24	energy_of_circ_struct	54
11.3.2.25	vrna_eval_hp_loop	54
11.4	Processing and Evaluating Decomposed Loops	55
11.4.1	Detailed Description	56
11.4.2	Function Documentation	56
11.4.2.1	E_ExtLoop	56
11.4.2.2	exp_E_ExtLoop	56
11.4.2.3	E_Stem	56
11.4.2.4	exp_E_Stem	57
11.4.2.5	get_gquad_matrix	58
11.4.2.6	parse_gquad	58
11.4.2.7	backtrack_GQuad_IntLoop	58
11.4.2.8	backtrack_GQuad_IntLoop_L	58
11.4.2.9	E_Hairpin	59
11.4.2.10	exp_E_Hairpin	60
11.4.2.11	vrna_E_hp_loop	60
11.4.2.12	vrna_exp_E_hp_loop	61
11.4.2.13	vrna_BT_hp_loop	61
11.4.2.14	E_IntLoop	61
11.4.2.15	exp_E_IntLoop	62
11.4.2.16	E_mb_loop_stack	62
11.4.2.17	vrna_BT_mb_loop	63
11.5	Energy Parameter Sets and Boltzmann Factors	65
11.5.1	Detailed Description	66
11.5.2	Function Documentation	66
11.5.2.1	vrna_exp_params_update	66
11.5.2.2	vrna_exp_params_rescale	66
11.5.2.3	vrna_params_get	67
11.5.2.4	vrna_params_copy	67
11.5.2.5	vrna_exp_params_get	68
11.5.2.6	vrna_exp_params_ali_get	68
11.5.2.7	vrna_exp_params_copy	69
11.5.2.8	get_scaled_pf_parameters	69
11.5.2.9	get_boltzmann_factors	69
11.5.2.10	get_boltzmann_factor_copy	70
11.5.2.11	get_scaled_alipf_parameters	70
11.5.2.12	get_boltzmann_factors_ali	70

11.5.2.13	scale_parameters	70
11.5.2.14	get_scaled_parameters	71
11.6	Data Structures and Preprocessor Macros	72
11.6.1	Detailed Description	73
11.7	Utilities	74
11.7.1	Detailed Description	76
11.7.2	Macro Definition Documentation	76
11.7.2.1	VRNA_INPUT_FASTA_HEADER	76
11.7.2.2	VRNA_INPUT_CONSTRAINT	77
11.7.2.3	VRNA_OPTION_MULTILINE	77
11.7.2.4	FILENAME_MAX_LENGTH	77
11.7.2.5	FILENAME_ID_LENGTH	77
11.7.3	Function Documentation	77
11.7.3.1	vrna_alloc	77
11.7.3.2	vrna_realloc	77
11.7.3.3	vrna_message_error	78
11.7.3.4	vrna_message_warning	78
11.7.3.5	vrna_urn	78
11.7.3.6	vrna_int_urn	78
11.7.3.7	vrna_time_stamp	79
11.7.3.8	vrna_random_string	79
11.7.3.9	vrna_hamming_distance	79
11.7.3.10	vrna_hamming_distance_bound	79
11.7.3.11	get_line	79
11.7.3.12	get_input_line	80
11.7.3.13	vrna_message_input_seq_simple	80
11.7.3.14	vrna_message_input_seq	80
11.7.3.15	vrna_seq_toRNA	80
11.7.3.16	vrna_seq_toupper	81
11.7.3.17	vrna_get_iindx	81
11.7.3.18	vrna_get_indx	81
11.7.3.19	vrna_nucleotide_encode	82
11.7.3.20	vrna_nucleotide_decode	82
11.7.3.21	vrna_get_ptypes	82
11.7.4	Variable Documentation	83
11.7.4.1	xsubi	83
11.8	Computing Minimum Free Energy (MFE) Structures	84
11.8.1	Detailed Description	85
11.8.2	Function Documentation	85
11.8.2.1	vrna_fold	85

11.8.2.2	fold_par	85
11.8.2.3	fold	86
11.8.2.4	circfold	87
11.8.2.5	free_arrays	87
11.8.2.6	update_fold_params	87
11.8.2.7	update_fold_params_par	87
11.8.2.8	export_fold_arrays	87
11.8.2.9	export_fold_arrays_par	87
11.8.2.10	export_circfold_arrays	88
11.8.2.11	export_circfold_arrays_par	88
11.9	Computing Partition Functions and Pair Probabilities	89
11.9.1	Detailed Description	90
11.9.2	Function Documentation	91
11.9.2.1	vrna_pf_fold	91
11.9.2.2	vrna_mean_bp_distance_pr	91
11.9.2.3	vrna_mean_bp_distance	91
11.9.2.4	vrna_stack_prob	92
11.9.2.5	pf_fold_par	92
11.9.2.6	pf_fold	93
11.9.2.7	pf_circ_fold	94
11.9.2.8	free_pf_arrays	94
11.9.2.9	update_pf_params	95
11.9.2.10	update_pf_params_par	95
11.9.2.11	export_bppm	95
11.9.2.12	get_pf_arrays	96
11.9.2.13	mean_bp_distance	96
11.9.2.14	mean_bp_distance_pr	96
11.9.2.15	vrna_pl_get_from_pr	97
11.9.2.16	assign_plist_from_pr	97
11.10	Compute the structure with maximum expected accuracy (MEA)	98
11.11	Compute the centroid structure	99
11.11.1	Detailed Description	99
11.11.2	Function Documentation	99
11.11.2.1	vrna_get_centroid_struct_pl	99
11.11.2.2	vrna_get_centroid_struct_pr	99
11.11.2.3	vrna_get_centroid_struct	100
11.12	Enumerating Suboptimal Structures	101
11.12.1	Detailed Description	101
11.13	Suboptimal structures according to Zuker et al. 1989	102
11.13.1	Detailed Description	102

11.13.2 Function Documentation	102
11.13.2.1 zuckersubopt	102
11.13.2.2 zuckersubopt_par	102
11.13.2.3 vrna_zuckersubopt	103
11.14 Suboptimal structures within an energy band around the MFE	104
11.14.1 Detailed Description	104
11.14.2 Function Documentation	104
11.14.2.1 subopt	104
11.14.2.2 subopt_circ	105
11.15 Stochastic backtracking in the Ensemble	106
11.15.1 Detailed Description	106
11.15.2 Function Documentation	106
11.15.2.1 vrna_pbacktrack5	106
11.15.2.2 vrna_pbacktrack	107
11.15.2.3 pbacktrack	107
11.15.2.4 pbacktrack_circ	107
11.15.3 Variable Documentation	108
11.15.3.1 st_back	108
11.16 Calculate Secondary Structures of two RNAs upon Dimerization	109
11.16.1 Detailed Description	109
11.17 MFE Structures of two hybridized Sequences	110
11.17.1 Detailed Description	111
11.17.2 Function Documentation	111
11.17.2.1 cofold	111
11.17.2.2 cofold_par	111
11.17.2.3 vrna_cofold	111
11.17.2.4 vrna_cut_point_insert	111
11.17.2.5 vrna_cut_point_remove	112
11.17.2.6 free_co_arrays	112
11.17.2.7 export_cofold_arrays_gq	112
11.17.2.8 export_cofold_arrays	113
11.18 Partition Function for two hybridized Sequences	114
11.18.1 Detailed Description	114
11.18.2 Function Documentation	115
11.18.2.1 vrna_co_pf_fold	115
11.18.2.2 vrna_co_pf_dimer_probs	115
11.18.2.3 vrna_co_pf_get_concentrations	115
11.19 Partition Function for two hybridized Sequences as a stepwise Process	117
11.19.1 Detailed Description	117
11.19.2 Function Documentation	117

11.19.2.1 pf_unstru	117
11.19.2.2 pf_interact	118
11.20 Predicting Consensus Structures from Alignment(s)	119
11.20.1 Detailed Description	120
11.20.2 Function Documentation	120
11.20.2.1 energy_of_alistruct	120
11.20.2.2 get_alipf_arrays	120
11.20.2.3 update_alifold_params	121
11.20.2.4 vrna_ali_get_mpi	121
11.20.2.5 get_mpi	122
11.20.2.6 encode_ali_sequence	122
11.20.2.7 alloc_sequence_arrays	122
11.20.2.8 free_sequence_arrays	123
11.20.3 Variable Documentation	123
11.20.3.1 cv_fact	123
11.20.3.2 nc_fact	123
11.21 MFE Consensus Structures for Sequence Alignment(s)	124
11.21.1 Detailed Description	124
11.21.2 Function Documentation	124
11.21.2.1 vrna_ali_fold	124
11.21.2.2 alifold	125
11.21.2.3 circalifold	125
11.21.2.4 free_alifold_arrays	126
11.22 Partition Function and Base Pair Probabilities for Sequence Alignment(s)	127
11.22.1 Detailed Description	127
11.22.2 Function Documentation	127
11.22.2.1 vrna_ali_pf_fold	127
11.22.2.2 alipf_fold_par	128
11.22.2.3 alipf_fold	128
11.22.2.4 alipf_circ_fold	128
11.22.2.5 export_ali_bppm	129
11.22.2.6 free_alipf_arrays	129
11.23 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	130
11.23.1 Detailed Description	130
11.23.2 Function Documentation	130
11.23.2.1 vrna_ali_pbacktrack	130
11.23.2.2 alipbacktrack	130
11.24 Predicting Locally stable structures of large sequences	132
11.24.1 Detailed Description	132
11.25 Local MFE structure Prediction and Z-scores	133

11.25.1 Detailed Description	133
11.25.2 Function Documentation	133
11.25.2.1 Lfold	133
11.25.2.2 Lfoldz	133
11.26 Partition functions for locally stable secondary structures	134
11.26.1 Detailed Description	134
11.26.2 Function Documentation	134
11.26.2.1 update_pf_paramsLP	134
11.26.2.2 pfl_fold	135
11.26.2.3 putoutpU_prob	135
11.26.2.4 putoutpU_prob_bin	135
11.27 Local MFE consensus structures for Sequence Alignments	137
11.27.1 Detailed Description	137
11.27.2 Function Documentation	137
11.27.2.1 aliLfold	137
11.28 Reading/Writing Energy Parameter Sets from/to File	138
11.28.1 Detailed Description	138
11.28.2 Function Documentation	138
11.28.2.1 read_parameter_file	138
11.28.2.2 write_parameter_file	138
11.29 Converting Energy Parameter Files	140
11.29.1 Detailed Description	140
11.29.2 Macro Definition Documentation	141
11.29.2.1 VRNA_CONVERT_OUTPUT_ALL	141
11.29.2.2 VRNA_CONVERT_OUTPUT_HP	141
11.29.2.3 VRNA_CONVERT_OUTPUT_STACK	141
11.29.2.4 VRNA_CONVERT_OUTPUT_MM_HP	141
11.29.2.5 VRNA_CONVERT_OUTPUT_MM_INT	141
11.29.2.6 VRNA_CONVERT_OUTPUT_MM_INT_1N	141
11.29.2.7 VRNA_CONVERT_OUTPUT_MM_INT_23	141
11.29.2.8 VRNA_CONVERT_OUTPUT_MM_MULTI	141
11.29.2.9 VRNA_CONVERT_OUTPUT_MM_EXT	141
11.29.2.10 VRNA_CONVERT_OUTPUT_DANGLE5	141
11.29.2.11 VRNA_CONVERT_OUTPUT_DANGLE3	141
11.29.2.12 VRNA_CONVERT_OUTPUT_INT_11	141
11.29.2.13 VRNA_CONVERT_OUTPUT_INT_21	142
11.29.2.14 VRNA_CONVERT_OUTPUT_INT_22	142
11.29.2.15 VRNA_CONVERT_OUTPUT_BULGE	142
11.29.2.16 VRNA_CONVERT_OUTPUT_INT	142
11.29.2.17 VRNA_CONVERT_OUTPUT_ML	142

11.29.2.18	VRNA_CONVERT_OUTPUT_MISC	142
11.29.2.19	VRNA_CONVERT_OUTPUT_SPECIAL_HP	142
11.29.2.20	VRNA_CONVERT_OUTPUT_VANILLA	142
11.29.2.21	VRNA_CONVERT_OUTPUT_NINIO	142
11.29.2.22	VRNA_CONVERT_OUTPUT_DUMP	142
11.29.3	Function Documentation	142
11.29.3.1	convert_parameter_file	142
11.30	Classified Dynamic Programming	144
11.30.1	Detailed Description	144
11.31	Distance based partitioning of the Secondary Structure Space	145
11.31.1	Detailed Description	145
11.32	Calculating MFE representatives of a Distance Based Partitioning	146
11.32.1	Detailed Description	146
11.32.2	Function Documentation	147
11.32.2.1	vrna_TwoD_fold	147
11.32.2.2	vrna_TwoD_backtrack5	147
11.32.2.3	get_TwoDfold_variables	148
11.32.2.4	destroy_TwoDfold_variables	148
11.32.2.5	TwoDfoldList	148
11.32.2.6	TwoDfold_backtrack_f5	149
11.33	Calculate Partition Functions of a Distance Based Partitioning	150
11.33.1	Detailed Description	150
11.33.2	Function Documentation	150
11.33.2.1	vrna_TwoD_pf_fold	150
11.34	Stochastic Backtracking of Structures from Distance Based Partitioning	152
11.34.1	Detailed Description	152
11.34.2	Function Documentation	152
11.34.2.1	vrna_TwoD_pbacktrack	152
11.34.2.2	vrna_TwoD_pbacktrack5	153
11.35	Compute the Density of States	154
11.35.1	Detailed Description	154
11.35.2	Variable Documentation	154
11.35.2.1	density_of_states	154
11.36	Constraining the Secondary Structure Recursions	155
11.36.1	Detailed Description	156
11.36.2	Macro Definition Documentation	156
11.36.2.1	VRNA_CONSTRAINT_DB_PIPE	156
11.36.2.2	VRNA_CONSTRAINT_DB_DOT	156
11.36.2.3	VRNA_CONSTRAINT_DB_X	156
11.36.2.4	VRNA_CONSTRAINT_DB_ANG_BRACK	157

11.36.2.5 VRNA_CONSTRAINT_DB_RND_BRACK	157
11.36.2.6 VRNA_CONSTRAINT_DB_INTRAMOL	157
11.36.2.7 VRNA_CONSTRAINT_DB_INTERMOL	157
11.36.2.8 VRNA_CONSTRAINT_DB_GQUAD	157
11.36.2.9 VRNA_CONSTRAINT_DB_ENFORCE_BP	157
11.36.2.10 VRNA_CONSTRAINT_DB	158
11.36.2.11 VRNA_CONSTRAINT_FILE	158
11.36.3 Function Documentation	158
11.36.3.1 vrna_message_constraint_options	158
11.36.3.2 vrna_message_constraints_all	158
11.36.3.3 vrna_add_constraints	159
11.37 Hard Constraints	160
11.37.1 Detailed Description	161
11.37.2 Function Documentation	161
11.37.2.1 vrna_hc_init	161
11.37.2.2 vrna_hc_add_up	161
11.37.2.3 vrna_hc_add_bp	161
11.37.2.4 vrna_hc_add_bp_nonspecific	162
11.37.2.5 vrna_hc_free	162
11.38 Soft Constraints	163
11.38.1 Detailed Description	164
11.38.2 Macro Definition Documentation	164
11.38.2.1 VRNA_OBJECTIVE_FUNCTION_QUADRATIC	164
11.38.2.2 VRNA_OBJECTIVE_FUNCTION_ABSOLUTE	164
11.38.2.3 VRNA_MINIMIZER_CONJUGATE_FR	164
11.38.2.4 VRNA_MINIMIZER_CONJUGATE_PR	165
11.38.2.5 VRNA_MINIMIZER_VECTOR_BFGS	165
11.38.2.6 VRNA_MINIMIZER_VECTOR_BFGS2	165
11.38.2.7 VRNA_MINIMIZER_STEEPEST_DESCENT	165
11.38.3 Typedef Documentation	165
11.38.3.1 progress_callback	165
11.38.4 Function Documentation	165
11.38.4.1 vrna_sc_init	165
11.38.4.2 vrna_sc_add_bp	166
11.38.4.3 vrna_sc_add_up	166
11.38.4.4 vrna_sc_remove	166
11.38.4.5 vrna_sc_free	166
11.38.4.6 vrna_sc_SHAPE_add_deigan	166
11.38.4.7 vrna_sc_SHAPE_add_deigan_ali	167
11.38.4.8 vrna_sc_SHAPE_add_zarringham	167

11.38.4.9 vrna_sc_SHAPE_to_pr	168
11.38.4.10 vrna_sc_minimize_pertubation	168
11.39 Generalized Soft Constraints	170
11.39.1 Detailed Description	170
11.39.2 Macro Definition Documentation	170
11.39.2.1 VRNA_SC_GEN_MFE	170
11.39.2.2 VRNA_SC_GEN_PF	171
11.39.3 Function Documentation	171
11.39.3.1 vrna_sc_add_f	171
11.39.3.2 vrna_sc_add_bt	171
11.39.3.3 vrna_sc_add_exp_f	171
11.39.3.4 vrna_sc_add_pre	172
11.39.3.5 vrna_sc_add_post	172
11.40 Basic Data Structures for Structure Prediction and Evaluation	173
11.40.1 Detailed Description	176
11.40.2 Macro Definition Documentation	176
11.40.2.1 VRNA_OPTION_MFE	176
11.40.2.2 VRNA_OPTION_PF	176
11.40.2.3 VRNA_OPTION_EVAL_ONLY	176
11.40.2.4 VRNA_MODEL_DEFAULT_TEMPERATURE	177
11.40.2.5 VRNA_MODEL_DEFAULT_PF_SCALE	177
11.40.2.6 VRNA_MODEL_DEFAULT_BETA_SCALE	177
11.40.2.7 VRNA_MODEL_DEFAULT_DANGLES	177
11.40.2.8 VRNA_MODEL_DEFAULT_SPECIAL_HP	177
11.40.2.9 VRNA_MODEL_DEFAULT_NO_LP	177
11.40.2.10 VRNA_MODEL_DEFAULT_NO_GU	177
11.40.2.11 VRNA_MODEL_DEFAULT_NO_GU_CLOSURE	178
11.40.2.12 VRNA_MODEL_DEFAULT_CIRC	178
11.40.2.13 VRNA_MODEL_DEFAULT_GQUAD	178
11.40.2.14 VRNA_MODEL_DEFAULT_UNIQ_ML	178
11.40.2.15 VRNA_MODEL_DEFAULT_ENERGY_SET	178
11.40.2.16 VRNA_MODEL_DEFAULT_BACKTRACK	178
11.40.2.17 VRNA_MODEL_DEFAULT_BACKTRACK_TYPE	178
11.40.2.18 VRNA_MODEL_DEFAULT_COMPUTE_BPP	179
11.40.2.19 VRNA_MODEL_DEFAULT_MAX_BP_SPAN	179
11.40.2.20 VRNA_MODEL_DEFAULT_LOG_ML	179
11.40.2.21 VRNA_MODEL_DEFAULT_ALI_OLD_EN	179
11.40.2.22 VRNA_MODEL_DEFAULT_ALI_RIBO	179
11.40.2.23 VRNA_MODEL_DEFAULT_ALI_CV_FACT	179
11.40.2.24 VRNA_MODEL_DEFAULT_ALI_NC_FACT	179

11.40.3 Enumeration Type Documentation	180
11.40.3.1 vrna_mx_t	180
11.40.3.2 vrna_vc_t	180
11.40.4 Function Documentation	180
11.40.4.1 vrna_get_fold_compound	180
11.40.4.2 vrna_get_fold_compound_aliases	181
11.40.4.3 vrna_params_update	182
11.40.4.4 vrna_free_fold_compound	182
11.40.4.5 vrna_free_mfe_matrices	182
11.40.4.6 vrna_free_pf_matrices	182
11.40.4.7 vrna_md_set_default	183
11.40.4.8 vrna_md_update	183
11.40.4.9 vrna_md_set_globals	183
11.40.5 Variable Documentation	183
11.40.5.1 temperature	183
11.40.5.2 pf_scale	183
11.40.5.3 dangles	184
11.40.5.4 tetra_loop	184
11.40.5.5 noLonelyPairs	184
11.40.5.6 canonicalBPonly	184
11.40.5.7 energy_set	184
11.40.5.8 do_backtrack	184
11.40.5.9 backtrack_type	184
11.40.5.10 nonstandards	184
11.40.5.11 max_bp_span	185
11.41 Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures	186
11.41.1 Detailed Description	188
11.41.2 Function Documentation	188
11.41.2.1 b2HIT	188
11.41.2.2 b2C	188
11.41.2.3 b2Shapiro	189
11.41.2.4 add_root	189
11.41.2.5 expand_Shapiro	189
11.41.2.6 expand_Full	189
11.41.2.7 unexpand_Full	190
11.41.2.8 unweight	191
11.41.2.9 unexpand_aligned_F	191
11.41.2.10 parse_structure	191
11.41.2.11 vrna_db_pack	191
11.41.2.12 vrna_db_unpack	192

11.41.2.13	vrna_pt_get	192
11.41.2.14	vrna_pt_pk_get	192
11.41.2.15	vrna_pt_copy	192
11.41.2.16	vrna_pt_snoop_get	193
11.41.2.17	vrna_pt_to_db	193
11.41.2.18	vrna_bp_distance	193
11.41.2.19	vrna_refBPcnt_matrix	193
11.41.2.20	vrna_refBPdist_matrix	193
11.41.2.21	vrna_parenthesis_zuker	194
11.41.2.22	vrna_pl_get	194
11.41.2.23	vrna_pl_to_db	194
11.41.2.24	assign_plist_from_db	194
11.41.2.25	pack_structure	195
11.41.2.26	unpack_structure	195
11.41.2.27	make_pair_table	195
11.41.2.28	copy_pair_table	196
11.41.2.29	alimake_pair_table	196
11.41.2.30	make_pair_table_snoop	196
11.41.2.31	bp_distance	196
11.41.2.32	make_referenceBP_array	196
11.41.2.33	compute_BPdifferences	197
11.41.2.34	parenthesis_structure	197
11.41.2.35	parenthesis_zuker	197
11.41.2.36	bppm_to_structure	197
11.41.2.37	bppm_symbol	197
11.42	Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments	198
11.42.1	Detailed Description	199
11.42.2	Function Documentation	199
11.42.2.1	vrna_structure_print_hx	199
11.42.2.2	vrna_structure_print_ct	200
11.42.2.3	vrna_structure_print_bpseq	200
11.42.2.4	vrna_read_fasta_record	200
11.42.2.5	vrna_extract_record_rest_constraint	201
11.42.2.6	vrna_read_SHAPE_file	202
11.42.2.7	vrna_read_constraints_file	202
11.42.2.8	read_record	202
11.43	Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More	203
11.43.1	Detailed Description	204
11.43.2	Macro Definition Documentation	204
11.43.2.1	VRNA_PLOT_TYPE_SIMPLE	204

11.43.2.2 VRNA_PLOT_TYPE_NAVIEW	204
11.43.2.3 VRNA_PLOT_TYPE_CIRCULAR	204
11.43.3 Function Documentation	205
11.43.3.1 simple_xy_coordinates	205
11.43.3.2 simple_circplot_coordinates	205
11.43.3.3 PS_rna_plot	205
11.43.3.4 PS_rna_plot_a	206
11.43.3.5 gmlRNA	206
11.43.3.6 ssv_rna_plot	206
11.43.3.7 svg_rna_plot	207
11.43.3.8 xrna_plot	207
11.43.3.9 PS_dot_plot_list	207
11.43.3.10aliPS_color_aln	208
11.43.3.11PS_dot_plot	208
11.43.4 Variable Documentation	208
11.43.4.1 rna_plot_type	208
12 Data Structure Documentation	209
12.1 bondT Struct Reference	209
12.1.1 Detailed Description	209
12.2 bondTEn Struct Reference	209
12.2.1 Detailed Description	209
12.3 cofoldF Struct Reference	209
12.4 ConcEnt Struct Reference	210
12.5 constrain Struct Reference	210
12.5.1 Detailed Description	210
12.6 COORDINATE Struct Reference	210
12.6.1 Detailed Description	210
12.7 cpair Struct Reference	210
12.7.1 Detailed Description	211
12.8 duplexT Struct Reference	211
12.9 dupVar Struct Reference	211
12.10folden Struct Reference	211
12.11interact Struct Reference	211
12.12intermediate_t Struct Reference	212
12.13INTERVAL Struct Reference	212
12.13.1 Detailed Description	212
12.14LIST Struct Reference	213
12.15LST_BUCKET Struct Reference	213
12.16move_t Struct Reference	213

12.17PAIR Struct Reference	213
12.17.1 Detailed Description	214
12.18pair_info Struct Reference	214
12.18.1 Detailed Description	214
12.19pairpro Struct Reference	215
12.20path_t Struct Reference	215
12.21plist Struct Reference	215
12.21.1 Detailed Description	215
12.22Postorder_list Struct Reference	215
12.23pu_contrib Struct Reference	216
12.23.1 Detailed Description	216
12.24pu_out Struct Reference	216
12.24.1 Detailed Description	216
12.25sect Struct Reference	217
12.25.1 Detailed Description	217
12.26snoopT Struct Reference	217
12.27SOLUTION Struct Reference	217
12.27.1 Detailed Description	217
12.28struct_en Struct Reference	217
12.29svm_model Struct Reference	217
12.30swString Struct Reference	218
12.31Tree Struct Reference	218
12.32TwoDfold_vars Struct Reference	218
12.32.1 Detailed Description	219
12.33TwoDpfold_vars Struct Reference	219
12.33.1 Detailed Description	220
12.34vrna_exp_param_t Struct Reference	221
12.34.1 Detailed Description	221
12.34.2 Field Documentation	221
12.34.2.1 id	221
12.34.2.2 alpha	222
12.35vrna_fold_compound Struct Reference	222
12.35.1 Detailed Description	224
12.35.2 Field Documentation	224
12.35.2.1 type	224
12.35.2.2 sequence	224
12.35.2.3 sequence_encoding	224
12.35.2.4 ptype	225
12.35.2.5 ptype_pf_compat	225
12.35.2.6 sc	225

12.35.2.7 sequences	225
12.35.2.8 n_seq	226
12.35.2.9 cons_seq	226
12.35.2.10 S_cons	226
12.35.2.11 S	226
12.35.2.12 S5	226
12.35.2.13 S3	226
12.35.2.14 p_score	227
12.35.2.15 scs	227
12.36 vrna_hc_t Struct Reference	227
12.36.1 Detailed Description	227
12.37 vrna_helix Struct Reference	228
12.38 vrna_md_t Struct Reference	228
12.38.1 Detailed Description	229
12.38.2 Field Documentation	229
12.38.2.1 dangles	229
12.38.2.2 min_loop_size	230
12.39 vrna_mx_mfe_t Struct Reference	230
12.39.1 Detailed Description	232
12.40 vrna_mx_pf_t Struct Reference	232
12.40.1 Detailed Description	233
12.41 vrna_param_t Struct Reference	233
12.41.1 Detailed Description	234
12.42 vrna_sc_t Struct Reference	234
12.42.1 Detailed Description	235
12.42.2 Field Documentation	235
12.42.2.1 f	235
12.42.2.2 bt	235
12.42.2.3 exp_f	236
12.42.2.4 pre	236
12.42.2.5 post	236
12.43 vrna_sol_TwoD_pf_t Struct Reference	236
12.43.1 Detailed Description	236
12.44 vrna_sol_TwoD_t Struct Reference	237
12.44.1 Detailed Description	237
13 File Documentation	239
13.1 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_epars.h File Reference	239
13.1.1 Detailed Description	239
13.2 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_intloops.h File Reference	239

13.2.1 Detailed Description	239
13.3 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dfold.h File Reference	240
13.4 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dpfold.h File Reference	241
13.4.1 Function Documentation	242
13.4.1.1 get_TwoDpfold_variables	242
13.4.1.2 destroy_TwoDpfold_variables	242
13.4.1.3 TwoDpfoldList	243
13.4.1.4 TwoDpfold_pbacktrack	243
13.4.1.5 TwoDpfold_pbacktrack5	244
13.5 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/alifold.h File Reference	244
13.5.1 Detailed Description	246
13.5.2 Function Documentation	246
13.5.2.1 vrna_alifold_get_pair_info	246
13.6 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/centroid.h File Reference	247
13.6.1 Detailed Description	248
13.6.2 Function Documentation	248
13.6.2.1 get_centroid_struct_pl	248
13.6.2.2 get_centroid_struct_pr	248
13.7 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/cofold.h File Reference	248
13.7.1 Detailed Description	250
13.7.2 Function Documentation	250
13.7.2.1 get_monomere_mfes	250
13.7.2.2 initialize_cofold	250
13.8 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/convert_epars.h File Reference	250
13.8.1 Detailed Description	251
13.9 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h File Reference	251
13.10/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/dist_vars.h File Reference	253
13.10.1 Detailed Description	254
13.10.2 Variable Documentation	254
13.10.2.1 edit_backtrack	254
13.10.2.2 cost_matrix	254
13.11/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/duplex.h File Reference	255
13.11.1 Detailed Description	255
13.12/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/edit_cost.h File Reference	255
13.12.1 Detailed Description	255
13.13/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/energy_const.h File Reference	256
13.13.1 Detailed Description	256
13.13.2 Macro Definition Documentation	256
13.13.2.1 GASCONST	256
13.13.2.2 K0	256

13.13.2.3 INF	257
13.13.2.4 FORBIDDEN	257
13.13.2.5 BONUS	257
13.13.2.6 NBPAIRS	257
13.13.2.7 TURN	257
13.13.2.8 MAXLOOP	257
13.14/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/eval.h File Reference	257
13.14.1 Detailed Description	260
13.15/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/exterior_loops.h File Reference	260
13.15.1 Detailed Description	260
13.16/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/file_formats.h File Reference	261
13.16.1 Detailed Description	262
13.17/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/findpath.h File Reference	262
13.17.1 Detailed Description	263
13.17.2 Function Documentation	263
13.17.2.1 find_saddle	263
13.17.2.2 get_path	263
13.17.2.3 free_path	263
13.18/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/fold.h File Reference	263
13.18.1 Detailed Description	265
13.18.2 Function Documentation	265
13.18.2.1 LoopEnergy	265
13.18.2.2 HairpinE	265
13.18.2.3 initialize_fold	265
13.19/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/fold_vars.h File Reference	265
13.19.1 Detailed Description	267
13.19.2 Variable Documentation	267
13.19.2.1 RibosumFile	267
13.19.2.2 james_rule	267
13.19.2.3 logML	267
13.19.2.4 cut_point	267
13.19.2.5 base_pair	267
13.19.2.6 pr	267
13.19.2.7 iindx	268
13.20/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/gquad.h File Reference	268
13.20.1 Detailed Description	269
13.21/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/hairpin_loops.h File Reference	269
13.21.1 Detailed Description	270
13.22/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/interior_loops.h File Reference	270
13.22.1 Detailed Description	271

13.23/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/inverse.h File Reference	272
13.23.1 Detailed Description	272
13.24/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/Lfold.h File Reference	272
13.24.1 Detailed Description	272
13.25/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/loop_energies.h File Reference	272
13.25.1 Detailed Description	273
13.26/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/LPfold.h File Reference	273
13.26.1 Detailed Description	274
13.26.2 Function Documentation	274
13.26.2.1 init_pf_foldLP	275
13.27/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/MEA.h File Reference	275
13.27.1 Detailed Description	275
13.27.2 Function Documentation	275
13.27.2.1 MEA	276
13.28/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/mm.h File Reference	276
13.28.1 Detailed Description	276
13.29/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/model.h File Reference	276
13.29.1 Detailed Description	278
13.30/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/multibranch_loops.h File Reference	279
13.30.1 Detailed Description	279
13.31/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/naview.h File Reference	280
13.32/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/params.h File Reference	280
13.33/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func.h File Reference	281
13.33.1 Detailed Description	283
13.33.2 Function Documentation	283
13.33.2.1 stackProb	283
13.33.2.2 init_pf_fold	284
13.33.2.3 centroid	284
13.33.2.4 get_centroid_struct_gquad_pr	284
13.33.2.5 mean_bp_dist	284
13.33.2.6 expLoopEnergy	284
13.33.2.7 expHairpinEnergy	284
13.34/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_co.h File Reference	284
13.34.1 Detailed Description	286
13.34.2 Function Documentation	286
13.34.2.1 co_pf_fold	286
13.34.2.2 co_pf_fold_par	287
13.34.2.3 get_plist	287
13.34.2.4 compute_probabilities	287
13.34.2.5 get_concentrations	288

13.34.2.6 init_co_pf_fold	288
13.34.2.7 export_co_bppm	288
13.34.2.8 update_co_pf_params	289
13.34.2.9 update_co_pf_params_par	289
13.35/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_up.h File Reference	289
13.35.1 Detailed Description	290
13.36/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/plot_layouts.h File Reference	291
13.36.1 Detailed Description	292
13.37/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/profiledist.h File Reference	293
13.37.1 Function Documentation	293
13.37.1.1 profile_edit_distance	293
13.37.1.2 Make_bp_profile_bppm	294
13.37.1.3 free_profile	294
13.37.1.4 Make_bp_profile	294
13.38/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/PS_dot.h File Reference	294
13.38.1 Detailed Description	295
13.39/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/read_epars.h File Reference	296
13.40/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/ribo.h File Reference	296
13.40.1 Detailed Description	296
13.41/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/RNAstruct.h File Reference	296
13.41.1 Detailed Description	297
13.42/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/stringdist.h File Reference	298
13.42.1 Detailed Description	298
13.42.2 Function Documentation	298
13.42.2.1 Make_swString	298
13.42.2.2 string_edit_distance	298
13.43/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/structure_utils.h File Reference	299
13.43.1 Detailed Description	301
13.44/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/subopt.h File Reference	302
13.44.1 Detailed Description	303
13.45/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/treedist.h File Reference	303
13.45.1 Detailed Description	303
13.45.2 Function Documentation	303
13.45.2.1 make_tree	303
13.45.2.2 tree_edit_distance	304
13.45.2.3 free_tree	304
13.46/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/Utils.h File Reference	304
13.46.1 Detailed Description	308
13.46.2 Function Documentation	308
13.46.2.1 str_uppercase	308

13.46.2.2 str_DNA2RNA	309
13.46.2.3 print_tty_input_seq	309
13.46.2.4 print_tty_input_seq_str	309
13.46.2.5 warn_user	309
13.46.2.6 nrerror	309
13.46.2.7 space	309
13.46.2.8 xrealloc	309
13.46.2.9 init_rand	310
13.46.2.10 urn	310
13.46.2.11 int_urn	310
13.46.2.12 random_string	310
13.46.2.13 filecopy	310
13.46.2.14 time_stamp	310
13.46.2.15 hamming	310
13.46.2.16 hamming_bound	310
Bibliography	311
Index	313

Chapter 1

ViennaRNA Package core - RNALib

A Library for folding and comparing RNA secondary structures

Date

1994-2015

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](#)
- [Example - A Small Example Program](#)

1.1 Introduction

The core of the Vienna RNA Package ([8], [6]) 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/~ivo/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 [4] 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 [12], 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, 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 [13]. 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 then 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[])
```

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 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)
```

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

```
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)
```

```
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 themselves 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/utils.h>
#include <ViennaRNA/eval.h>
#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>

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

    char *seq = "
        AGACGACAAGGUUGAAUCGCACCCACAGUCUAUGAGUCGGUGACAACAUAUACGAAAGGCUGUAAAAUCAUUUAUUCACCACAGGGGGCCCCGUGUCUAG";
    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 *vc = vrna_get_fold_compound(seq, NULL,
        VRNA_OPTION_MFE | VRNA_OPTION_PF);

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

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

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

    /* call PF function */
    FLT_OR_DBL en = vrna_pf_fold(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_get_centroid_struct(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_free_fold_compound(vc);

    return EXIT_SUCCESS;
}
```

Chapter 5

Input / Output File Formats

5.1 File formats for Secondary Structure Constraints

5.1.1 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.

5.1.1.1 Constraint commands

The following set of commands is recognized:

- `F ...` Force
- `P ...` Prohibit
- `W ...` Weakly enforce, i.e. remove conflicts only
- `A ...` Allow (for non-canonical pairs)
- `E ...` Soft constraints for unpaired position(s), or base pair(s)

5.1.1.2 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`)

5.1.1.3 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.

5.1.1.4 Sequence coordinates

Sequence positions of nucleotides/base pairs are 1-based and consist of three positions i , j , and k . Alternatively, 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 $k - l$.

5.1.1.5 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 $(i, j), \dots, (i + (k - 1), j - (k - 1))$ 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. "Prohibiting a set of consecutive base pairs to form":

Syntax:

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

Description:

Prohibit the base pairs $(i, j), \dots, (i + (k - 1), j - (k - 1))$ 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. "Weakly prohibit a range of nucleotide positions to be paired":

Syntax:

`W i 0 k [WHERE]`

Description:

This command is meant as a complement to *prohibiting* nucleotides to be paired, as described above. It too marks the corresponding nucleotides to be unpaired, however, they are not required to appear in the optional loop type context. They are rather prohibited from forming base pairs only. The optional loop type context specifier [WHERE] may be used to prohibit pairing in specific contexts.

7. "Weakly enforce a set of consecutive base pairs":

Syntax:

`W i j k`

Description:

Remove all base pairs that conflict with a set of consecutive base pairs $(i, j), \dots, (i + (k - 1), j - (k - 1))$. 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 $(i, j), \dots, (i + (k - 1), j - (k - 1))$, 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-canonical* 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 $(i, j), \dots, (i + (k - 1), j - (k - 1))$. Energies are expected to be given in *kcal/mol*.

Chapter 6

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="CGCAGGGGAUACCCGCG", *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 omitted when compiling example.c

Chapter 7

Deprecated List

globalScope> Global [alifold](#) (const char **strings, char *structure)

Usage of this function is discouraged! Use [vrna_ali_fold\(\)](#) instead

See also

[vrna_ali_fold\(\)](#)

globalScope> Global [alimake_pair_table](#) (const char *structure)

Use [vrna_pt_ali_get\(\)](#) instead!

globalScope> Global [alipbacktrack](#) (double *prob)

Use [vrna_ali_pbacktrack\(\)](#) instead!

globalScope> Global [alipf_circ_fold](#) (const char **sequences, char *structure, plist **pl)

Use [vrna_ali_pf_fold\(\)](#) instead

globalScope> Global [alipf_fold](#) (const char **sequences, char *structure, plist **pl)

Use [vrna_ali_pf_fold\(\)](#) instead

globalScope> Global [alipf_fold_par](#) (const char **sequences, char *structure, plist **pl, [vrna_exp_↔param_t](#) *parameters, int calculate_bppm, int is_constrained, int is_circular)

Use [vrna_ali_pf_fold\(\)](#) instead

globalScope> Global [assign_plist_from_db](#) (plist **pl, const char *struc, float pr)

Use [vrna_pl_get\(\)](#) instead

globalScope> Global [assign_plist_from_pr](#) (plist **pl, FLT_OR_DBL *probs, int length, double cutoff)

Use [vrna_pl_get_from_pr\(\)](#) instead!

globalScope> Global [base_pair](#)

Do not use this variable anymore!

globalScope> Global [bp_distance](#) (const char *str1, const char *str2)

Use [vrna_bp_distance](#) instead

Parameters

<i>str1</i>	First structure in dot-bracket notation
<i>str2</i>	Second structure in dot-bracket notation

Returns

The base pair distance between str1 and str2

globalScope> Global [bppm_symbol](#) (const float *x)

Use [vrna_bpp_symbol\(\)](#) instead!

globalScope> Global [bppm_to_structure](#) (char *structure, FLT_OR_DBL *pr, unsigned int length)

Use [vrna_db_get_from_pr\(\)](#) instead!

globalScope> Global [centroid](#) (int length, double *dist)

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\(\)](#)

globalScope> Global [circularfold](#) (const char **strings, char *structure)

Usage of this function is discouraged! Use [vrna_ali_fold\(\)](#) instead

See also

[vrna_ali_fold\(\)](#)

globalScope> Global [circfold](#) (const char *sequence, char *structure)

Use [vrna_fold\(\)](#) instead!

globalScope> Global [co_pf_fold](#) (char *sequence, char *structure)

{Use [vrna_co_pf_fold\(\)](#) instead!}

globalScope> Global [cofold](#) (const char *sequence, char *structure)

use [vrna_cofold\(\)](#) instead

**globalScope> Global [cofold_par](#) (const char *string, char *structure, [vrna_param_t](#) *parameters, int is_↵
constrained)**

use [vrna_cofold\(\)](#) instead

globalScope> Global [compute_BPdifferences](#) (short *pt1, short *pt2, unsigned int turn)

Use [vrna_refBPdist_matrix\(\)](#) instead

**globalScope> Global [compute_probabilities](#) (double FAB, double FEA, double FEB, struct plist *prAB,
struct plist *prA, struct plist *prB, int Alength)**

{ Use [vrna_co_pf_dimer_probs\(\)](#) instead!}

globalScope> Global [copy_pair_table](#) (const short *pt)

Use [vrna_pt_copy\(\)](#) instead

globalScope> Global [cv_fact](#)

See [vrna_md_t.cv_fact](#), and [vrna_ali_fold\(\)](#) to avoid using global variables

globalScope> Global [destroy_TwoDfold_variables](#) ([TwoDfold_vars](#) *our_variables)

Use the new API that relies on [vrna_fold_compound](#) and the corresponding functions [vrna_get_fold_↵
compound_2D\(\)](#), [vrna_TwoD_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

globalScope> Global [destroy_TwoDpfold_variables](#) ([TwoDpfold_vars](#) *vars)

Use the new API that relies on [vrna_fold_compound](#) and the corresponding functions [vrna_get_fold_↵
compound_2D\(\)](#), [vrna_TwoD_pf_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

**globalScope> 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!

globalScope> 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!

**globalScope> 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!

**globalScope> Global [energy_of_circ_structure](#) (const char *string, const char *structure, int verbosity_↵
level)**

Use [vrna_eval_structure\(\)](#) or [vrna_eval_structure_verbose\(\)](#) instead!

globalScope> Global `energy_of_move` (const char *string, const char *structure, int m1, int m2)
 Use `vrna_eval_move()` instead!

globalScope> Global `energy_of_move_pt` (short *pt, short *s, short *s1, int m1, int m2)
 Use `vrna_eval_move_pt()` instead!

globalScope> 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!

globalScope> 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!

globalScope> 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!

globalScope> 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!

globalScope> Global `energy_of_structure` (const char *string, const char *structure, int verbosity_level)
 Use `vrna_eval_structure()` or `vrna_eval_structure_verbose()` instead!

globalScope> 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!

globalScope> Global `expHairpinEnergy` (int u, int type, short si1, short sj1, const char *string)
 Use `exp_E_Hairpin()` from `loop_energies.h` instead

globalScope> 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

globalScope> Global `export_ali_bppm` (void)
 Usage of this function is discouraged! The new `vrna_fold_compound` 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_ali_pf_fold()`, or any of the old API calls for consensus structure partition function folding.

globalScope> Global `export_circfold_arrays` (int *Fc_p, int *FcH_p, int *Fcl_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_fold()` and `vrna_fold_compound` for the usage of the new API!

globalScope> Global `export_circfold_arrays_par` (int *Fc_p, int *FcH_p, int *Fcl_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_fold()` and `vrna_fold_compound` for the usage of the new API!

globalScope> 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 fold compound. Thus, this function will only work in conjunction with a prior call to `cofold()` or `cofold_par()`

globalScope> Global `export_cofold_arrays_gg` (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()`

globalScope> Global `export_fold_arrays` (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p)
 See `vrna_fold()` and `vrna_fold_compound` for the usage of the new API!

globalScope> 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_fold()` and `vrna_fold_compound` for the usage of the new API!

globalScope> Global [filecopy](#) (FILE *from, FILE *to)

Use [vrna_file_copy\(\)](#) instead!

globalScope> Global [fold](#) (const char *sequence, char *structure)

use [vrna_fold\(\)](#) instead

**globalScope> Global [fold_par](#) (const char *sequence, char *structure, [vrna_param_t](#) *parameters, int is_↵
_constrained, int is_circular)**

use [vrna_fold\(\)](#) instead

globalScope> 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](#) is handled by [vrna_vrna_free_fold_compound\(\)](#)

globalScope> 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!

globalScope> Global [free_arrays](#) (void)

See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

globalScope> Global [free_co_arrays](#) (void)

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

globalScope> Global [free_pf_arrays](#) (void)

See [vrna_fold_compound](#) and its related functions for how to free memory occupied by the dynamic programming matrices

globalScope> 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, FL↵
T_OR_DBL **qln_p, int **pscore)**

It is discouraged to use this function! The new [vrna_fold_compound](#) allows direct access to all necessary consensus structure prediction related variables!

globalScope> Global [get_boltzmann_factor_copy](#) ([vrna_exp_param_t](#) *parameters)

Use [vrna_exp_params_copy\(\)](#) instead!

globalScope> Global [get_boltzmann_factors](#) (double temperature, double betaScale, [vrna_md_t](#) md, double pf_scale)

Use [vrna_exp_params_get\(\)](#) instead!

**globalScope> Global [get_boltzmann_factors_ali](#) (unsigned int n_seq, double temperature, double beta↵
Scale, [vrna_md_t](#) md, double pf_scale)**

Use [vrna_exp_params_ali_get\(\)](#) instead!

globalScope> 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!

See also

[vrna_get_centroid_struct\(\)](#), [vrna_get_centroid_struct_pr\(\)](#), [vrna_get_centroid_struct_pl\(\)](#)

globalScope> Global [get_centroid_struct_pl](#) (int length, double *dist, plist *pl)

This function was renamed to [vrna_get_centroid_struct_pl\(\)](#)

globalScope> Global [get_centroid_struct_pr](#) (int length, double *dist, FLT_OR_DBL *pr)

This function was renamed to [vrna_get_centroid_struct_pr\(\)](#)

globalScope> Global [get_concentrations](#) (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double *startconc)

{ Use [vrna_co_pf_get_concentrations\(\)](#) instead! }

globalScope> Global [get_mpi](#) (char *Aseq[], int n_seq, int length, int *mini)
 Use [vrna_ali_get_mpi\(\)](#) as a replacement

globalScope> Global [get_plist](#) (struct plist *pl, int length, double cut_off)
 { This function is deprecated and will be removed soon!} use [assign_plist_from_pr\(\)](#) instead!

globalScope> Global [get_scaled_alipf_parameters](#) (unsigned int n_seq)
 Use [vrna_exp_params_ali_get\(\)](#) instead!

globalScope> Global [get_scaled_parameters](#) (double temperature, [vrna_md_t](#) md)
 Use [vrna_params_get\(\)](#) instead!

globalScope> Global [get_scaled_pf_parameters](#) (void)
 Use [vrna_exp_params_get\(\)](#) instead!

globalScope> 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](#) and the corresponding functions [vrna_get_fold_compound_2D\(\)](#), [vrna_TwoD_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

globalScope> Global [get_TwoDpfold_variables](#) (const char *seq, const char *structure1, char *structure2, int circ)
 Use the new API that relies on [vrna_fold_compound](#) and the corresponding functions [vrna_get_fold_compound_2D\(\)](#), [vrna_TwoD_pf_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

globalScope> 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!}

globalScope> Global [hamming](#) (const char *s1, const char *s2)
 Use [vrna_hamming_distance\(\)](#) instead!

globalScope> Global [hamming_bound](#) (const char *s1, const char *s2, int n)
 Use [vrna_hamming_distance_bound\(\)](#) instead!

globalScope> Global [iindx](#)
 Do not use this variable anymore!

globalScope> Global [init_co_pf_fold](#) (int length)
 { This function is deprecated and will be removed soon!}

globalScope> Global [init_pf_fold](#) (int length)
 This function is obsolete and will be removed soon!

globalScope> Global [init_rand](#) (void)
 Use [vrna_init_rand\(\)](#) instead!

globalScope> Global [initialize_cofold](#) (int length)
 {This function is obsolete and will be removed soon!}

globalScope> Global [initialize_fold](#) (int length)
 {This function is deprecated and will be removed soon!}
 See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

globalScope> Global [int_urn](#) (int from, int to)
 Use [vrna_int_urn\(\)](#) instead!

globalScope> Global [loop_energy](#) (short *ptable, short *s, short *s1, int i)
 Use [vrna_eval_loop_pt\(\)](#) instead!

globalScope> 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!}

globalScope> Global [Make_bp_profile](#) (int length)
 This function is deprecated and will be removed soon! See [Make_bp_profile_bppm\(\)](#) for a replacement

globalScope> Global [make_pair_table](#) (const char *structure)

Use [vrna_pt_get\(\)](#) instead

globalScope> Global [make_pair_table_snoop](#) (const char *structure)

Use [vrna_pt_snoop_get\(\)](#) instead!

globalScope> Global [make_referenceBP_array](#) (short *reference_pt, unsigned int turn)

Use [vrna_refBPcnt_matrix\(\)](#) instead

globalScope> Global [mean_bp_dist](#) (int length)

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

globalScope> Global [mean_bp_distance](#) (int length)

Use [vrna_mean_bp_distance\(\)](#) or [vrna_mean_bp_distance_pr\(\)](#) instead!

See also

[vrna_mean_bp_distance\(\)](#), [vrna_mean_bp_distance_pr\(\)](#)

globalScope> Global [mean_bp_distance_pr](#) (int length, FLT_OR_DBL *pr)

Use [vrna_mean_bp_distance\(\)](#) or [vrna_mean_bp_distance_pr\(\)](#) instead!

globalScope> Global [nc_fact](#)

See `#vrna_md_t.nc_fact`, and [vrna_ali_fold\(\)](#) to avoid using global variables

globalScope> Global [nrerror](#) (const char message[])

Use [vrna_message_error\(\)](#) instead!

globalScope> Global [pack_structure](#) (const char *struc)

Use [vrna_db_pack\(\)](#) as a replacement

Parameters

<i>struc</i>	The secondary structure in dot-bracket notation
--------------	---

Returns

The binary encoded structure

globalScope> Global [parenthesis_structure](#) (char *structure, **bondT *bp, int length)**

use [vrna_parenthesis_structure\(\)](#) instead

globalScope> Global [parenthesis_zuker](#) (char *structure, **bondT *bp, int length)**

use [vrna_parenthesis_zuker](#) instead

globalScope> Global [pbacktrack_circ](#) (char *sequence)

Use [vrna_pbacktrack\(\)](#) instead.

globalScope> Global [pf_circ_fold](#) (const char *sequence, char *structure)

Use [vrna_pf_fold\(\)](#) instead!

Parameters

<i>in</i>	<i>sequence</i>	The RNA sequence input
<i>in, out</i>	<i>structure</i>	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

globalScope> 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_fold\(\)](#) instead

globalScope> Global [pr](#)

Do not use this variable anymore!

globalScope> Global `print_tty_input_seq` (void)

Use `vrna_message_input_seq_simple()` instead!

globalScope> Global `print_tty_input_seq_str` (const char *s)

Use `vrna_message_input_seq()` instead!

globalScope> Global `PS_dot_plot` (char *string, char *file)

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

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

Use `vrna_random_string()` instead!

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

This function is deprecated! Use `vrna_read_fasta_record()` as a replacement.

globalScope> Global `scale_parameters` (void)

Use `vrna_params_get()` instead!

globalScope> Global `space` (unsigned size)

Use `vrna_alloc()` instead!

globalScope> Global `st_back`

set the `uniq_ML` flag in `vrna_md_t` before passing it to `vrna_get_fold_compound()`.

globalScope> Global `stackProb` (double cutoff)

Use `vrna_stack_prob()` instead!

globalScope> Global `str_DNA2RNA` (char *sequence)

Use `vrna_seq_toRNA()` instead!

globalScope> Global `str_uppercase` (char *sequence)

Use `vrna_seq_toupper()` instead!

globalScope> Global `time_stamp` (void)

Use `vrna_time_stamp()` instead!

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

Use the new API that relies on `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_fold()`, `vrna_TwoD_backtrack5()`, and `vrna_free_fold_compound()` instead!

Class `TwoDfold_vars`

This data structure will be removed from the library soon! Use `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_fold()`, and `vrna_free_fold_compound()` instead!

globalScope> Global `TwoDfoldList` (`TwoDfold_vars` *vars, int distance1, int distance2)

Use the new API that relies on `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_fold()`, and `vrna_free_fold_compound()` instead!

globalScope> Global `TwoDpfold_pbacktrack` (`TwoDpfold_vars` *vars, int d1, int d2)

Use the new API that relies on `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, `vrna_TwoD_pbacktrack()`, and `vrna_free_fold_compound()` instead!

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

Use the new API that relies on `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, `vrna_TwoD_pbacktrack5()`, and `vrna_free_fold_compound()` instead!

Class `TwoDpfold_vars`

This data structure will be removed from the library soon! Use `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, and `vrna_free_fold_compound()` instead!

globalScope> Global `TwoDpfoldList` (`TwoDpfold_vars` *vars, int maxDistance1, int maxDistance2)

Use the new API that relies on `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, and `vrna_free_fold_compound()` instead!

globalScope> Global `unpack_structure` (const char *packed)

Use `vrna_db_unpack()` as a replacement

Parameters

<i>packed</i>	The binary encoded packed secondary structure
---------------	---

Returns

The unpacked secondary structure in dot-bracket notation

globalScope> Global [update_alifold_params](#) (void)

Usage of this function is discouraged! The new API uses [vrna_fold_compound](#) 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](#).

globalScope> Global [update_co_pf_params](#) (int length)

Use [vrna_exp_params_update\(\)](#) instead!

globalScope> Global [update_co_pf_params_par](#) (int length, [vrna_exp_param_t](#) *parameters)

Use [vrna_exp_params_update\(\)](#) instead!

globalScope> Global [update_fold_params](#) (void)

use [vrna_params_update\(\)](#) instead

globalScope> Global [update_fold_params_par](#) ([vrna_param_t](#) *parameters)

use [vrna_params_update\(\)](#) instead

globalScope> Global [update_pf_params](#) (int length)

Use [vrna_exp_params_update\(\)](#) instead

globalScope> Global [update_pf_params_par](#) (int length, [vrna_exp_param_t](#) *parameters)

Use [vrna_exp_params_update\(\)](#) instead

globalScope> Global [urn](#) (void)

Use [vrna_urn\(\)](#) instead!

Global [vrna_exp_param_t::id](#)

This attribute will be removed in version 3

Global [vrna_fold_compound::ptype_pf_compat](#)

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

Warning

Only available if

`type==VRNA_VC_TYPE_SINGLE`

globalScope> Global [vrna_md_set_globals](#) ([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!

globalScope> Global [warn_user](#) (const char message[])

Use [vrna_message_warning\(\)](#) instead!

globalScope> Global [xrealloc](#) (void *p, unsigned size)

Use [vrna_realloc\(\)](#) instead!

globalScope> Global [zukersubopt](#) (const char *string)

use [vrna_zukersubopt\(\)](#) instead

globalScope> Global [zukersubopt_par](#) (const char *string, [vrna_param_t](#) *parameters)

use [vrna_zukersubopt\(\)](#) instead

Chapter 8

Module Index

8.1 Modules

Here is a list of all modules:

RNA Secondary Structure Prediction	37
Computing Minimum Free Energy (MFE) Structures	84
MFE Structures of two hybridized Sequences	110
MFE Consensus Structures for Sequence Alignment(s)	124
Local MFE structure Prediction and Z-scores	133
Calculating MFE representatives of a Distance Based Partitioning	146
Computing Partition Functions and Pair Probabilities	89
Compute the structure with maximum expected accuracy (MEA)	98
Compute the centroid structure	99
Partition Function for two hybridized Sequences	114
Partition Function for two hybridized Sequences as a stepwise Process	117
Partition Function and Base Pair Probabilities for Sequence Alignment(s)	127
Partition functions for locally stable secondary structures	134
Calculate Partition Functions of a Distance Based Partitioning	150
Enumerating Suboptimal Structures	101
Suboptimal structures according to Zuker et al. 1989	102
Suboptimal structures within an energy band around the MFE	104
Stochastic backtracking in the Ensemble	106
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	130
Stochastic Backtracking of Structures from Distance Based Partitioning	152
Calculate Secondary Structures of two RNAs upon Dimerization	109
MFE Structures of two hybridized Sequences	110
Partition Function for two hybridized Sequences	114
Partition Function for two hybridized Sequences as a stepwise Process	117
Predicting Consensus Structures from Alignment(s)	119
MFE Consensus Structures for Sequence Alignment(s)	124
Partition Function and Base Pair Probabilities for Sequence Alignment(s)	127
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	130
Local MFE consensus structures for Sequence Alignments	137
Predicting Locally stable structures of large sequences	132
Local MFE structure Prediction and Z-scores	133
Partition functions for locally stable secondary structures	134
Local MFE consensus structures for Sequence Alignments	137
Classified Dynamic Programming	144
Distance based partitioning of the Secondary Structure Space	145
Calculating MFE representatives of a Distance Based Partitioning	146
Calculate Partition Functions of a Distance Based Partitioning	150

Stochastic Backtracking of Structures from Distance Based Partitioning	152
Compute the Density of States	154
Constraining the Secondary Structure Recursions	155
Hard Constraints	160
Soft Constraints	163
Generalized Soft Constraints	170
Inverse Secondary Structure Prediction	40
Free Energy Evaluation for given Sequence / Structure Pairs	42
Processing and Evaluating Decomposed Loops	55
Energy Parameter Sets and Boltzmann Factors	65
Reading/Writing Energy Parameter Sets from/to File	138
Converting Energy Parameter Files	140
Data Structures and Preprocessor Macros	72
Basic Data Structures for Structure Prediction and Evaluation	173
Utilities	74
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures	186
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments	198
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More	203

Chapter 9

Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

bondT	Base pair	209
bondTEn	Base pair with associated energy	209
cofoldF	209
ConcEnt	210
constrain	Constraints for cofolding	210
COORDINATE	This is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE	210
cpair	This datastructure is used as input parameter in functions of PS_dot.c	210
duplexT	211
dupVar	211
folden	211
interact	211
intermediate_t	212
INTERVAL	Sequence interval stack element used in subopt.c	212
LIST	213
LST_BUCKET	213
move_t	213
PAIR	Base pair data structure used in subopt.c	213
pair_info	A base pair info structure	214
pairpro	215
path_t	215
plist	This datastructure is used as input parameter in functions of PS_dot.h and others	215
Postorder_list	215
pu_contrib	Contributions to p_u	216
pu_out	Collection of all free_energy of beeing unpaired values for output	216
sect	Stack of partial structures for backtracking	217

snoopT	217
SOLUTION	
Solution element from subopt.c	217
struct_en	217
svm_model	217
swString	218
Tree	218
TwoDfold_vars	
Variables compound for 2Dfold MFE folding	218
TwoDpfold_vars	
Variables compound for 2Dfold partition function folding	219
vrna_exp_param_t	
The datastructure that contains temperature scaled Boltzmann weights of the energy parameters	221
vrna_fold_compound	
The most basic data structure required by many functions throughout the RNAlib	222
vrna_hc_t	
The hard constraints data structure	227
vrna_helix	228
vrna_md_t	
The data structure that contains the complete model details used throughout the calculations	228
vrna_mx_mfe_t	
Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound	230
vrna_mx_pf_t	
Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound	232
vrna_param_t	
The datastructure that contains temperature scaled energy parameters	233
vrna_sc_t	
The soft constraints data structure	234
vrna_sol_TwoD_pf_t	
Solution element returned from vrna_TwoD_pf_fold()	236
vrna_sol_TwoD_t	
Solution element returned from vrna_TwoD_fold()	237

Chapter 10

File Index

10.1 File List

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

/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_epars.h	
Free energy parameters for parameter file conversion	239
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_intloops.h	
Free energy parameters for interior loop contributions needed by the parameter file conversion functions	239
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dfold.h	240
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dpfold.h	241
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/ali_plex.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/alifold.h	
Compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments	244
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/aln_util.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/centroid.h	
Centroid structure computation	247
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/cofold.h	
MFE version of cofolding routines	248
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/constraints.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/convert_epars.h	
Functions and definitions for energy parameter file format conversion	250
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h	251
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/dist_vars.h	
Global variables for Distance-Package	253
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/duplex.h	
Duplex folding function declarations..	255
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/edit_cost.h	
Global variables for Edit Costs included by treedist.c and stringdist.c	255
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/energy_const.h	256
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/energy_par.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/eval.h	
Functions and variables related to energy evaluation of sequence/structure pairs	257
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/exterior_loops.h	
Energy evaluation of exterior loops for MFE and partition function calculations	260
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/file_formats.h	
Functions dealing with file formats for RNA sequences, structures, and alignments	261
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/findpath.h	
Compute direct refolding paths between two secondary structures	262
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/fold.h	
MFE calculations for single RNA sequences	263

/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/fold_vars.h	
Here all all declarations of the global variables used throughout RNALib	265
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/gquad.h	
Various functions related to G-quadruplex computations	268
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/hairpin_loops.h	
Energy evaluation of hairpin loops for MFE and partition function calculations	269
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/interior_loops.h	
Energy evaluation of interior loops for MFE and partition function calculations	270
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/intl11.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/intl11dH.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/intl21.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/intl21dH.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/intl22.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/intl22dH.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/inverse.h	
Inverse folding routines	272
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/Lfold.h	
Predicting local MFE structures of large sequences	272
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/list.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/loop_energies.h	
Energy evaluation for MFE and partition function calculations	272
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/LPfold.h	
Function declarations of partition function variants of the Lfold algorithm	273
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/MEA.h	
Computes a MEA (maximum expected accuracy) structure	275
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/mm.h	
Several Maximum Matching implementations	276
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/model.h	
The model details data structure and its corresponding modifiers	276
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/move_set.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/multibranch_loops.h	
Energy evaluation of multibranch loops for MFE and partition function calculations	279
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/naview.h	280
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/pair_mat.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/params.h	280
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func.h	
Partition function of single RNA sequences	281
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_co.h	
Partition function for two RNA sequences	284
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_up.h	
Partition Function Cofolding as stepwise process	289
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/perturbation_fold.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/PKplex.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/plex.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/plot_layouts.h	
Secondary structure plot layout algorithms	291
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/ProfileAln.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/profiledist.h	293
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/PS_dot.h	
Various functions for plotting RNA secondary structures, dot-plots and other visualizations	294
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/read_epars.h	296
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/ribo.h	
Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments	296
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/RNAstruct.h	
Parsing and Coarse Graining of Structures	296
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/snofold.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/snoop.h	??

/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/stringdist.h	
Functions for String Alignment	298
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/structure_utils.h	
Various utility- and helper-functions for secondary structure parsing, converting, etc	299
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/subopt.h	
RNAsubopt and density of states declarations	302
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/svm_utils.h	??
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/treedist.h	
Functions for Tree Edit Distances	303
/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/utils.h	
General utility- and helper-functions used throughout the <i>ViennaRNA Package</i>	304

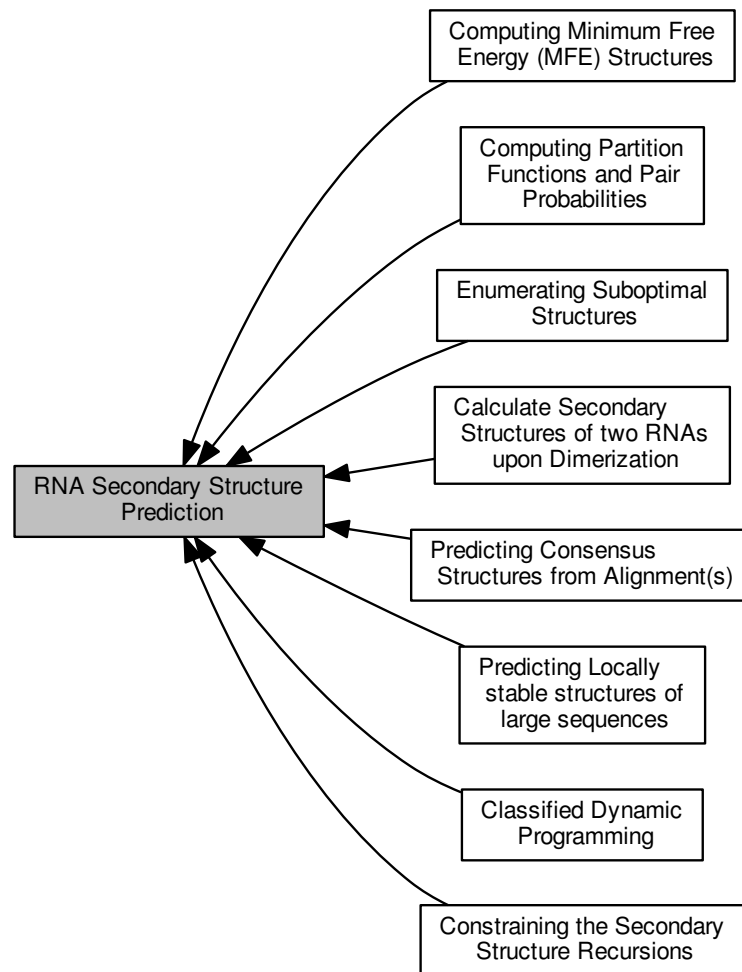
Chapter 11

Module Documentation

11.1 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](#)

- [Constraining the Secondary Structure Recursions](#)

This module covers all functions and variables related to the problem of incorporating secondary structure constraints into the folding recursions.

Files

- file [mm.h](#)

Several Maximum Matching implementations.

11.1.1 Detailed Description

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

11.2 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](#)

11.2.1 Detailed Description

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

11.2.2 Function Documentation

11.2.2.1 float [inverse_fold](#) (char * *start*, const char * *target*)

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

<i>start</i>	The start sequence
<i>target</i>	The target secondary structure in dot-bracket notation

Returns

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

11.2.2.2 float [inverse_pf_fold](#) (char * *start*, const char * *target*)

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

<i>start</i>	The start sequence
<i>target</i>	The target secondary structure in dot-bracket notation

Returns

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

11.2.3 Variable Documentation

11.2.3.1 float final_cost

when to stop [inverse_pf_fold\(\)](#)

11.2.3.2 int give_up

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

11.2.3.3 int inv_verbose

print out substructure on which [inverse_fold\(\)](#) fails

11.3 Free Energy Evaluation for given Sequence / Structure Pairs

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

Functions

- float [vrna_eval_structure](#) ([vrna_fold_compound](#) *vc, const char *structure)
Calculate the free energy of an already folded RNA.
- float [vrna_eval_covar_structure](#) ([vrna_fold_compound](#) *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](#) *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_simple_verbose](#) (const char *string, const char *structure, FILE *file)
Calculate the free energy of an already folded RNA and print contributions per loop.
- int [vrna_eval_structure_pt](#) ([vrna_fold_compound](#) *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](#) *vc, const short *pt, 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_loop_pt](#) ([vrna_fold_compound](#) *vc, int i, const short *pt)
Calculate energy of a loop.
- float [vrna_eval_move](#) ([vrna_fold_compound](#) *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](#) *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)
- PRIVATE int `vrna_eval_ext_hp_loop` (`vrna_fold_compound` *vc, int i, int j)
Evaluate free energy of an exterior hairpin loop.
- PRIVATE int `vrna_eval_hp_loop` (`vrna_fold_compound` *vc, int i, int j)
Evaluate free energy of a hairpin loop.

Variables

- int `cut_point`
set to first pos of second seq for cofolding
- int `eos_debug`
verbose info from energy_of_struct

11.3.1 Detailed Description

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

11.3.2 Function Documentation

11.3.2.1 float `vrna_eval_structure` (`vrna_fold_compound` *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` 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_get_fold_compound(sequence, NULL,
    VRNA_OPTION_MFE | VRNA_OPTION_EVAL_ONLY);
```

Note

Accepts `vrna_fold_compound` 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_get_fold_compound()`, `vrna_get_fold_compound_aliases()`, `vrna_eval_covar_structure()`

Parameters

<code>vc</code>	A <code>vrna_fold_compound</code> containing the energy parameters and model details
<code>structure</code>	Secondary structure in dot-bracket notation

Returns

The free energy of the input structure given the input sequence in kcal/mol

11.3.2.2 float vrna_eval_covar_structure (vrna_fold_compound * 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` 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_get_fold_compound_aln(alignment, NULL,
    VRNA_OPTION_MFE | VRNA_OPTION_EVAL_ONLY);
```

Note

Accepts `vrna_fold_compound` of type `VRNA_VC_TYPE_ALIGNMENT` only!

See also

`vrna_get_fold_compound_aln()`, `vrna_eval_structure()`

Parameters

<code>vc</code>	A <code>vrna_fold_compound</code> containing the energy parameters and model details
<code>structure</code>	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

11.3.2.3 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

<code>string</code>	RNA sequence in uppercase letters
<code>structure</code>	Secondary structure in dot-bracket notation

Returns

The free energy of the input structure given the input sequence in kcal/mol

11.3.2.4 float vrna_eval_structure_verbose (vrna_fold_compound * 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 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. 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_get_fold_compound(sequence, NULL,
    VRNA_OPTION_MFE | VRNA_OPTION_EVAL_ONLY);
```

See also

[vrna_eval_structure_pt\(\)](#), [vrna_eval_structure_verbose\(\)](#), [vrna_eval_structure_pt_verbose\(\)](#),

Parameters

<i>vc</i>	A vrna_fold_compound containing the energy parameters and model details
<i>structure</i>	Secondary structure in dot-bracket notation
<i>file</i>	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

11.3.2.5 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 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. 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. 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_verbose\(\)](#), [vrna_eval_structure_pt\(\)](#), [vrna_eval_structure_verbose\(\)](#), [vrna_eval_structure_pt_verbose\(\)](#),

Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>file</i>	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

11.3.2.6 int vrna_eval_structure_pt (vrna_fold_compound * *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_pt_get\(\)](#). 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_get_fold_compound(sequence, NULL,
    VRNA_OPTION_MFE | VRNA_OPTION_EVAL_ONLY);
```

See also

[vrna_pt_get\(\)](#), [vrna_eval_structure\(\)](#), [vrna_eval_structure_pt_verbose\(\)](#)

Parameters

<i>vc</i>	A vrna_fold_compound containing the energy parameters and model details
<i>pt</i>	Secondary structure as pair_table

Returns

The free energy of the input structure given the input sequence in 10cal/mol

11.3.2.7 `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. 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_pt_get\(\)](#), [vrna_eval_structure_simple\(\)](#), [vrna_eval_structure_pt\(\)](#)

Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>pt</i>	Secondary structure as pair_table

Returns

The free energy of the input structure given the input sequence in 10cal/mol

11.3.2.8 `int vrna_eval_structure_pt_verbose (vrna_fold_compound * vc, const short * pt, 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_pt_get\(\)](#). 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_get_fold_compound(sequence, NULL,
    VRNA_OPTION_MFE | 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.

See also

[vrna_pt_get\(\)](#), [vrna_eval_structure_pt\(\)](#), [vrna_eval_structure_verbose\(\)](#)

Parameters

<i>vc</i>	A vrna_fold_compound containing the energy parameters and model details
<i>pt</i>	Secondary structure as pair_table

<i>file</i>	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

11.3.2.9 `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 allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from [vrna_pt_get\(\)](#). 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_get_fold_compound(sequence, NULL,
    VRNA_OPTION_MFE | 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. 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_pt_get\(\)](#), [vrna_eval_structure_pt_verbose\(\)](#), [vrna_eval_structure_simple\(\)](#)

Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>pt</i>	Secondary structure as pair_table
<i>file</i>	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

11.3.2.10 `int vrna_eval_loop_pt (vrna_fold_compound * vc, int i, const short * pt)`

Calculate energy of a loop.

Parameters

<i>vc</i>	A vrna_fold_compound containing the energy parameters and model details
<i>i</i>	position of covering base pair
<i>pt</i>	the pair table of the secondary structure

Returns

free energy of the loop in 10cal/mol

11.3.2.11 `float vrna_eval_move (vrna_fold_compound * 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

<i>vc</i>	A vrna_fold_compound containing the energy parameters and model details
<i>structure</i>	secondary structure in dot-bracket notation
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

Returns

energy change of the move in kcal/mol

11.3.2.12 `int vrna_eval_move_pt (vrna_fold_compound * 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

<i>vc</i>	A vrna_fold_compound containing the energy parameters and model details
<i>pt</i>	the pair table of the secondary structure
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

Returns

energy change of the move in 10cal/mol

11.3.2.13 `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

<i>string</i>	RNA sequence
---------------	--------------

<i>structure</i>	secondary structure in dot-bracket notation
<i>verbosity_level</i>	a flag to turn verbose output on/off

Returns

the free energy of the input structure given the input sequence in kcal/mol

11.3.2.14 `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

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in kcal/mol

11.3.2.15 `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

<i>string</i>	RNA sequence
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in kcal/mol

11.3.2.16 `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

<i>string</i>	RNA sequence
<i>structure</i>	Secondary structure in dot-bracket notation
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in kcal/mol

11.3.2.17 `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

<i>string</i>	RNA sequence
<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>verbosity_level</i>	a flag to turn verbose output on/off

Returns

the free energy of the input structure given the input sequence in 10kcal/mol

11.3.2.18 `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

<i>string</i>	RNA sequence in uppercase letters
<i>ptable</i>	The pair table of the secondary structure
<i>s</i>	Encoded RNA sequence
<i>s1</i>	Encoded RNA sequence
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to turn verbose output on/off

Returns

The free energy of the input structure given the input sequence in 10kcal/mol

11.3.2.19 `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

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

Returns

energy change of the move in kcal/mol

11.3.2.20 `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

<i>pt</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

Returns

energy change of the move in 10cal/mol

11.3.2.21 `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

<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>i</i>	position of covering base pair

Returns

free energy of the loop in 10cal/mol

11.3.2.22 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

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation

Returns

the free energy of the input structure given the input sequence in kcal/mol

11.3.2.23 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

<i>string</i>	RNA sequence
<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence

Returns

the free energy of the input structure given the input sequence in 10kcal/mol

11.3.2.24 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

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation

Returns

the free energy of the input structure given the input sequence in kcal/mol

11.3.2.25 PRIVATE int vrna_eval_hp_loop (vrna_fold_compound * *vc*, int *i*, int *j*)

Evaluate free energy of a hairpin loop.

Note

This function is polymorphic! The provided [vrna_fold_compound](#) may be of type [VRNA_VC_TYPE_SINGLE](#) or [VRNA_VC_TYPE_ALIGNMENT](#)

Parameters

<i>vc</i>	The vrna_fold_compound for the particular energy evaluation
<i>i</i>	5'-position of the base pair
<i>j</i>	3'-position of the base pair

Returns

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

11.4 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

- PRIVATE int [E_ExtLoop](#) (int type, int si1, int sj1, [vrna_param_t](#) *P)
- PRIVATE double [exp_E_ExtLoop](#) (int type, int si1, int sj1, [vrna_exp_param_t](#) *P)
- PRIVATE int [E_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna_param_t](#) *P)
- PRIVATE double [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 double [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.
- PRIVATE int [vrna_E_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
Evaluate the free energy of a hairpin loop and consider possible hard constraints.
- PRIVATE int [vrna_E_ext_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.
- PRIVATE double [vrna_exp_E_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
High-Level function for hairpin loop energy evaluation (partition function variant)
- PRIVATE int [vrna_BT_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j, int en, [bondT](#) *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 double [exp_E_IntLoop](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, [vrna_exp_param_t](#) *P)
- PRIVATE int [E_stack](#) (int i, int j, [vrna_fold_compound](#) *vc)
Evaluate energy of a base pair stack closed by (i, j)
- PRIVATE int [vrna_BT_stack](#) ([vrna_fold_compound](#) *vc, int *i, int *j, int *en, [bondT](#) *bp_stack, int *stack_↔ count)
Backtrack a stacked pair closed by (i, j).

- PRIVATE int [vrna_BT_int_loop](#) ([vrna_fold_compound](#) *vc, int *i, int *j, int en, [bondT](#) *bp_stack, int *stack↔_count)
Backtrack an interior loop closed by (i, j).
- PRIVATE int [E_mb_loop_stack](#) (int i, int j, [vrna_fold_compound](#) *vc)
Evaluate energy of a multi branch helices stacking onto closing pair (i,j)
- PRIVATE int [vrna_BT_mb_loop](#) ([vrna_fold_compound](#) *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).

11.4.1 Detailed Description

11.4.2 Function Documentation

11.4.2.1 PRIVATE int E_ExtLoop (int type, int si1, int sj1, [vrna_param_t](#) * P)

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

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

Returns

The energy contribution of the introduced exterior-loop stem

11.4.2.2 PRIVATE double exp_E_ExtLoop (int type, int si1, int sj1, [vrna_exp_param_t](#) * P)

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

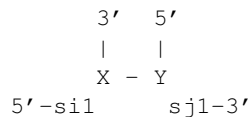
11.4.2.3 PRIVATE int E_Stem (int type, int si1, int sj1, int extLoop, [vrna_param_t](#) * P)

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 behavior 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 incorporating 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:



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 negative value, only 5' or 3' dangling end contributions are taken into account. To prohibit any of these mismatch contributions to be incorporated, 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

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

Returns

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

11.4.2.4 PRIVATE double exp_E_Stem (int type, int si1, int sj1, int extLoop, vrna_exp_param_t * P)

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

11.4.2.5 `int* get_gquad_matrix (short * S, vrna_param_t * P)`

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

<i>S</i>	The encoded sequence
<i>P</i>	A pointer to the data structure containing the precomputed energy contributions

Returns

A pointer to the G-quadruplex contribution matrix

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

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; ... ;`

11.4.2.7 `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)`

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

Parameters

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

Returns

1 on success, 0 if no gquad found

11.4.2.8 `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)`

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

Parameters

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

Returns

1 on success, 0 if no gquad found

11.4.2.9 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.

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

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

Returns

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

11.4.2.10 PRIVATE double 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.

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

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

Returns

The Boltzmann weight of the Hairpin-loop

11.4.2.11 PRIVATE int vrna_E_hp_loop (vrna_fold_compound * *vc*, int *i*, int *j*)

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

Note

This function is polymorphic! The provided [vrna_fold_compound](#) may be of type [VRNA_VC_TYPE_SINGLE](#) or [VRNA_VC_TYPE_ALIGNMENT](#)

11.4.2.12 PRIVATE double vrna_exp_E_hp_loop (vrna_fold_compound * vc, int i, int j)

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

See also

E_hp_loop() for it's free energy counterpart

11.4.2.13 PRIVATE int vrna_BT_hp_loop (vrna_fold_compound * vc, int i, int j, int en, bondT * bp_stack, int * stack_count)

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

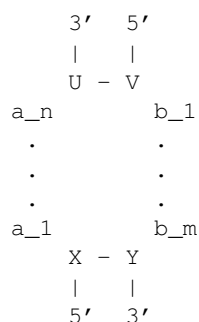
Note

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

11.4.2.14 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)

Compute the Energy of an interior-loop

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



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 around' when evaluating the free energy of the interior-loop

See also

[scale_parameters\(\)](#)
[vrna_param_t](#)

Note

This function is threadsafe

Parameters

<i>n1</i>	The size of the 'left'-loop (number of unpaired nucleotides)
<i>n2</i>	The size of the 'right'-loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the interior loop
<i>type_2</i>	The pair type of the enclosed base pair
<i>si1</i>	The 5'-mismatching nucleotide of the closing pair
<i>sj1</i>	The 3'-mismatching nucleotide of the closing pair
<i>sp1</i>	The 3'-mismatching nucleotide of the enclosed pair
<i>sq1</i>	The 5'-mismatching nucleotide of the enclosed pair
<i>P</i>	The datastructure containing scaled energy parameters

Returns

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

11.4.2.15 **PRIVATE** double exp_E_IntLoop (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*,
vrna_exp_param_t * *P*)

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

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

Returns

The Boltzmann weight of the Interior-loop

11.4.2.16 **PRIVATE** int E_mb_loop_stack (int *i*, int *j*, vrna_fold_compound * *vc*)

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)

```
11.4.2.17 PRIVATE int vrna_BT_mb_loop ( vrna_fold_compound * 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) .

Parameters

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

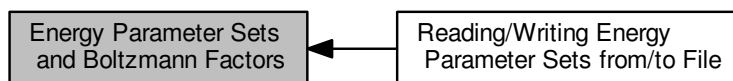
Returns

1, if backtracking succeeded, 0 otherwise.

11.5 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_t](#)
The datastructure that contains temperature scaled energy parameters.
- struct [vrna_exp_param_t](#)
The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.

Functions

- void [vrna_exp_params_update](#) ([vrna_fold_compound](#) *vc, [vrna_exp_param_t](#) *params)
Update the energy parameters for subsequent partition function computations.
- void [vrna_exp_params_rescale](#) ([vrna_fold_compound](#) *vc, double *mfe)
Rescale Boltzmann factors for partition function computations.
- [vrna_param_t](#) * [vrna_params_get](#) ([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_get](#) ([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_alifold_get](#) (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)
- [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.

11.5.1 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.

11.5.2 Function Documentation

11.5.2.1 void vrna_exp_params_update (vrna_fold_compound * vc, vrna_exp_param_t * params)

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`. 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_rescale()`, `vrna_exp_param_t`, `vrna_md_t`, `vrna_exp_params_get()`

Parameters

<code>vc</code>	The fold compound data structure
<code>params</code>	A pointer to the new energy parameters

11.5.2.2 void vrna_exp_params_rescale (vrna_fold_compound * vc, double * mfe)

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 $184.3 \times \text{length}$ 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 $s = \exp((s_{\text{fact}} * \text{MFE}) / kT / \text{length})$, 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_update\(\)](#), [vrna_md_t](#), [vrna_exp_param_t](#), [vrna_fold_compound](#)

Parameters

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

11.5.2.3 `vrna_param_t* vrna_params_get (vrna_md_t * md)`

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_get\(\)](#)

Parameters

<code>md</code>	A pointer to the model details to store inside the structure (Maybe <code>NULL</code>)
-----------------	---

Returns

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

11.5.2.4 `vrna_param_t* vrna_params_copy (vrna_param_t * par)`

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_get\(\)](#), [vrna_param_t](#)

Parameters

<i>par</i>	The free energy parameters that are to be copied (Maybe NULL)
------------	---

Returns

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

11.5.2.5 `vrna_exp_param_t* vrna_exp_params_get (vrna_md_t * md)`

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_get\(\)](#), 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_get\(\)](#), [vrna_rescale_pf_params\(\)](#)

Parameters

<i>md</i>	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

11.5.2.6 `vrna_exp_param_t* vrna_exp_params_ali_get (unsigned int n_seq, vrna_md_t * md)`

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_get\(\)](#), [vrna_params_get\(\)](#)

Parameters

<i>n_seq</i>	The number of sequences in the alignment
<i>md</i>	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

11.5.2.7 `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)

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

See also

[vrna_exp_params_get\(\)](#), [vrna_exp_param_t](#)

Parameters

<i>par</i>	The free energy parameters that are to be copied (Maybe NULL)
------------	---

Returns

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

11.5.2.8 `vrna_exp_param_t* get_scaled_pf_parameters (void)`

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_get\(\)](#) instead!

Returns

The datastructure containing Boltzmann weights for use in partition function calculations

11.5.2.9 `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.

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_get\(\)](#) instead!

See also

[get_scaled_pf_parameters\(\)](#), [get_boltzmann_factor_copy\(\)](#)

Parameters

<i>temperature</i>	The temperature in degrees Celcius used for (re-)scaling the energy contributions
<i>betaScale</i>	A scaling value that is used as a multiplication factor for the absolute temperature of the system

<i>md</i>	The model details to be used
<i>pf_scale</i>	The scaling factor for the Boltzmann factors

Returns

A set of precomputed Boltzmann factors

11.5.2.10 `vrna_exp_param_t* get_boltzmann_factor_copy (vrna_exp_param_t * parameters)`

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

<i>parameters</i>	The input data structure that shall be copied
-------------------	---

Returns

A copy of the provided Boltzmann factor dataset

11.5.2.11 `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)

Deprecated Use `vrna_exp_params_ali_get()` instead!

11.5.2.12 `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.

Deprecated Use `vrna_exp_params_ali_get()` instead!

11.5.2.13 `vrna_param_t* scale_parameters (void)`

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_get()` instead!

Returns

A set of precomputed energy contributions

11.5.2.14 `vrna_param_t*` `get_scaled_parameters (double temperature, vrna_md_t md)`

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_get()` instead!

See also

`vrna_md_t`, `set_model_details()`

Parameters

<i>temperature</i>	The temperature in degrees Celcius
<i>md</i>	The model details

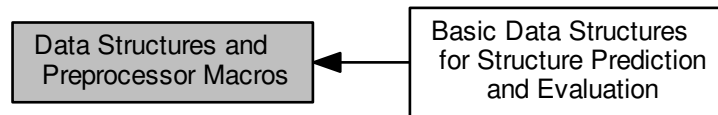
Returns

precomputed energy contributions and model settings

11.6 Data Structures and Preprocessor Macros

All datastructures and typedefs shared among the Vienna RNA Package can be found here.

Collaboration diagram for Data Structures and Preprocessor Macros:



Modules

- [Basic Data Structures for Structure Prediction and Evaluation](#)

This module provides interfaces that deal with the most basic data structures used in structure predicting and energy evaluating function of the RNAlib.

Data Structures

- struct [plist](#)
this datastructure is used as input parameter in functions of [PS_dot.h](#) and others
- struct [cpair](#)
this datastructure is used as input parameter in functions of [PS_dot.c](#)
- struct [sect](#)
Stack of partial structures for backtracking.
- struct [bondT](#)
Base pair.
- struct [bondTE](#)
Base pair with associated energy.
- struct [PAIR](#)
Base pair data structure used in [subopt.c](#).
- struct [INTERVAL](#)
Sequence interval stack element used in [subopt.c](#).
- struct [SOLUTION](#)
Solution element from [subopt.c](#).
- struct [cofoldF](#)
- struct [ConcEnt](#)
- struct [pairpro](#)
- struct [pair_info](#)
A base pair info structure.
- struct [move_t](#)
- struct [intermediate_t](#)
- struct [path_t](#)
- struct [pu_contrib](#)
contributions to p_u
- struct [interact](#)

- struct `pu_out`
Collection of all free_energy of beeing unpaired values for output.
- struct `constrain`
constraints for cofolding
- struct `duplexT`
- struct `folden`
- struct `snoopT`
- struct `dupVar`

Macros

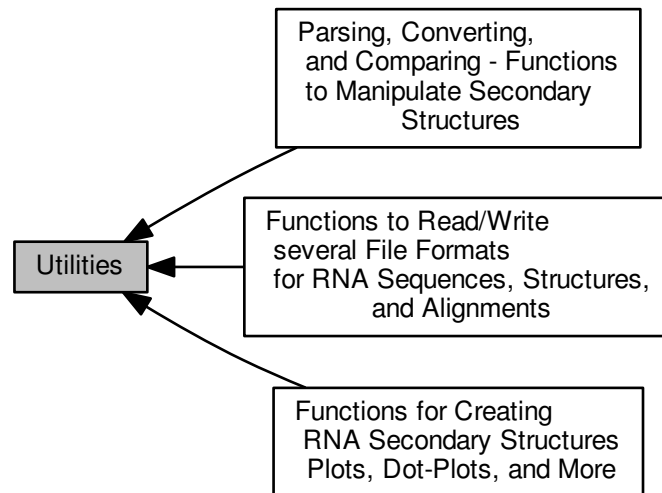
- `#define MAXDOS 1000`
Maximum density of states discretization for subopt.

11.6.1 Detailed Description

All datastructures and typedefs shared among the Vienna RNA Package can be found [here](#).

11.7 Utilities

Collaboration diagram for Utilities:



Modules

- [Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures](#)
- [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 [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 occurred, 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 truncate the line by eliminating white spaces at end of line".

 - `#define VRNA_INPUT_NO_REST 512U`
- Input switch for `vrna_read_fasta_record()`: "do fill rest array".

 - `#define VRNA_INPUT_NO_SPAN 1024U`
- Input switch for `vrna_read_fasta_record()`: "never allow data to span more than one line".

 - `#define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U`
- Input switch for `vrna_read_fasta_record()`: "do not skip empty lines".

 - `#define VRNA_INPUT_BLANK_LINE 4096U`
- Output flag for `vrna_read_fasta_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_read_fasta_record()`: "read a comment".

 - `#define VRNA_OPTION_MULTILINE 32U`
- Tell a function that an input is assumed to span several lines.

 - `#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.

 - `#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

- `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 * [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.
- 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.
- 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.
- int * [vrna_get_iindx](#) (unsigned int length)
Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.
- int * [vrna_get_indx](#) (unsigned int length)
Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.
- 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.
- char * [vrna_get_ptypes](#) (const short *S, [vrna_md_t](#) *md)
Get an array of the numerical encoding for each possible base pair (i,j)

Variables

- unsigned short [xsubi](#) [3]
Current 48 bit random number.

11.7.1 Detailed Description

11.7.2 Macro Definition Documentation

11.7.2.1 #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.

the function will return this flag if a fasta header was read

11.7.2.2 `#define VRNA_INPUT_CONSTRAINT 32U`

Input flag for `get_input_line()`:

Tell `get_input_line()` that we assume to read a structure constraint.

11.7.2.3 `#define VRNA_OPTION_MULTILINE 32U`

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_read_fasta_record()`, `vrna_extract_record_rest_constraint()`

11.7.2.4 `#define FILENAME_MAX_LENGTH 80`

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 file-names of output files is declared.

11.7.2.5 `#define FILENAME_ID_LENGTH 42`

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

11.7.3 Function Documentation

11.7.3.1 `void* vrna_alloc (unsigned size)`

Allocate space safely.

Parameters

<i>size</i>	The size of the memory to be allocated in bytes
-------------	---

Returns

A pointer to the allocated memory

11.7.3.2 `void* vrna_realloc (void * p, unsigned size)`

Reallocate space safely.

Parameters

<i>p</i>	A pointer to the memory region to be reallocated
<i>size</i>	The size of the memory to be allocated in bytes

Returns

A pointer to the newly allocated memory

11.7.3.3 void vrna_message_error (const char *message*[])

Die with an error message.

See also

[vrna_message_warning\(\)](#)

Parameters

<i>message</i>	The error message to be printed before exiting with 'FAILURE'
----------------	---

11.7.3.4 void vrna_message_warning (const char *message*[])

Print a warning message.

Print a warning message to *stderr*

Parameters

<i>message</i>	The warning message
----------------	---------------------

11.7.3.5 double vrna_urn (void)

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]

11.7.3.6 int vrna_int_urn (int *from*, int *to*)

Generates a pseudo random integer in a specified range.

See also

[vrna_urn\(\)](#), [vrna_init_rand\(\)](#)

Parameters

<i>from</i>	The first number in range
<i>to</i>	The last number in range

Returns

A pseudo random number in range [from, to]

11.7.3.7 char* vrna_time_stamp (void)

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

11.7.3.8 char* vrna_random_string (int l, const char symbols[])

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

Parameters

<i>l</i>	The length of the sequence
<i>symbols</i>	The symbol set

Returns

A random string of length 'l' containing characters from the symbolset

11.7.3.9 int vrna_hamming_distance (const char * s1, const char * s2)

Calculate hamming distance between two sequences.

Parameters

<i>s1</i>	The first sequence
<i>s2</i>	The second sequence

Returns

The hamming distance between s1 and s2

11.7.3.10 int vrna_hamming_distance_bound (const char * s1, const char * s2, int n)

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

<i>s1</i>	The first sequence
<i>s2</i>	The second sequence

Returns

The hamming distance between s1 and s2

11.7.3.11 char* get_line (FILE * fp)

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

<i>fp</i>	A file pointer to the stream where the function should read from
-----------	--

Returns

A pointer to the resulting string

11.7.3.12 unsigned int get_input_line (char ** *string*, unsigned int *options*)

Retrieve a line from 'stdin' safely 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_WS_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

<i>string</i>	A pointer to the character array that contains the line read
<i>options</i>	A collection of options for switching the functions behavior

Returns

A flag with information about what has been read

11.7.3.13 void vrna_message_input_seq_simple (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

11.7.3.14 void vrna_message_input_seq (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

Parameters

<i>s</i>	A user defined string that will be printed to stdout
----------	--

11.7.3.15 void vrna_seq_toRNA (char * *sequence*)

Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.

This function substitutes *T* and *t* with *U* and *u*, respectively

Parameters

<i>sequence</i>	The sequence to be converted
-----------------	------------------------------

11.7.3.16 void vrna_seq_toupper (char * *sequence*)

Convert an input sequence to uppercase.

Parameters

<i>sequence</i>	The sequence to be converted
-----------------	------------------------------

11.7.3.17 int* vrna_get_iindx (unsigned int *length*)

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

$$(i, j) \sim \text{iindx}[i] - j$$

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_get_indx\(\)](#)

Parameters

<i>length</i>	The length of the RNA sequence
---------------	--------------------------------

Returns

The mapper array

11.7.3.18 int* vrna_get_indx (unsigned int *length*)

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

$$(i, j) \sim \text{indx}[j] + i$$

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_get_iindx\(\)](#)

Parameters

<i>length</i>	The length of the RNA sequence
---------------	--------------------------------

Returns

The mapper array

11.7.3.19 `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

<i>c</i>	The nucleotide character to encode
<i>md</i>	The model details that determine the kind of encoding

Returns

The encoded nucleotide

11.7.3.20 `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

<i>enc</i>	The encoded nucleotide
<i>md</i>	The model details that determine the kind of decoding

Returns

The decoded nucleotide character

11.7.3.21 `char* vrna_get_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 via jindx, in contrast to previously different indexing between mfe and pf variants!

See also

[vrna_get_indx\(\)](#), [vrna_fold_compound](#)

11.7.4 Variable Documentation

11.7.4.1 unsigned short xsubi[3]

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\(\)](#).

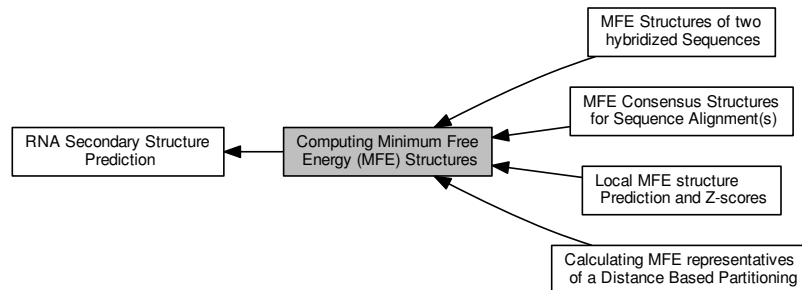
See also

[vrna_urn\(\)](#)

11.8 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 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_fold](#) (vrna_fold_compound *vc, char *structure)
Compute minimum free energy and an appropriate secondary structure of an 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)

11.8.1 Detailed Description

This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

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 in [17]. 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 [7] 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.

The library provides a fast dynamic programming minimum free energy folding algorithm as described by Zuker & Stiegler (1981).

11.8.2 Function Documentation

11.8.2.1 float vrna_fold (vrna_fold_compound * vc, char * structure)

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

The second parameter, *structure*, should point to an allocated block of memory with a size of at least `strlen(sequence) + 1` to store the backtracked MFE structure. If `NULL` is passed, no backtracking will be performed.

See also

[vrna_fold_compound\(\)](#), [vrna_get_fold_compound\(\)](#)

Parameters

<i>vc</i>	fold compound
<i>structure</i>	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

11.8.2.2 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.

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 `strlen(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_fold\(\)](#) instead

See also

[vrna_fold\(\)](#), [fold\(\)](#), [circfold\(\)](#), [vrna_md_t](#), [set_energy_model\(\)](#), [get_scaled_parameters\(\)](#)

Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details. (NULL may be passed, see OpenMP notes above)
<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
<i>is_circular</i>	Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

Returns

the minimum free energy (MFE) in kcal/mol

11.8.2.3 float fold (const char * *sequence*, char * *structure*)

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\(\)](#) instead

See also

[fold_par\(\)](#), [circfold\(\)](#)

Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	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

11.8.2.4 float circfold (const char * *sequence*, char * *structure*)

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_fold\(\)](#) instead!

See also

[fold_par\(\)](#), [circfold\(\)](#)

Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	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

11.8.2.5 void free_arrays (void)

Free arrays for mfe folding.

Deprecated See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

11.8.2.6 void update_fold_params (void)

Recalculate energy parameters.

Deprecated use [vrna_params_update\(\)](#) instead

11.8.2.7 void update_fold_params_par (vrna_param_t * *parameters*)

Recalculate energy parameters.

Deprecated use [vrna_params_update\(\)](#) instead

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

Deprecated See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

11.8.2.9 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*)

Deprecated See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

11.8.2.10 `void export_circfold_arrays (int * Fc_p, int * FcH_p, int * Fcl_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)`

Deprecated See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

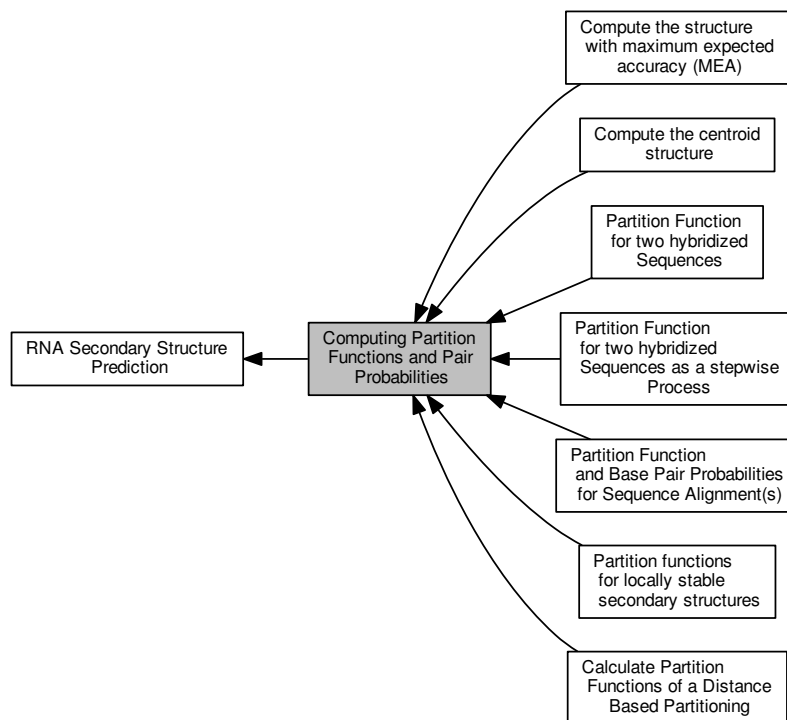
11.8.2.11 `void export_circfold_arrays_par (int * Fc_p, int * FcH_p, int * Fcl_p, int * FcM_p, int ** fM2_p, int ** f5_p, int ** c_p, int ** fML_p, int ** fM1_p, int ** indx_p, char ** ptype_p, vrna_param_t ** P_p)`

Deprecated See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

11.9 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 [part_func.h](#)
Partition function of single RNA sequences.

Functions

- float [vrna_pf_fold](#) ([vrna_fold_compound](#) *vc, char *structure)
Compute the partition function Q for a given RNA sequence.
- 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](#) *vc)
Get the mean base pair distance in the thermodynamic ensemble.
- [plist](#) * [vrna_stack_prob](#) ([vrna_fold_compound](#) *vc, double cutoff)
Compute stacking probabilities.
- float [pf_fold_par](#) (const char *sequence, char *structure, [vrna_exp_param_t](#) *parameters, int calculate_bp↵, 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 relevant 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.
- [plist](#) * [vrna_pl_get_from_pr](#) ([vrna_fold_compound](#) *vc, double cut_off)
Create a [plist](#) from base pair probability matrix.
- void [assign_plist_from_pr](#) ([plist](#) **pl, FLT_OR_DBL *probs, int length, double cutoff)
Create a [plist](#) from a probability matrix.

11.9.1 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 [11].

11.9.2 Function Documentation

11.9.2.1 float vrna_pf_fold (vrna_fold_compound * vc, char * structure)

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.

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\(\)](#)

See also

[vrna_get_fold_compound\(\)](#), [vrna_db_get_from_pr\(\)](#), [export_bppm\(\)](#), [get_boltzmann_factors\(\)](#), [free_pf_arrays\(\)](#)

Parameters

<i>in, out</i>	<i>vc</i>	The fold compound data structure
<i>in, out</i>	<i>structure</i>	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

11.9.2.2 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.

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs p_{ij} as

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

Parameters

<i>length</i>	The length of the sequence
<i>pr</i>	The matrix containing the base pair probabilities

Returns

The mean pair distance of the structure ensemble

11.9.2.3 double vrna_mean_bp_distance (vrna_fold_compound * vc)

Get the mean base pair distance in the thermodynamic ensemble.

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs p_{ij} as

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

Parameters

<code>vc</code>	The fold compound data structure
-----------------	----------------------------------

Returns

The mean pair distance of the structure ensemble

11.9.2.4 `plist* vrna_stack_prob (vrna_fold_compound * vc, double cutoff)`

Compute stacking probabilities.

For each possible base pair (i, j) , compute the probability of a stack $(i, j), (i + 1, j - 1)$.

Parameters

<code>vc</code>	The fold compound data structure with precomputed base pair probabilities
<code>cutoff</code>	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 probability p

11.9.2.5 `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.

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_fold()` 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_fold()`, `bppm_to_structure()`, `export_bppm()`, `get_boltzmann_factors()`, `free_pf_arrays()`

Parameters

in	<i>sequence</i>	The RNA sequence input
in, out	<i>structure</i>	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	<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
in	<i>calculate_bppm</i>	Switch to Base pair probability calculations on/off (0==off)
in	<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
in	<i>is_circular</i>	Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

Returns

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

11.9.2.6 float pf_fold (const char * *sequence*, char * *structure*)

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

<i>sequence</i>	The RNA sequence input
<i>structure</i>	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

11.9.2.7 float pf_circ_fold (const char * *sequence*, char * *structure*)

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_fold\(\)](#)

Deprecated Use [vrna_pf_fold\(\)](#) instead!

Parameters

<i>in</i>	<i>sequence</i>	The RNA sequence input
<i>in, out</i>	<i>structure</i>	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

11.9.2.8 void free_pf_arrays (void)

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](#) 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\(\)](#)

11.9.2.9 void update_pf_params (int *length*)

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_update\(\)](#) instead

11.9.2.10 void update_pf_params_par (int *length*, vrna_exp_param_t * *parameters*)

Recalculate energy parameters.

Deprecated Use [vrna_exp_params_update\(\)](#) instead

11.9.2.11 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.

```
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_get_iindx\(\)](#)

Returns

A pointer to the base pair probability array

11.9.2.12 `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 relevant 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	<code>S_p</code>	A pointer to the 'S' array (integer representation of nucleotides)
out	<code>S1_p</code>	A pointer to the 'S1' array (2nd integer representation of nucleotides)
out	<code>ptype_p</code>	A pointer to the pair type matrix
out	<code>qb_p</code>	A pointer to the Q^B matrix
out	<code>qm_p</code>	A pointer to the Q^M matrix
out	<code>q1k_p</code>	A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1, k)$)
out	<code>qln_p</code>	A pointer to the 3' slice of the Q matrix ($qln(l) = Q(l, n)$)

Returns

Non Zero if everything went fine, 0 otherwise

11.9.2.13 `double mean_bp_distance (int length)`

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

<code>length</code>	
---------------------	--

Returns

mean base pair distance in thermodynamic ensemble

11.9.2.14 `double mean_bp_distance_pr (int length, FLT_OR_DBL * pr)`

Get the mean base pair distance in the thermodynamic ensemble.

This is a threadsafe implementation of [mean_bp_dist\(\)](#) !

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs p_{ij} as

$$\langle d \rangle = \sum_{i,j} p_{ij} (1 - p_{ij})$$

Deprecated Use [vrna_mean_bp_distance\(\)](#) or [vrna_mean_bp_distance_pr\(\)](#) instead!

Parameters

<i>length</i>	The length of the sequence
<i>pr</i>	The matrix containing the base pair probabilities

Returns

The mean pair distance of the structure ensemble

11.9.2.15 `plist* vrna_pl_get_from_pr (vrna_fold_compound * vc, double cut_off)`

Create a [plist](#) from base pair probability matrix.

The probability matrix provided via the [vrna_fold_compound](#) 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

<i>in</i>	<i>vc</i>	The fold compound
<i>in</i>	<i>cutoff</i>	The cutoff value

Returns

A pointer to the plist that is to be created

11.9.2.16 `void assign_plist_from_pr (plist ** pl, FLT_OR_DBL * probs, int length, double cutoff)`

Create a plist 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

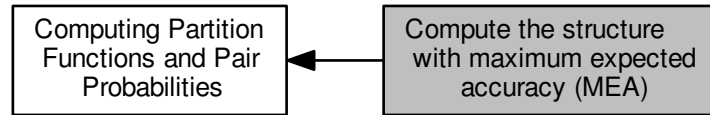
Deprecated Use [vrna_pl_get_from_pr\(\)](#) instead!

Parameters

<i>out</i>	<i>pl</i>	A pointer to the plist that is to be created
<i>in</i>	<i>probs</i>	The probability matrix used for creting the plist
<i>in</i>	<i>length</i>	The length of the RNA sequence
<i>in</i>	<i>cutoff</i>	The cutoff value

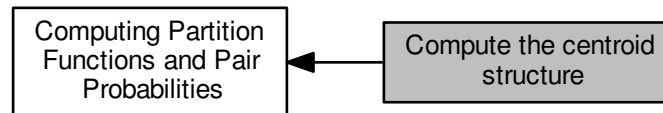
11.10 Compute the structure with maximum expected accuracy (MEA)

Collaboration diagram for Compute the structure with maximum expected accuracy (MEA):



11.11 Compute the centroid structure

Collaboration diagram for Compute the centroid structure:



Functions

- char * [vrna_get_centroid_struct_pl](#) (int length, double *dist, [plist](#) *pl)
Get the centroid structure of the ensemble.
- char * [vrna_get_centroid_struct_pr](#) (int length, double *dist, FLT_OR_DBL *pr)
Get the centroid structure of the ensemble.
- char * [vrna_get_centroid_struct](#) ([vrna_fold_compound](#) *vc, double *dist)
Get the centroid structure of the ensemble.

11.11.1 Detailed Description

11.11.2 Function Documentation

11.11.2.1 char* vrna_get_centroid_struct_pl (int length, double * dist, plist * pl)

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for [centroid\(\)](#) with a 'plist' input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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 addressed by *dist*.

Parameters

in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>pl</i>	A pair list containing base pair probability information about the ensemble

Returns

The centroid structure of the ensemble in dot-bracket notation

11.11.2.2 char* vrna_get_centroid_struct_pr (int length, double * dist, FLT_OR_DBL * pr)

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

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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 addressed by *dist*.

Parameters

in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>pr</i>	A upper triangular matrix containing base pair probabilities (access via vrna_get_iindx())

Returns

The centroid structure of the ensemble in dot-bracket notation

11.11.2.3 `char* vrna_get_centroid_struct (vrna_fold_compound * vc, double * dist)`

Get the centroid structure of the ensemble.

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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 addressed by *dist*.

Parameters

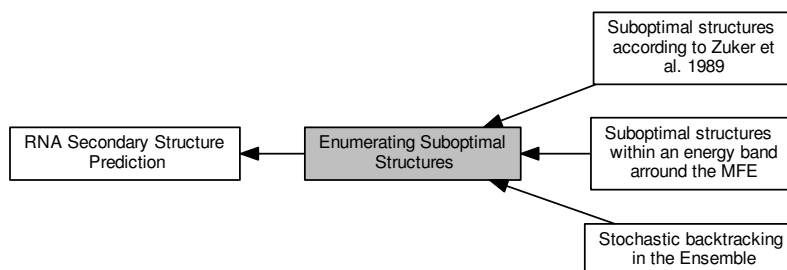
in	<i>vc</i>	The fold compound data structure
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to

Returns

The centroid structure of the ensemble in dot-bracket notation

11.12 Enumerating Suboptimal Structures

Collaboration diagram for Enumerating Suboptimal Structures:



Modules

- [Suboptimal structures according to Zuker et al. 1989](#)
- [Suboptimal structures within an energy band around the MFE](#)
- [Stochastic backtracking in the Ensemble](#)

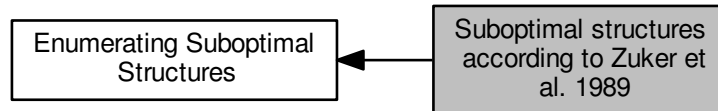
Files

- file [subopt.h](#)
RNAsubopt and density of states declarations.

11.12.1 Detailed Description

11.13 Suboptimal structures according to Zuker et al. 1989

Collaboration diagram for Suboptimal structures according to Zuker et al. 1989:



Functions

- **SOLUTION** * `zuckersubopt` (const char *string)
Compute Zuker type suboptimal structures.
- **SOLUTION** * `zuckersubopt_par` (const char *string, `vrna_param_t` *parameters)
Compute Zuker type suboptimal structures.
- **SOLUTION** * `vrna_zuckersubopt` (`vrna_fold_compound` *vc)
Compute Zuker type suboptimal structures.

11.13.1 Detailed Description

11.13.2 Function Documentation

11.13.2.1 **SOLUTION*** `zuckersubopt` (const char * *string*)

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_zuckersubopt()` instead

Parameters

<i>string</i>	RNA sequence
---------------	--------------

Returns

List of zuker suboptimal structures

11.13.2.2 **SOLUTION*** `zuckersubopt_par` (const char * *string*, `vrna_param_t` * *parameters*)

Compute Zuker type suboptimal structures.

Deprecated use `vrna_zuckersubopt()` instead

11.13.2.3 SOLUTION* vrna_zukersubopt (vrna_fold_compound * vc)

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.

Parameters

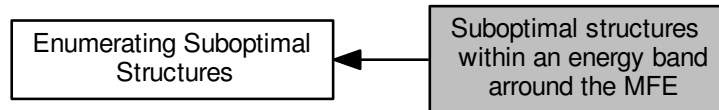
vc	fold compound
----	---------------

Returns

List of zuker suboptimal structures

11.14 Suboptimal structures within an energy band around the MFE

Collaboration diagram for Suboptimal structures within an energy band around the MFE:



Functions

- **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

- int **subopt_sorted**
Sort output by energy.
- double **print_energy**
printing threshold for use with logML

11.14.1 Detailed Description

11.14.2 Function Documentation

11.14.2.1 **SOLUTION*** **subopt** (char * *seq*, char * *structure*, int *delta*, FILE * *fp*)

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

<i>seq</i>	
<i>structure</i>	
<i>delta</i>	

<i>fp</i>	
-----------	--

Returns

11.14.2.2 **SOLUTION*** `subopt_circ (char * seq, char * sequence, int delta, FILE * fp)`

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

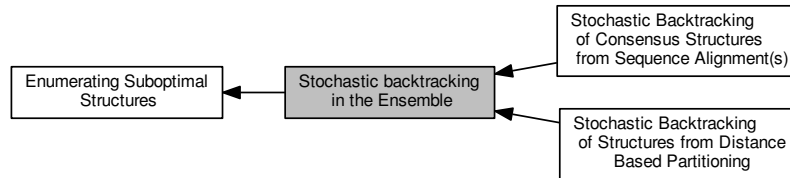
Parameters

<i>seq</i>	
<i>sequence</i>	
<i>delta</i>	
<i>fp</i>	

Returns

11.15 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 *vc, int length)`
Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.
- `char * vrna_pbacktrack (vrna_fold_compound *vc)`
Sample a secondary 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 auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.

11.15.1 Detailed Description

11.15.2 Function Documentation

11.15.2.1 `char* vrna_pbacktrack5 (vrna_fold_compound * vc, int length)`

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_get_fold_compound()`
`vrna_pf_fold()` has to be called first to fill the partition function matrices

Parameters

<i>vc</i>	The fold compound data structure
<i>length</i>	The length of the subsequence to consider (starting with 5' end)

Returns

A sampled secondary structure in dot-bracket notation

11.15.2.2 `char* vrna_pbacktrack (vrna_fold_compound * vc)`

Sample a secondary structure from the Boltzmann ensemble according its probability.

Precondition

The fold compound has to be obtained using the #VRNA_OPTION_HYBRID option in [vrna_get_fold_compound\(\)](#)
[vrna_pf_fold\(\)](#) has to be called first to fill the partition function matrices

Note

The function will automagically detect cicular RNAs based on the model_details in exp_params as provided via the [vrna_fold_compound](#)

Parameters

<i>vc</i>	The fold compound data structure
<i>length</i>	The length of the subsequence to consider (starting with 5' end)

Returns

A sampled secondary structure in dot-bracket notation

11.15.2.3 `char* pbacktrack (char * sequence)`

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

<i>sequence</i>	The RNA sequence
-----------------	------------------

Returns

A sampled secondary structure in dot-bracket notation

11.15.2.4 `char* pbacktrack_circ (char * sequence)`

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

<i>sequence</i>	The RNA sequence
-----------------	------------------

Returns

A sampled secondary structure in dot-bracket notation

11.15.3 Variable Documentation

11.15.3.1 int `st_back`

Flag indicating that auxiliary 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_get_fold_compound()`.

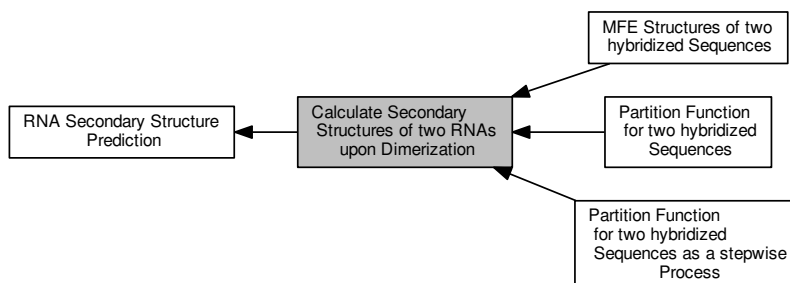
See also

`pbacktrack()`, `pbacktrack_circ`

11.16 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.

11.16.1 Detailed Description

Predict structures formed by two molecules upon hybridization.

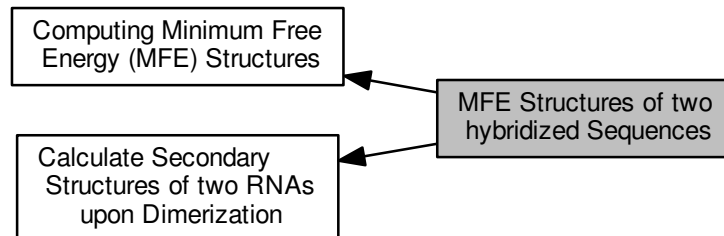
The function of an RNA molecule often depends on its interaction with other RNAs. The following routines therefore allow to predict structures formed by two RNA molecules upon hybridization.

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.

11.17 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 [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.
- float [vrna_cofold](#) ([vrna_fold_compound](#) *vc, char *structure)
Compute the minimum free energy of two interacting RNA molecules.
- 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 [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 quadruplex 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.

11.17.1 Detailed Description

11.17.2 Function Documentation

11.17.2.1 float cofold (const char * *sequence*, char * *structure*)

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_cofold\(\)](#) instead

Parameters

<i>sequence</i>	The two sequences concatenated
<i>structure</i>	Will hold the bracket dot structure of the dimer molecule

Returns

minimum free energy of the structure

11.17.2.2 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.

Deprecated use [vrna_cofold\(\)](#) instead

11.17.2.3 float vrna_cofold (vrna_fold_compound * *vc*, char * *structure*)

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the [vrna_fold\(\)](#) function.

Parameters

<i>vc</i>	fold compound
<i>structure</i>	Will hold the bracket dot structure of the dimer molecule

Returns

minimum free energy of the structure

11.17.2.4 char* vrna_cut_point_insert (const char * *string*, int *cp*)

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

<i>string</i>	The original string
---------------	---------------------

<i>cp</i>	The cut-point position
-----------	------------------------

Returns

A copy of the provided string including the cut-point character

11.17.2.5 `char* vrna_cut_point_remove (const char * string, int * cp)`

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

<i>string</i>	The original string
<i>cp</i>	The cut-point position

Returns

A copy of the input string with the '&' being sliced out

11.17.2.6 `void free_co_arrays (void)`

Free memory occupied by [cofold\(\)](#)

Deprecated This function will only free memory allocated by a prior call of [cofold\(\)](#) or [cofold_par\(\)](#). See [vrna_↔cofold\(\)](#) 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_cofold\(\)](#)

11.17.2.7 `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)`

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_cofold\(\)](#) for the new API

Parameters

<i>f5_p</i>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<i>c_p</i>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<i>fML_p</i>	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
<i>fM1_p</i>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<i>fc_p</i>	A pointer to the 'fc' array, i.e. array ...
<i>ggg_p</i>	A pointer to the 'ggg' array, i.e. array containing best free energy of a quadruplex delimited by [i..j]
<i>indx_p</i>	A pointer to the indexing array used for accessing the energy matrices
<i>pctype_p</i>	A pointer to the pctype array containing the base pair types for each possibility (i,j)

11.17.2.8 `void export_cofold_arrays (int ** f5_p, int ** c_p, int ** fML_p, int ** fM1_p, int ** fc_p, int ** indx_p, char ** pctype_p)`

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 fold compound. Thus, this function will only work in conjunction with a prior call to `cofold()` or `cofold_par()`

See also

[vrna_cofold\(\)](#) for the new API

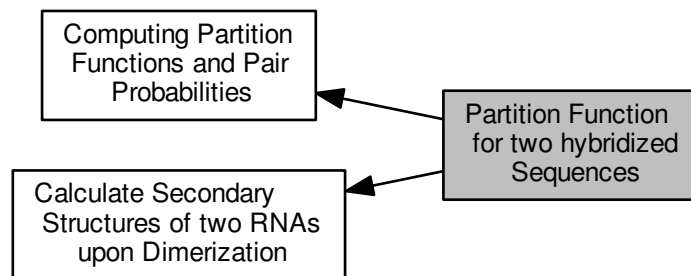
Parameters

<i>f5_p</i>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<i>c_p</i>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<i>fML_p</i>	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
<i>fM1_p</i>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<i>fc_p</i>	A pointer to the 'fc' array, i.e. array ...
<i>indx_p</i>	A pointer to the indexing array used for accessing the energy matrices
<i>pctype_p</i>	A pointer to the pctype array containing the base pair types for each possibility (i,j)

11.18 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.

Functions

- [cofoldF vrna_co_pf_fold](#) ([vrna_fold_compound](#) *vc, char *structure)
Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.
- void [vrna_co_pf_dimer_probs](#) (double FAB, double FA, double FB, struct [plist](#) *prAB, const [plist](#) *prA, const [plist](#) *prB, int Alength, const [vrna_exp_param_t](#) *exp_params)
Compute Boltzmann probabilities of dimerization without homodimers.
- [ConcEnt](#) * [vrna_co_pf_get_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.

11.18.1 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.

11.18.2 Function Documentation

11.18.2.1 `cofoldF vrna_co_pf_fold (vrna_fold_compound * vc, char * structure)`

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This is the cofold partition function folding.

See also

[vrna_get_fold_compound\(\)](#) for how to retrieve the necessary data structure

Parameters

<code>vc</code>	the fold compound data structure
<code>structure</code>	Will hold the structure or constraints

Returns

`cofoldF` structure containing a set of energies needed for concentration computations.

11.18.2.2 `void vrna_co_pf_dimer_probs (double FAB, double FA, double FB, struct plist * prAB, const plist * prA, const plist * prB, int Alength, const vrna_exp_param_t * exp_params)`

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.

Parameters

<code>FAB</code>	free energy of dimer AB
<code>FEA</code>	free energy of monomer A
<code>FEB</code>	free energy of monomer B
<code>prAB</code>	pair probabilities for dimer
<code>prA</code>	pair probabilities monomer
<code>prB</code>	pair probabilities monomer
<code>Alength</code>	Length of molecule A
<code>exp_params</code>	The precomputed Boltzmann factors

11.18.2.3 `ConcEnt* vrna_co_pf_get_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.

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 `cofoldF` struct.

Parameters

<i>FEAB</i>	Free energy of AB dimer (FcAB entry)
<i>FEAA</i>	Free energy of AA dimer (FcAB entry)
<i>FEBB</i>	Free energy of BB dimer (FcAB entry)
<i>FEA</i>	Free energy of monomer A
<i>FEB</i>	Free energy of monomer B
<i>startconc</i>	List of start concentrations [a0],[b0],[a1],[b1],...,[an],[bn],[0],[0]
<i>exp_params</i>	The precomputed Boltzmann factors

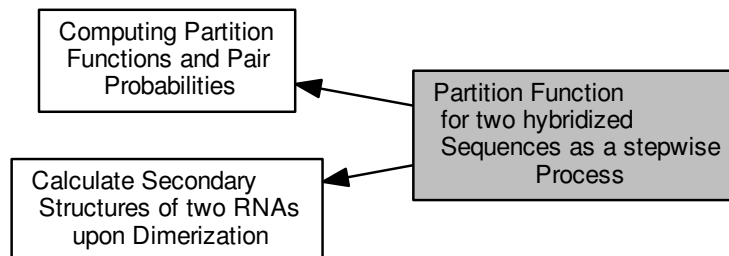
Returns

[ConcEnt](#) array containing the equilibrium energies and start concentrations

11.19 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\(\)](#).

11.19.1 Detailed Description

Partition Function Cofolding as a stepwise process.

11.19.2 Function Documentation

11.19.2.1 [pu_contrib](#)* [pf_unstru](#) (char * *sequence*, int *max_w*)

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

<i>sequence</i>	
<i>max_w</i>	

Returns

11.19.2.2 `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)`

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 constraints for intermolecular interaction are: '.' (no constrain), 'x' (the base has no intermolecular interaction) and '|' (the corresponding base has to be paired intermolecularly). 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 P_i and the minimum free energy in G_i , where *i* is the position in sequence 's1'. The member G_{ikjl} 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'. G_{ikjl_wo} is G_{ikjl} without the probability of being unpaired.

Use `free_interact()` to free the returned structure, all other stuff is freed inside `pf_interact()`.

Parameters

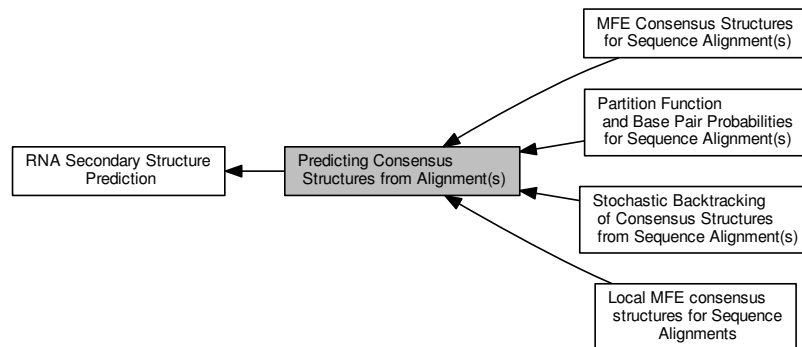
<i>s1</i>	
<i>s2</i>	
<i>p_c</i>	
<i>p_c2</i>	
<i>max_w</i>	
<i>cstruc</i>	
<i>incr3</i>	
<i>incr5</i>	

Returns

11.20 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, int **pscore)
Get pointers to (almost) all relevant arrays used in alifold's partition function computation.
- void [update_alifold_params](#) (void)
Update the energy parameters for alifold function.
- int [vrna_alifold_get_mpi](#) (char *Aseq[], int n_seq, int length, int *mini)
Get the mean pairwise identity in steps from ?to?(ident)
- int [get_mpi](#) (char *Aseq[], int n_seq, int length, int *mini)
Get the mean pairwise identity in steps from ?to?(ident)
- void [encode_alifold_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.

11.20.1 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 [\[5\]](#) and [\[1\]](#).

11.20.2 Function Documentation

11.20.2.1 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.

Deprecated Usage of this function is discouraged! Use [vrna_eval_structure\(\)](#), and [vrna_eval_covar_structure\(\)](#) instead!

Parameters

<i>sequences</i>	The NULL terminated array of sequences
<i>structure</i>	The consensus structure
<i>n_seq</i>	The number of sequences in the alignment
<i>energy</i>	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

11.20.2.2 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, int ** pscore)

Get pointers to (almost) all relevant 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](#) allows direct access to all necessary consensus structure prediction related variables!

See also

[vrna_fold_compound](#), [vrna_get_fold_compound_alifold\(\)](#), [vrna_alifold_fold\(\)](#)

Parameters

<i>S_p</i>	A pointer to the 'S' array (integer representation of nucleotides)
<i>S5_p</i>	A pointer to the 'S5' array
<i>S3_p</i>	A pointer to the 'S3' array
<i>a2s_p</i>	A pointer to the pair type matrix
<i>Ss_p</i>	A pointer to the 'Ss' array
<i>qb_p</i>	A pointer to the Q^B matrix
<i>qm_p</i>	A pointer to the Q^M matrix
<i>q1k_p</i>	A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1, k)$)
<i>qln_p</i>	A pointer to the 3' slice of the Q matrix ($qln(l) = Q(l, n)$)

Returns

Non Zero if everything went fine, 0 otherwise

11.20.2.3 void update_alifold_params (void)

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](#) 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](#).

11.20.2.4 int vrna_alifold_get_mpi (char * Aseq[], int n_seq, int length, int * mini)

Get the mean pairwise identity in steps from ?to?(ident)

Parameters

<i>Aseq</i>	
<i>n_seq</i>	The number of sequences in the alignment
<i>length</i>	The length of the alignment
<i>mini</i>	

Returns

The mean pairwise identity

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

Get the mean pairwise identity in steps from ?to?(ident)

Deprecated Use `vrna.ali.get_mpi()` as a replacement

Parameters

<i>Alseq</i>	
<i>n_seq</i>	The number of sequences in the alignment
<i>length</i>	The length of the alignment
<i>mini</i>	

Returns

The mean pairwise identity

11.20.2.6 `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.

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

<i>sequence</i>	The gapped sequence from the alignment
<i>S</i>	pointer to an array that holds encoded sequence
<i>s5</i>	pointer to an array that holds the next base 5' of alignment position i
<i>s3</i>	pointer to an array that holds the next base 3' of alignment position i
<i>ss</i>	
<i>as</i>	
<i>circ</i>	assume the molecules to be circular instead of linear (circ=0)

11.20.2.7 `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.

Note that these arrays will also be initialized according to the sequence alignment given

See also

[free_sequence_arrays\(\)](#)

Parameters

<i>sequences</i>	The aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position

<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	A pointer to the array that contains the ungapped sequence
<i>circ</i>	assume the molecules to be circular instead of linear (<i>circ</i> =0)

11.20.2.8 `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.

This function frees the memory previously allocated with [alloc_sequence_arrays\(\)](#)

See also

[alloc_sequence_arrays\(\)](#)

Parameters

<i>n_seq</i>	The number of aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position
<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	A pointer to the array that contains the ungapped sequence

11.20.3 Variable Documentation

11.20.3.1 `double cv_fact`

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_ali_fold\(\)](#) to avoid using global variables

Default is 1.

11.20.3.2 `double nc_fact`

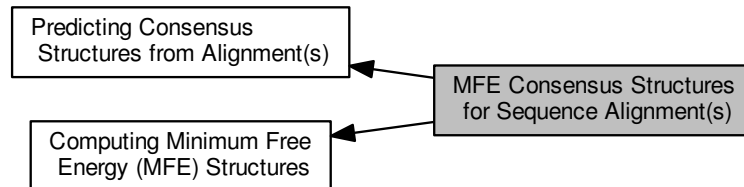
This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

Deprecated See [#vrna_md_t.nc_fact](#), and [vrna_ali_fold\(\)](#) to avoid using global variables

Default is 1.

11.21 MFE Consensus Structures for Sequence Alignment(s)

Collaboration diagram for MFE Consensus Structures for Sequence Alignment(s):



Functions

- float [vrna_alifold](#) ([vrna_fold_compound](#) *vc, char *structure)
Compute MFE and according consensus structure of an alignment of sequences.
- float [alifold](#) (const char **strings, char *structure)
Compute MFE and according consensus structure of an alignment of sequences.
- float [circularifold](#) (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.

11.21.1 Detailed Description

11.21.2 Function Documentation

11.21.2.1 float vrna_alifold (vrna_fold_compound * vc, char * structure)

Compute MFE and according consensus structure of an alignment of sequences.

This function predicts the consensus structure for the alignment stored in vc and returns the minimum free energy; the mfe structure in bracket notation is returned in 'structure'.

Note

vc has to be of type [VRNA_VC_TYPE_ALIGNMENT](#).

Sufficient space must be allocated for 'structure' before calling [vrna_alifold\(\)](#). Passing NULL to the 'structure' or setting [vrna_md_t.backtrack](#) to 0 turns of backtracing and no structure will be returned.

See also

[vrna_get_fold_compound_alifold\(\)](#)

Parameters

<i>vc</i>	The fold compound structure of type VRNA_VC_TYPE_ALIGNMENT
<i>structure</i>	A pointer to a character array that will be overwritten by a consensus structure that exhibits the MFE. (Maybe NULL)

Returns

The free energy score in kcal/mol

11.21.2.2 float alifold (const char ** *strings*, char * *structure*)

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\(\)](#) instead

See also

[vrna_alifold\(\)](#)

Parameters

<i>strings</i>	A pointer to a NULL terminated array of character arrays
<i>structure</i>	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

11.21.2.3 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.

Deprecated Usage of this function is discouraged! Use [vrna_alifold\(\)](#) instead

See also

[vrna_alifold\(\)](#)

Parameters

<i>strings</i>	A pointer to a NULL terminated array of character arrays
<i>structure</i>	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

11.21.2.4 void free_alifold_arrays (void)

Free the memory occupied by MFE alifold functions.

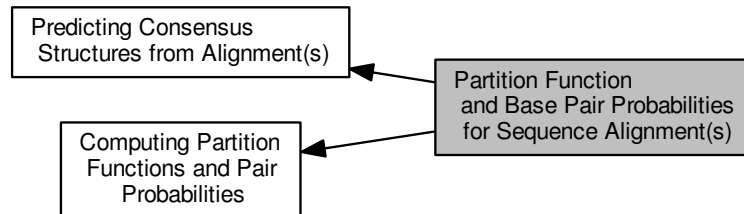
Deprecated Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced [vrna_fold_compound](#) is handled by `vrna_vrna_free_fold_compound()`

See also

`vrna_vrna_free_fold_compound()`

11.22 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_ali_pf_fold` (`vrna_fold_compound` *vc, char *structure, `plist` **pl)

Compute partition function and base pair probabilities for a sequence alignment.
- float `alipf_fold_par` (const char **sequences, char *structure, `plist` **pl, `vrna_exp_param_t` *parameters, int calculate_bppm, int is_constrained, int is_circular)
- float `alipf_fold` (const char **sequences, char *structure, `plist` **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 `pair_info` structs. The list is terminated by the first entry with `pi.i = 0`.
- float `alipf_circ_fold` (const char **sequences, char *structure, `plist` **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.

11.22.1 Detailed Description

11.22.2 Function Documentation

11.22.2.1 float `vrna_ali_pf_fold` (`vrna_fold_compound` * vc, char * *structure*, `plist` ** *pl*)

Compute partition function and base pair probabilities for a sequence alignment.

The partition function version of `vrna_ali_fold()` works in analogy to `vrna_pf_fold()`. Pair probabilities are returned via the 'pl' variable as a list of `plist` structs. The list is terminated by the first entry with `pl.i = 0`.

See also

`vrna_ali_get_pair_info()` for a replacement of `pl` with more detailed information

Parameters

<i>vc</i>	The vrna_fold_compound of type VRNA_VC_TYPE_ALIGNMENT
<i>structure</i>	A pointer to a character array of length of the alignment (Maybe NULL)
<i>pl</i>	A pointer to a plist pointer where the pair probabilities are stored (Maybe NULL)

Returns

Gibbs free energy of the consensus fold space

11.22.2.2 `float alipf_fold_par (const char ** sequences, char * structure, plist ** pl, vrna_exp_param_t * parameters, int calculate_bppm, int is_constrained, int is_circular)`

Deprecated Use [vrna_ali_pf_fold\(\)](#) instead

Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	
<i>parameters</i>	
<i>calculate_bppm</i>	
<i>is_constrained</i>	
<i>is_circular</i>	

Returns

11.22.2.3 `float alipf_fold (const char ** sequences, char * structure, plist ** 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 [pair_info](#) structs. The list is terminated by the first entry with pi.i = 0.

Deprecated Use [vrna_ali_pf_fold\(\)](#) instead

Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

Returns

11.22.2.4 `float alipf_circ_fold (const char ** sequences, char * structure, plist ** pl)`

Deprecated Use [vrna_ali_pf_fold\(\)](#) instead

Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

Returns

11.22.2.5 FLT_OR_DBL* export_alibppm (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 Usage of this function is discouraged! The new [vrna_fold_compound](#) 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_alibpf_fold\(\)](#), or any of the old API calls for consensus structure partition function folding.

See also

[vrna_fold_compound](#), [vrna_get_fold_compound_alibpf\(\)](#), and [vrna_alibpf_fold\(\)](#)

Returns

A pointer to the base pair probability array

11.22.2.6 void free_alibpf_arrays (void)

Free the memory occupied by folding matrices allocated by [alibpf_fold](#), [alibpf_circ_fold](#), etc.

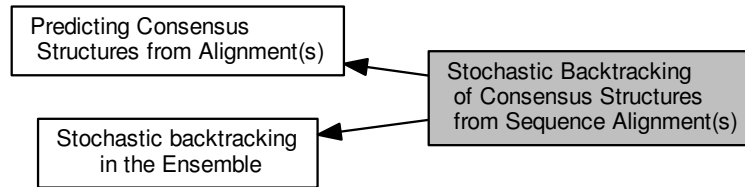
Deprecated Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with [vrna_](#)) will be not affected!

See also

[vrna_fold_compound](#), [vrna_vrna_free_fold_compound\(\)](#)

11.23 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)

Collaboration diagram for Stochastic Backtracking of Consensus Structures from Sequence Alignment(s):



Functions

- `char * vrna_alipbacktrack (vrna_fold_compound *vc, double *prob)`
Sample a consensus secondary structure from the Boltzmann ensemble according its probability
- `char * alipbacktrack (double *prob)`
Sample a consensus secondary structure from the Boltzmann ensemble according its probability

11.23.1 Detailed Description

11.23.2 Function Documentation

11.23.2.1 `char* vrna_alipbacktrack (vrna_fold_compound * vc, double * prob)`

Sample a consensus secondary structure from the Boltzmann ensemble according its probability

See also

[vrna_alipf_fold\(\)](#) for precomputing the partition function matrices, and

Parameters

<i>vc</i>	The vrna_fold_compound of type VRNA_VC_TYPE_ALIGNMENT with precomputed partition function matrices
<i>prob</i>	to be described (berni)

Returns

A sampled consensus secondary structure in dot-bracket notation

11.23.2.2 `char* alipbacktrack (double * prob)`

Sample a consensus secondary structure from the Boltzmann ensemble according its probability

Deprecated Use [vrna_alipbacktrack\(\)](#) instead!

Parameters

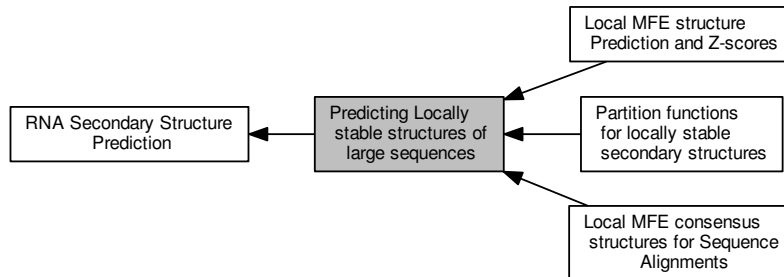
<i>prob</i>	to be described (berni)
-------------	-------------------------

Returns

A sampled consensus secondary structure in dot-bracket notation

11.24 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

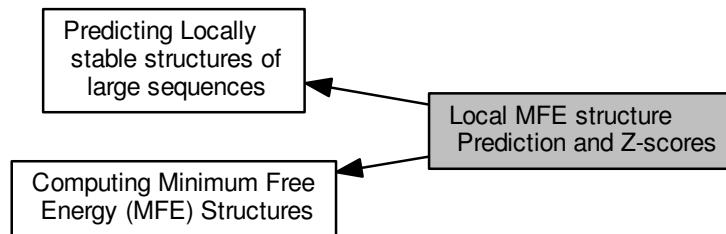
- file [Lfold.h](#)
Predicting local MFE structures of large sequences.

11.24.1 Detailed Description

Local structures can be predicted by a modified version of the [fold\(\)](#) algorithm that restricts the span of all base pairs.

11.25 Local MFE structure Prediction and Z-scores

Collaboration diagram for Local MFE structure Prediction and Z-scores:



Functions

- 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)

11.25.1 Detailed Description

11.25.2 Function Documentation

11.25.2.1 float [Lfold](#) (const char * *string*, char * *structure*, int *maxdist*)

The local analog to [fold\(\)](#).

Computes the minimum free energy structure including only base pairs with a span smaller than 'maxdist'

Parameters

<i>string</i>	
<i>structure</i>	
<i>maxdist</i>	

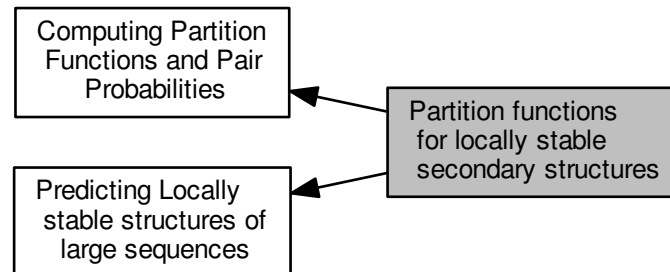
11.25.2.2 float [Lfoldz](#) (const char * *string*, char * *structure*, int *maxdist*, int *zsc*, double *min_z*)

Parameters

<i>string</i>	
<i>structure</i>	
<i>maxdist</i>	
<i>zsc</i>	
<i>min_z</i>	

11.26 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, struct [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, struct [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.

11.26.1 Detailed Description

11.26.2 Function Documentation

11.26.2.1 void update_pf_paramsLP (int length)

Parameters

<i>length</i>	
---------------	--

11.26.2.2 **plist*** `pfl_fold` (*char* * *sequence*, *int* *winSize*, *int* *pairSize*, *float* *cutoffb*, *double* ** *pU*, *struct plist* ** *dpp2*, *FILE* * *pUfp*, *FILE* * *spup*)

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 \geq '*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 immediately. If *spup* is given (i.e. is not NULL), the pair probabilities in *pl* are not saved but put out immediately.

Parameters

<i>sequence</i>	RNA sequence
<i>winSize</i>	size of the window
<i>pairSize</i>	maximum size of base pair
<i>cutoffb</i>	cutoffb for base pairs
<i>pU</i>	array holding all unpaired probabilities
<i>dpp2</i>	array of dependent pair probabilities
<i>pUfp</i>	file pointer for <i>pU</i>
<i>spup</i>	file pointer for pair probabilities

Returns

list of pair probabilities

11.26.2.3 **void** `putoutpU_prob` (*double* ** *pU*, *int* *length*, *int* *ulength*, *FILE* * *fp*, *int* *energies*)

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

<i>pU</i>	pair probabilities
<i>length</i>	length of RNA sequence
<i>ulength</i>	maximum length of unpaired stretch
<i>fp</i>	file pointer of destination file
<i>energies</i>	switch to put out as opening energies

11.26.2.4 **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.

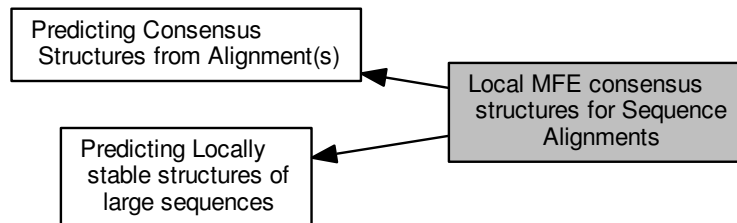
Can write either the unpaired probabilities (accessibilities) *pU* or the opening energies $-\log(pU)kT$ into a file

Parameters

<i>pU</i>	pair probabilities
<i>length</i>	length of RNA sequence
<i>ulength</i>	maximum length of unpaired stretch
<i>fp</i>	file pointer of destination file
<i>energies</i>	switch to put out as opening energies

11.27 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)

11.27.1 Detailed Description

11.27.2 Function Documentation

11.27.2.1 float aliLfold (const char ** *strings*, char * *structure*, int *maxdist*)

Parameters

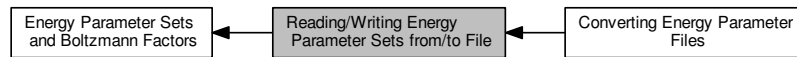
<i>strings</i>	
<i>structure</i>	
<i>maxdist</i>	

Returns

11.28 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.

11.28.1 Detailed Description

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

A default set of parameters, identical to the one described in [10] and [14], is compiled into the library.

11.28.2 Function Documentation

11.28.2.1 void [read_parameter_file](#) (const char *fname*[])

Read energy parameters from a file.

Parameters

<i>fname</i>	The path to the file containing the energy parameters
--------------	---

11.28.2.2 void [write_parameter_file](#) (const char *fname*[])

Write energy parameters to a file.

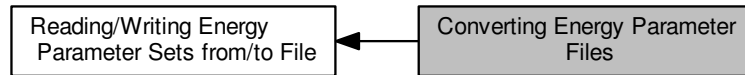
Parameters

<i>fname</i>	A filename (path) for the file where the current energy parameters will be written to
--------------	---

11.29 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)

11.29.1 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

11.29.2 Macro Definition Documentation

11.29.2.1 `#define VRNA_CONVERT_OUTPUT_ALL 1U`

Flag to indicate printing of a complete parameter set

11.29.2.2 `#define VRNA_CONVERT_OUTPUT_HP 2U`

Flag to indicate printing of hairpin contributions

11.29.2.3 `#define VRNA_CONVERT_OUTPUT_STACK 4U`

Flag to indicate printing of base pair stack contributions

11.29.2.4 `#define VRNA_CONVERT_OUTPUT_MM_HP 8U`

Flag to indicate printing of hairpin mismatch contribution

11.29.2.5 `#define VRNA_CONVERT_OUTPUT_MM_INT 16U`

Flag to indicate printing of interior loop mismatch contribution

11.29.2.6 `#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U`

Flag to indicate printing of 1:n interior loop mismatch contribution

11.29.2.7 `#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U`

Flag to indicate printing of 2:3 interior loop mismatch contribution

11.29.2.8 `#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U`

Flag to indicate printing of multi loop mismatch contribution

11.29.2.9 `#define VRNA_CONVERT_OUTPUT_MM_EXT 256U`

Flag to indicate printing of exterior loop mismatch contribution

11.29.2.10 `#define VRNA_CONVERT_OUTPUT_DANGLE5 512U`

Flag to indicate printing of 5' dangle contribution

11.29.2.11 `#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U`

Flag to indicate printing of 3' dangle contribution

11.29.2.12 `#define VRNA_CONVERT_OUTPUT_INT_11 2048U`

Flag to indicate printing of 1:1 interior loop contribution

11.29.2.13 `#define VRNA_CONVERT_OUTPUT_INT_21 4096U`

Flag to indicate printing of 2:1 interior loop contribution

11.29.2.14 `#define VRNA_CONVERT_OUTPUT_INT_22 8192U`

Flag to indicate printing of 2:2 interior loop contribution

11.29.2.15 `#define VRNA_CONVERT_OUTPUT_BULGE 16384U`

Flag to indicate printing of bulge loop contribution

11.29.2.16 `#define VRNA_CONVERT_OUTPUT_INT 32768U`

Flag to indicate printing of interior loop contribution

11.29.2.17 `#define VRNA_CONVERT_OUTPUT_ML 65536U`

Flag to indicate printing of multi loop contribution

11.29.2.18 `#define VRNA_CONVERT_OUTPUT_MISC 131072U`

Flag to indicate printing of misc contributions (such as terminalAU)

11.29.2.19 `#define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U`

Flag to indicate printing of special hairpin contributions (tri-, tetra-, hexa-loops)

11.29.2.20 `#define VRNA_CONVERT_OUTPUT_VANILLA 524288U`

Flag to indicate printing of given parameters only

Note

This option overrides all other output options, except [VRNA_CONVERT_OUTPUT_DUMP](#) !

11.29.2.21 `#define VRNA_CONVERT_OUTPUT_NINIO 1048576U`

Flag to indicate printing of interior loop asymmetry contribution

11.29.2.22 `#define VRNA_CONVERT_OUTPUT_DUMP 2097152U`

Flag to indicate dumping the energy contributions from the library instead of an input file

11.29.3 Function Documentation

11.29.3.1 `void convert_parameter_file (const char * iname, const char * oname, unsigned int options)`

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_OUTPUT_MM_EXT
 VRNA_CONVERT_OUTPUT_DANGLE5, VRNA_CONVERT_OUTPUT_DANGLE3, VRNA_CONVERT_OUTPUT_INT_11
 VRNA_CONVERT_OUTPUT_INT_21, VRNA_CONVERT_OUTPUT_INT_22, VRNA_CONVERT_OUTPUT_BULGE
 VRNA_CONVERT_OUTPUT_INT, VRNA_CONVERT_OUTPUT_ML, VRNA_CONVERT_OUTPUT_MISC
 VRNA_CONVERT_OUTPUT_SPECIAL_HP, VRNA_CONVERT_OUTPUT_VANILLA, VRNA_CONVERT_OUTPUT_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:

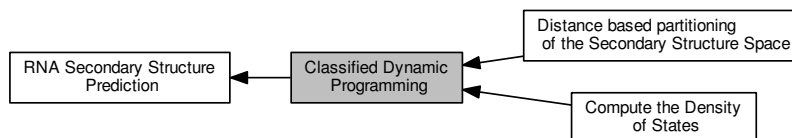
```
convert_parameter_file(ifile, ofile, option_1 | option_2 | option_n)
```

Parameters

<i>iname</i>	The input file name (If NULL input is read from stdin)
<i>oname</i>	The output file name (If NULL output is written to stdout)
<i>options</i>	The options (as described above)

11.30 Classified Dynamic Programming

Collaboration diagram for Classified Dynamic Programming:



Modules

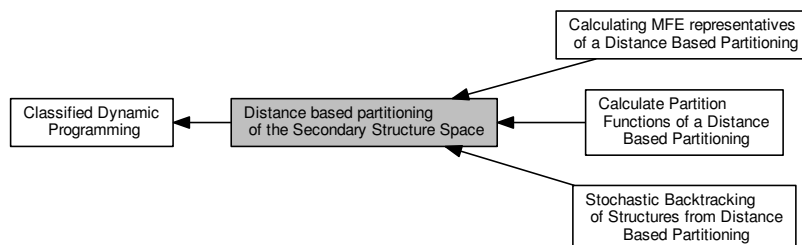
- [Distance based partitioning of the Secondary Structure Space](#)
Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.
- [Compute the Density of States](#)

11.30.1 Detailed Description

11.31 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.

11.31.1 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

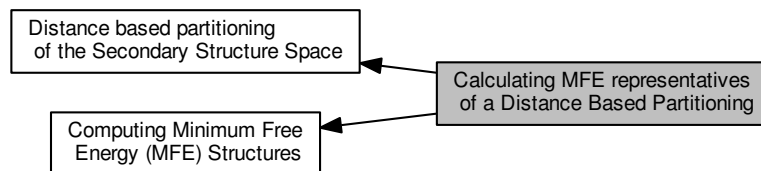
See also

For further details have a look into [9]

11.32 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_TwoD_fold\(\)](#)
- struct [TwoDfold_vars](#)
Variables compound for 2Dfold MFE folding.

Functions

- [vrna_sol_TwoD_t](#) * [vrna_TwoD_fold](#) ([vrna_fold_compound](#) *vc, int distance1, int distance2)
Compute MFE's and representative for distance partitioning.
- char * [vrna_TwoD_backtrack5](#) ([vrna_fold_compound](#) *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.

11.32.1 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.

11.32.2 Function Documentation

11.32.2.1 `vrna_sol_TwoD_t* vrna_TwoD_fold (vrna_fold_compound * vc, int distance1, int distance2)`

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 $k=-1$ 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_get_fold_compound_2D\(\)](#), [vrna_free_fold_compound\(\)](#), [vrna_TwoD_pf_fold\(\)](#) [vrna_TwoD_backtrack5\(\)](#), [vrna_sol_TwoD_t](#), [vrna_fold_compound](#)

Parameters

<code>vc</code>	The datastructure containing all precomputed folding attributes
<code>distance1</code>	maximum distance to reference1 (-1 means no restriction)
<code>distance2</code>	maximum distance to reference2 (-1 means no restriction)

Returns

A list of minimum free energies (and corresponding structures) for each distance class

11.32.2.2 `char* vrna_TwoD_backtrack5 (vrna_fold_compound * vc, int k, int l, unsigned int j)`

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_TwoD_fold\(\)](#) belong to.

Note

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to [vrna_TwoD_fold\(\)](#) preceding this function is mandatory!

See also

[vrna_TwoD_fold\(\)](#)

Parameters

<code>vc</code>	The datastructure containing all precomputed folding attributes
<code>j</code>	The length in nucleotides beginning from the 5' end
<code>k</code>	distance to reference1 (may be -1)

/	distance to reference2
---	------------------------

11.32.2.3 `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.

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` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_fold()`, and `vrna_free_fold_compound()` instead!

Parameters

<code>seq</code>	The RNA sequence
<code>structure1</code>	The first reference structure in dot-bracket notation
<code>structure2</code>	The second reference structure in dot-bracket notation
<code>circ</code>	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

11.32.2.4 `void destroy_TwoDfold_variables (TwoDfold_vars * our_variables)`

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` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_fold()`, and `vrna_free_fold_compound()` instead!

Parameters

<code>our_variables</code>	A pointer to the datastructure to be destroyed
----------------------------	--

11.32.2.5 `vrna_sol_TwoD_t* TwoDfoldList (TwoDfold_vars * vars, int distance1, int distance2)`

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=-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](#) and the corresponding functions [vrna_get_fold_compound_2D\(\)](#), [vrna_TwoD_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

Parameters

<i>vars</i>	the datastructure containing all predefined folding attributes
<i>distance1</i>	maximum distance to reference1 (-1 means no restriction)
<i>distance2</i>	maximum distance to reference2 (-1 means no restriction)

11.32.2.6 `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.

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_TwoDfold\(\)](#) belong to.

Note

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to [vrna_TwoDfold\(\)](#) preceding this function is mandatory!

Deprecated Use the new API that relies on [vrna_fold_compound](#) and the corresponding functions [vrna_get_fold_compound_2D\(\)](#), [vrna_TwoD_fold\(\)](#), [vrna_TwoD_backtrack5\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

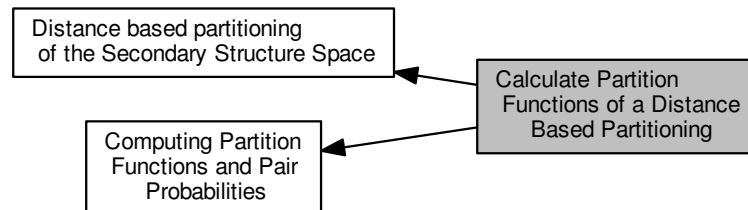
Parameters

<i>j</i>	The length in nucleotides beginning from the 5' end
<i>k</i>	distance to reference1 (may be -1)
<i>l</i>	distance to reference2
<i>vars</i>	the datastructure containing all predefined folding attributes

11.33 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_TwoD_pf_fold\(\)](#)

Functions

- [vrna_sol_TwoD_pf_t * vrna_TwoD_pf_fold \(vrna_fold_compound *vc, int maxDistance1, int maxDistance2\)](#)
Compute the partition function for all distance classes.

11.33.1 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.

11.33.2 Function Documentation

11.33.2.1 `vrna_sol_TwoD_pf_t* vrna_TwoD_pf_fold (vrna_fold_compound * vc, 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 [vrna_TwoD_fold\(\)](#) 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 distances 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=-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_get_fold_compound_2D()`, [vrna_free_fold_compound\(\)](#), [vrna_fold_compound](#) [vrna_sol_TwoD_pf_t](#)

Parameters

<code>vc</code>	The datastructure containing all necessary folding attributes and matrices
<code>maxDistance1</code>	The maximum basepair distance to reference1 (may be -1)
<code>maxDistance2</code>	The maximum basepair distance to reference2 (may be -1)

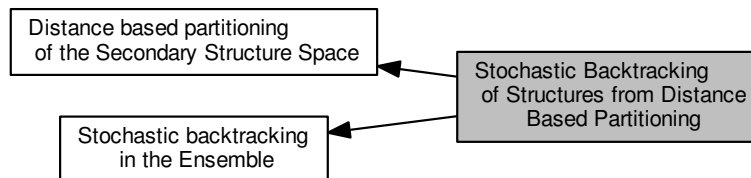
Returns

A list of partition funtions for the corresponding distance classes

11.34 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_TwoD_pbacktrack (vrna_fold_compound *vc, int d1, int d2)`
Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.
- `char * vrna_TwoD_pbacktrack5 (vrna_fold_compound *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.

11.34.1 Detailed Description

Contains functions related to stochastic backtracking from a specified distance class.

11.34.2 Function Documentation

11.34.2.1 `char* vrna_TwoD_pbacktrack (vrna_fold_compound * vc, 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 `vrna_TwoD_pf_fold()` preceding this function is mandatory!

See also

[vrna_TwoD_pf_fold\(\)](#)

Parameters

in	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
in	<i>d1</i>	the distance to reference1 (may be -1)
in	<i>d2</i>	the distance to reference2

Returns

A sampled secondary structure in dot-bracket notation

11.34.2.2 `char* vrna_TwoD_pbacktrack5 (vrna_fold_compound * 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.

This function does essentially the same as [vrna_TwoD_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 [vrna_TwoD_pf_fold\(\)](#) preceding this function is mandatory!

See also

[vrna_TwoD_pbacktrack\(\)](#), [vrna_TwoD_pf_fold\(\)](#)

Parameters

in	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
in	<i>d1</i>	the distance to reference1 (may be -1)
in	<i>d2</i>	the distance to reference2
in	<i>length</i>	the length of the structure beginning from the 5' end

Returns

A sampled secondary structure in dot-bracket notation

11.35 Compute the Density of States

Collaboration diagram for Compute the Density of States:



Variables

- int [density_of_states](#) [MAXDOS+1]
The Density of States.

11.35.1 Detailed Description

11.35.2 Variable Documentation

11.35.2.1 int density_of_states[MAXDOS+1]

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

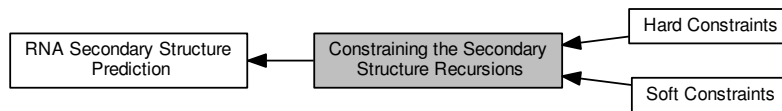
See also

[subopt_par\(\)](#), [subopt\(\)](#), [subopt_circ\(\)](#)

11.36 Constraining the Secondary Structure Recursions

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 Recursions:



Modules

- [Hard Constraints](#)
- [Soft Constraints](#)

Macros

- `#define VRNA_CONSTRAINT_DB_PIPE 1U`
Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.
- `#define VRNA_CONSTRAINT_DB_DOT 2U`
dot '.' switch for structure constraints (no constraint at all)
- `#define VRNA_CONSTRAINT_DB_X 4U`
'x' switch for structure constraint (base must not pair)
- `#define VRNA_CONSTRAINT_DB_ANG_BRACK 8U`
angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)
- `#define VRNA_CONSTRAINT_DB_RND_BRACK 16U`
round brackets '(', ')' switch for structure constraint (base i pairs base j)
- `#define VRNA_CONSTRAINT_DB_INTRAMOL 2048U`
Flag that is used to indicate the character 'I' in pseudo dot-bracket notation of hard constraints.
- `#define VRNA_CONSTRAINT_DB_INTERMOL 4096U`
Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.
- `#define VRNA_CONSTRAINT_DB_GQUAD 8192U`
'+' switch for structure constraint (base is involved in a gquad)
- `#define VRNA_CONSTRAINT_DB_ENFORCE_BP 16384U`
Switch for dot-bracket structure constraint to enforce base pairs.
- `#define VRNA_CONSTRAINT_ALL 128U`
placeholder for all constraining characters
- `#define VRNA_CONSTRAINT_DB 256U`
Flag for `vrna_add_constraints()` to indicate that constraint is passed in pseudo dot-bracket notation.
- `#define VRNA_CONSTRAINT_FILE 512U`
Flag for `vrna_add_constraints()` to indicate that constraints are present in a text file.

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_constraints_all](#) (void)
Print structure constraint characters to stdout (full constraint support)
- void [vrna_add_constraints](#) ([vrna_fold_compound](#) *vc, const char *constraint, unsigned int options)
Add constraints to a [vrna_fold_compound](#) data structure.

11.36.1 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.

11.36.2 Macro Definition Documentation

11.36.2.1 `#define VRNA_CONSTRAINT_DB_PIPE 1U`

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_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.2 `#define VRNA_CONSTRAINT_DB_DOT 2U`

dot '.' switch for structure constraints (no constraint at all)

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.3 `#define VRNA_CONSTRAINT_DB_X 4U`

'x' switch for structure constraint (base must not pair)

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.4 #define VRNA_CONSTRAINT_DB_ANG_BRACK 8U

angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.5 #define VRNA_CONSTRAINT_DB_RND_BRACK 16U

round brackets '(', ')' switch for structure constraint (base i pairs base j)

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.6 #define VRNA_CONSTRAINT_DB_INTRAMOL 2048U

Flag that is used to indicate the character 'I' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'I' character (intramolecular pairs only)

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.7 #define VRNA_CONSTRAINT_DB_INTERMOL 4096U

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_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.8 #define VRNA_CONSTRAINT_DB_GQUAD 8192U

'+' switch for structure constraint (base is involved in a gquad)

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

Warning

This flag is for future purposes only! No implementation recognizes it yet.

11.36.2.9 #define VRNA_CONSTRAINT_DB_ENFORCE_BP 16384U

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_add_constraints\(\)](#)

11.36.2.10 `#define VRNA_CONSTRAINT_DB 256U`

Flag for `vrna_add_constraints()` to indicate that constraint is passed in pseudo dot-bracket notation.

See also

[vrna_add_constraints\(\)](#), [vrna_message_constraint_options\(\)](#), [vrna_message_constraints_all\(\)](#)

11.36.2.11 `#define VRNA_CONSTRAINT_FILE 512U`

Flag for `vrna_add_constraints()` to indicate that constraints are present in a text file.

See also

[vrna_add_constraints\(\)](#)

11.36.3 Function Documentation

11.36.3.1 `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)

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_constraints_all\(\)](#), [vrna_add_constraints\(\)](#), [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

<i>option</i>	Option switch that tells which constraint help will be printed
---------------	--

11.36.3.2 `void vrna_message_constraints_all (void)`

Print structure constraint characters to stdout (full constraint support)

See also

[vrna_message_constraint_options\(\)](#), [vrna_add_constraints\(\)](#), [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](#)

11.36.3.3 void `vrna_add_constraints` (`vrna_fold_compound` * *vc*, const char * *constraint*, unsigned int *options*)

Add constraints to a `vrna_fold_compound` 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. Depending on the type of the string the user has to pass `VRNA_CONSTRAINT_FILE` or `VRNA_CONSTRAINT_DB` in the option parameter, respectively. If none of these two options are passed, no action is performed, other than to guarantee that at least a hard constraints data structure of type `vrna_hc_t` with default values is present in 'vc'. Already existing hard constraints are not touched.

In case, a pseudo dot-bracket string is passed as the second argument, 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

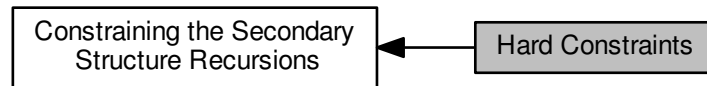
`vrna_hc_init()`, `vrna_sc_init()`, `vrna_hc_add_up()`, `vrna_hc_add_bp()`, `vrna_sc_add_up()`, `vrna_sc_add_bp()`, `vrna_sc_SHAPE_add_deigan()`, `vrna_sc_SHAPE_add_zarringhalam()`, `vrna_hc_free()`, `vrna_sc_free()`, `VRNA_CONSTRAINT_FILE`, `VRNA_CONSTRAINT_DB`, `VRNA_CONSTRAINT_DB_PIPE`, `VRNA_CONSTRAINT_DB_DOT`, `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`

Parameters

<i>vc</i>	The fold compound
<i>constraint</i>	A string with either the filename of the constraint definitions or a pseudo dot-bracket notation of the hard constraint. May be NULL.
<i>options</i>	The option flags

11.37 Hard Constraints

Collaboration diagram for Hard Constraints:



Data Structures

- struct [vrna_hc_t](#)
The hard constraints data structure.

Macros

- #define [VRNA_CONSTRAINT_MULTILINE](#) 32U
constraint may span over several lines
- #define [VRNA_CONSTRAINT_NO_HEADER](#) 64U
do not print the header information line
- #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.

Functions

- void [vrna_hc_init](#) ([vrna_fold_compound](#) *vc)
Initialize/Reset hard constraints to default values.
- void [vrna_hc_add_up](#) ([vrna_fold_compound](#) *vc, int i, char option)
Make a certain nucleotide unpaired.
- void [vrna_hc_add_bp](#) ([vrna_fold_compound](#) *vc, int i, int j, char option)
Favorize/Enforce a certain base pair (i,j)
- void [vrna_hc_add_bp_nonspecific](#) ([vrna_fold_compound](#) *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.

11.37.1 Detailed Description

11.37.2 Function Documentation

11.37.2.1 void vrna_hc_init (vrna_fold_compound * vc)

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 before initialization.

See also

[vrna_hc_add_bp\(\)](#), [vrna_hc_add_bp_nonspecific\(\)](#), [vrna_hc_add_up\(\)](#)

Parameters

<i>vc</i>	The fold compound
-----------	-------------------

11.37.2.2 void vrna_hc_add_up (vrna_fold_compound * vc, int i, char option)

Make a certain nucleotide unpaired.

See also

[vrna_hc_add_bp\(\)](#), [vrna_hc_add_bp_nonspecific\(\)](#), [vrna_hc_init\(\)](#), [VRNA_CONSTRAINT_CONTEXT_E↵XT_LOOP](#), [VRNA_CONSTRAINT_CONTEXT_HP_LOOP](#), [VRNA_CONSTRAINT_CONTEXT_INT_LOOP](#), [VRNA_CONSTRAINT_CONTEXT_MB_LOOP](#), [VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS](#)

Parameters

<i>vc</i>	The vrna_fold_compound the hard constraints are associated with
<i>i</i>	The position that needs to stay unpaired (1-based)
<i>option</i>	The options flag indicating how/where to store the hard constraints

11.37.2.3 void vrna_hc_add_bp (vrna_fold_compound * vc, int i, int j, char option)

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_E↵XT_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

<i>vc</i>	The vrna_fold_compound the hard constraints are associated with
<i>i</i>	The 5' located nucleotide position of the base pair (1-based)
<i>j</i>	The 3' located nucleotide position of the base pair (1-based)

<i>option</i>	The options flag indicating how/where to store the hard constraints
---------------	---

11.37.2.4 `void vrna_hc_add_bp_nonspecific (vrna_fold_compound * vc, int i, int d, char option)`

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

<i>vc</i>	The vrna_fold_compound the hard constraints are associated with
<i>i</i>	The position that needs to stay unpaired (1-based)
<i>d</i>	The direction of base pairing ($d < 0$: pairs upstream, $d > 0$: pairs downstream, $d == 0$: no direction)
<i>option</i>	The options flag indicating in which loop type context the pairs may appear

11.37.2.5 `void vrna_hc_free (vrna_hc_t * hc)`

Free the memory allocated by a [vrna_hc_t](#) data structure.

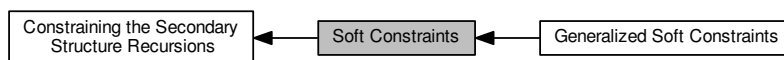
Use this function to free all memory that was allocated for a data structure of type [vrna_hc_t](#).

See also

[get_hard_constraints\(\)](#), [vrna_hc_t](#)

11.38 Soft Constraints

Collaboration diagram for Soft Constraints:



Modules

- [Generalized Soft Constraints](#)

Data Structures

- struct [vrna_sc_t](#)
The soft constraints data structure.

Macros

- `#define VRNA_CONSTRAINT_SOFT_MFE 8192U`
Soft constraints flag, apply constraints for MFE calculations.
- `#define VRNA_CONSTRAINT_SOFT_PF 16384U`
Soft constraints flag, apply constraints for partition function calculations.
- `#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_init` (`vrna_fold_compound` *vc)
Initialize an empty soft constraints data structure within a `vrna_fold_compound`.
- void `vrna_sc_add_bp` (`vrna_fold_compound` *vc, const double **constraints, unsigned int options)
Add soft constraints for paired nucleotides.
- void `vrna_sc_add_up` (`vrna_fold_compound` *vc, const double *constraints, unsigned int options)
Add soft constraints for unpaired nucleotides.
- void `vrna_sc_remove` (`vrna_fold_compound` *vc)
Remove soft constraints from `vrna_fold_compound`.
- void `vrna_sc_free` (`vrna_sc_t` *sc)
Free memory occupied by a `vrna_sc_t` data structure.
- int `vrna_sc_SHAPE_add_deigan` (`vrna_fold_compound` *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_SHAPE_add_deigan_aliases` (`vrna_fold_compound` *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_SHAPE_add_zarrinhalam` (`vrna_fold_compound` *vc, const double *reactivities, double b, double default_value, const char *shape_conversion, unsigned int options)
Add SHAPE reactivity data as soft constraints (Zarrinhalam 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.
- void `vrna_sc_minimize_perturbation` (`vrna_fold_compound` *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 discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

11.38.1 Detailed Description

11.38.2 Macro Definition Documentation

11.38.2.1 #define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

Use the sum of squared aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{\epsilon_i^2}{\tau^2} + \sum_{i=1}^n \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow \min$$

11.38.2.2 #define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

Use the sum of absolute aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{|\epsilon_i|}{\tau^2} + \sum_{i=1}^n \frac{|p_i(\vec{\epsilon}) - q_i|}{\sigma^2} \rightarrow \min$$

11.38.2.3 #define VRNA_MINIMIZER_CONJUGATE_FR 1

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

11.38.2.4 #define VRNA_MINIMIZER_CONJUGATE_PR 2

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

11.38.2.5 #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.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

11.38.2.6 #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.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

11.38.2.7 #define VRNA_MINIMIZER_STEEPEST_DESCENT 5

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

11.38.3 Typedef Documentation**11.38.3.1 typedef void(* progress_callback) (int iteration, double score, double *epsilon)**

Callback for following the progress of the minimization process.

Parameters

<i>iteration</i>	The number of the current iteration
<i>score</i>	The score of the objective function
<i>epsilon</i>	The perturbation vector yielding the reported score

11.38.4 Function Documentation**11.38.4.1 void vrna_sc_init (vrna_fold_compound * vc)**

Initialize an empty soft constraints data structure within a [vrna_fold_compound](#).

This function adds a proper soft constraints data structure to the [vrna_fold_compound](#) data structure. If soft constraints already exist within the fold compound, they are removed.

Note

Accepts [vrna_fold_compound](#) of type [VRNA_VC_TYPE_SINGLE](#) and [VRNA_VC_TYPE_ALIGNMENT](#)

See also

[vrna_sc_add_bp\(\)](#), [vrna_sc_add_up\(\)](#), [vrna_sc_SHAPE_add_deigan\(\)](#), [vrna_sc_SHAPE_add_zarringham\(\)](#), [vrna_sc_remove\(\)](#), [vrna_sc_add_f\(\)](#), [vrna_sc_add_exp_f\(\)](#), [vrna_sc_add_pre\(\)](#), [vrna_sc_add_post\(\)](#)

Parameters

<i>vc</i>	The vrna_fold_compound where an empty soft constraint feature is to be added to
-----------	---

11.38.4.2 `void vrna_sc_add_bp (vrna_fold_compound * vc, const double ** constraints, unsigned int options)`

Add soft constraints for paired nucleotides.

Parameters

<i>vc</i>	The vrna_fold_compound the soft constraints are associated with
<i>constraints</i>	A two-dimensional array of pseudo free energies in <i>kcal/mol</i>
<i>options</i>	The options flag indicating how/where to store the soft constraints

11.38.4.3 `void vrna_sc_add_up (vrna_fold_compound * vc, const double * constraints, unsigned int options)`

Add soft constraints for unpaired nucleotides.

Parameters

<i>vc</i>	The vrna_fold_compound the soft constraints are associated with
<i>constraints</i>	A vector of pseudo free energies in <i>kcal/mol</i>
<i>options</i>	The options flag indicating how/where to store the soft constraints

11.38.4.4 `void vrna_sc_remove (vrna_fold_compound * vc)`

Remove soft constraints from [vrna_fold_compound](#).

Note

Accepts [vrna_fold_compound](#) of type [VRNA_VC_TYPE_SINGLE](#) and [VRNA_VC_TYPE_ALIGNMENT](#)

Parameters

<i>vc</i>	The vrna_fold_compound possibly containing soft constraints
-----------	---

11.38.4.5 `void vrna_sc_free (vrna_sc_t * sc)`

Free memory occupied by a [vrna_sc_t](#) data structure.

Parameters

<i>sc</i>	The data structure to free from memory
-----------	--

11.38.4.6 `int vrna_sc_SHAPE_add_deigan (vrna_fold_compound * vc, const double * reactivities, double m, double b, unsigned int options)`

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 [3].

[vrna_sc_remove\(\)](#), [vrna_sc_SHAPE_add_zarringhalam\(\)](#), [vrna_sc_minimize_pertubation\(\)](#)

Parameters

<i>vc</i>	The vrna_fold_compound the soft constraints are associated with
<i>reactivities</i>	A vector of normalized SHAPE reactivities
<i>m</i>	The slope of the conversion function
<i>b</i>	The intercept of the conversion function
<i>options</i>	The options flag indicating how/where to store the soft constraints

Returns

1 on successful extraction of the method, 0 on errors

11.38.4.7 `int vrna_sc_SHAPE_add_deigan_ali (vrna_fold_compound * 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)

Parameters

<i>vc</i>	The vrna_fold_compound the soft constraints are associated with
<i>shape_files</i>	A set of filenames that contain normalized SHAPE reactivity data
<i>shape_file_↔ association</i>	An array of integers that associate the files with sequences in the alignment
<i>m</i>	The slope of the conversion function
<i>b</i>	The intercept of the conversion function
<i>options</i>	The options flag indicating how/where to store the soft constraints

Returns

1 on successful extraction of the method, 0 on errors

11.38.4.8 `int vrna_sc_SHAPE_add_zarringhalam (vrna_fold_compound * 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)

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 [16]

[vrna_sc_remove\(\)](#), [vrna_sc_SHAPE_add_deigan\(\)](#), [vrna_sc_minimize_pertubation\(\)](#)

Parameters

<i>vc</i>	The vrna_fold_compound the soft constraints are associated with
<i>reactivities</i>	A vector of normalized SHAPE reactivities
<i>b</i>	The scaling factor β of the conversion function
<i>options</i>	The options flag indicating how/where to store the soft constraints

Returns

1 on successful extraction of the method, 0 on errors

11.38.4.9 `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.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the double array values.

See also

[vrna_read_SHAPE_file\(\)](#)

Parameters

<i>shape_↔ conversion</i>	String defining the method used for the conversion process
<i>values</i>	Pointer to an array of SHAPE reactivities
<i>length</i>	Length of the array of SHAPE reactivities
<i>default_value</i>	Result used for position with invalid/missing reactivity values

11.38.4.10 `void vrna_sc_minimize_perturbation (vrna_fold_compound * 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 discrepancies between predicted and observed pairing probabilities and the amount of necessary 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 discrepancies 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 probabilities. 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 refer to [15].

Parameters

<i>vc</i>	Pointer to a fold compound
<i>q_prob_unpaired</i>	Pointer to an array containing the probability to be unpaired for each nucleotide
<i>objective_↔ function</i>	The type of objective function to be used (VRNA_OBJECTIVE_FUNCTION_QUADRATIC / VRNA_OBJECTIVE_FUNCTION_LINEAR)
<i>sigma_squared</i>	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the null vector.
<i>tau_squared</i>	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the data provided in <i>q_prob_unpaired</i> .
<i>algorithm</i>	The minimization algorithm (VRNA_MINIMIZER_*)
<i>sample_size</i>	The number of sampled sequences used for estimating the pairing probabilities. A value ≤ 0 will lead to an exact evaluation.
<i>epsilon</i>	A pointer to an array used for storing the calculated vector of perturbation energies
<i>callback</i>	A pointer to a callback function used for reporting the current minimization progress

11.39 Generalized Soft Constraints

Collaboration diagram for Generalized Soft Constraints:



Macros

- `#define VRNA_DECOMP_PAIR_HP 1`
Generalized constraint folding flag indicating hairpin loop decomposition step.
- `#define VRNA_DECOMP_PAIR_IL 2`
Generalized constraint folding flag indicating interior loop decomposition step.
- `#define VRNA_SC_GEN_MFE (char)1`
A flag passed to the generalized soft constraints pre-, and post- functions to indicate Minimum Free Energy (MFE) processing.
- `#define VRNA_SC_GEN_PF (char)2`
A flag passed to the generalized soft constraints pre-, and post- functions to indicate Partition function (PF) processing.

Functions

- `void vrna_sc_add_f (vrna_fold_compound *vc, int(*f)(int, int, int, int, char, void *), void *data)`
Bind a function pointer for generalized soft constraint feature (MFE version)
- `void vrna_sc_add_bt (vrna_fold_compound *vc, PAIR *(*f)(int, int, int, int, char, void *))`
Bind a backtracking function pointer for generalized soft constraint feature.
- `void vrna_sc_add_exp_f (vrna_fold_compound *vc, FLT_OR_DBL(*exp_f)(int, int, int, int, char, void *), void *data)`
Bind a function pointer for generalized soft constraint feature (PF version)
- `void vrna_sc_add_pre (vrna_fold_compound *vc, void(*pre)(vrna_fold_compound *, char))`
Add a pre-processing function for the generalized soft constraints feature.
- `void vrna_sc_add_post (vrna_fold_compound *vc, void(*post)(vrna_fold_compound *, char))`
Add a post-processing function for the generalized soft constraints feature.

11.39.1 Detailed Description

11.39.2 Macro Definition Documentation

11.39.2.1 `#define VRNA_SC_GEN_MFE (char)1`

A flag passed to the generalized soft constraints pre-, and post- functions to indicate Minimum Free Energy (MFE) processing.

This flag is passed as second argument to the pre-, and post- processing funtions that are bound to the `vrna_sc_t` structure via `vrna_sc_add_pre()`, and `vrna_sc_add_post()`, respectively. Use it in your implementation of the pre-, and post-processing functions to determine the mode of action required for corresponding pre-, and post-processing of data available to the function.

Note

This flag will be passed by calls of `vrna_fold()`, `vrna_alifold()`, `vrna_cofold()`, and `vrna_subopt()`

11.39.2.2 #define VRNA_SC_GEN_PF (char)2

A flag passed to the generalized soft constraints pre-, and post- functions to indicate Partition function (PF) processing.

This flag is passed as second argument to the pre-, and post- processing functions that are bound to the `vrna_sc_t` structure via `vrna_sc_add_pre()`, and `vrna_sc_add_post()`, respectively. Use it in your implementation of the pre-, and post-processing functions to determine the mode of action required for corresponding pre-, and post-processing of data available to the function.

Note

This flag will be passed by calls of `vrna_pf_fold()`, `vrna_alifold()`, and `vrna_co_pf_fold()`.

11.39.3 Function Documentation

11.39.3.1 void vrna_sc_add_f (vrna_fold_compound * vc, int (*)(int, int, int, int, char, void *) f, void * data)

Bind a function pointer for generalized 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`. The function for evaluating the generalized soft constraint feature has to return a pseudo free energy \hat{E} in *dacal/mol*, where $1\text{dacal/mol} = 10\text{cal/mol}$.

Parameters

<code>vc</code>	The fold compound the generalized soft constraint function should be bound to
<code>f</code>	A pointer to the function that evaluates the generalized soft constraint feature
<code>data</code>	A pointer to the data structure that holds required data for function 'f'

11.39.3.2 void vrna_sc_add_bt (vrna_fold_compound * vc, PAIR (*)(int, int, int, int, char, void *) f)

Bind a backtracking function pointer for generalized 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`. The provided function should be used for backtracking purposes in loop regions that were altered via the generalized soft constraint feature. It has to return an array of `PAIR` data structures, where the last element in the list is indicated by a value of -1 in its `i` position.

Parameters

<code>vc</code>	The fold compound the generalized soft constraint function should be bound to
<code>f</code>	A pointer to the function that returns additional base pairs

11.39.3.3 void vrna_sc_add_exp_f (vrna_fold_compound * vc, FLT_OR_DBL (*)(int, int, int, int, char, void *) exp_f, void * data)

Bind a function pointer for generalized 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`. The function for evaluating the generalized 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*.

Parameters

<i>vc</i>	The fold compound the generalized soft constraint function should be bound to
<i>exp_f</i>	A pointer to the function that evaluates the generalized soft constraint feature
<i>data</i>	A pointer to the data structure that holds required data for function 'f'

11.39.3.4 `void vrna_sc_add_pre (vrna_fold_compound * vc, void(*)(vrna_fold_compound *, char) pre)`

Add a pre-processing function for the generalized soft constraints feature.

Note

The function pointer passed will be used by calls of [vrna_fold\(\)](#), [vrna_pf_fold\(\)](#), [vrna_ali_fold\(\)](#), [vrna_ali_pf_fold\(\)](#), [vrna_cofold\(\)](#), [vrna_co_pf_fold\(\)](#), and [vrna_subopt\(\)](#). Each call is provided with the current [vrna_fold_compound](#), and a flag to indicate whether general free energy evaluation ([VRNA_SC_GEN_MFE](#)), or partition function computations ([VRNA_SC_GEN_PF](#)) will take place next.

See also

[VRNA_SC_GEN_MFE](#), [VRNA_SC_GEN_PF](#), [vrna_sc_t](#), [vrna_fold_compound](#)

Parameters

<i>vc</i>	The fold compound the generalized soft constraint function should be bound to
<i>pre</i>	A pointer to the pre-processing function

11.39.3.5 `void vrna_sc_add_post (vrna_fold_compound * vc, void(*)(vrna_fold_compound *, char) post)`

Add a post-processing function for the generalized soft constraints feature.

Note

The function pointer passed will be used by calls of [vrna_fold\(\)](#), [vrna_pf_fold\(\)](#), [vrna_ali_fold\(\)](#), [vrna_ali_pf_fold\(\)](#), [vrna_cofold\(\)](#), [vrna_co_pf_fold\(\)](#), and [vrna_subopt\(\)](#). Each call is provided with the current [vrna_fold_compound](#), and a flag to indicate whether general free energy evaluation ([VRNA_SC_GEN_MFE](#)), or partition function computations ([VRNA_SC_GEN_PF](#)) has taken place before.

See also

[VRNA_SC_GEN_MFE](#), [VRNA_SC_GEN_PF](#), [vrna_sc_t](#), [vrna_fold_compound](#)

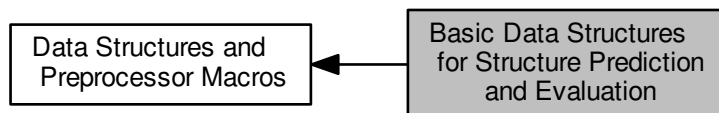
Parameters

<i>vc</i>	The fold compound the generalized soft constraint function should be bound to
<i>post</i>	A pointer to the post-processing function

11.40 Basic Data Structures for Structure Prediction and Evaluation

This module provides interfaces that deal with the most basic data structures used in structure predicting and energy evaluating function of the RNAlib.

Collaboration diagram for Basic Data Structures for Structure Prediction and Evaluation:



Files

- file [model.h](#)
The model details data structure and its corresponding modifiers.

Data Structures

- struct [vrna_mx_mfe_t](#)
Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).
- struct [vrna_mx_pf_t](#)
Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).
- struct [vrna_fold_compound](#)
The most basic data structure required by many functions throughout the RNAlib.
- struct [vrna_md_t](#)
The data structure that contains the complete model details used throughout the calculations.

Macros

- `#define VRNA_OPTION_MFE 1`
Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.
- `#define VRNA_OPTION_PF 2`
Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.
- `#define VRNA_OPTION_EVAL_ONLY 8`
Option flag to specify that neither MFE, nor PF DP matrices are required.
- `#define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0`
Default temperature for structure prediction and free energy evaluation in °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_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.*

Enumerations

- enum `vrna_mx_t` { `VRNA_MX_DEFAULT`, `VRNA_MX_LFOLD`, `VRNA_MX_2DFOLD` }
- An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.*
- enum `vrna_vc_t` { `VRNA_VC_TYPE_SINGLE`, `VRNA_VC_TYPE_ALIGNMENT` }
- An enumerator that is used to specify the type of a `vrna_fold_compound`.*

Functions

- `vrna_fold_compound * vrna_get_fold_compound` (const char *sequence, `vrna_md_t` *md_p, unsigned int options)
Retrieve a `vrna_fold_compound` data structure for single sequences and hybridizing sequences.
- `vrna_fold_compound * vrna_get_fold_compound.ali` (const char **sequences, `vrna_md_t` *md_p, unsigned int options)
Retrieve a `vrna_fold_compound` data structure for sequence alignments.
- void `vrna_params_update` (`vrna_fold_compound` *vc, `vrna_param_t` *par)
Update/Reset energy parameters data structure within a `vrna_fold_compound`.
- void `vrna_free_fold_compound` (`vrna_fold_compound` *vc)
Free memory occupied by a `vrna_fold_compound`.
- void `vrna_free_mfe_matrices` (`vrna_fold_compound` *vc)
Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.
- void `vrna_free_pf_matrices` (`vrna_fold_compound` *vc)
Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.
- void `vrna_md_set_default` (`vrna_md_t` *md)
Set default model details.
- void `vrna_md_update` (`vrna_md_t` *md)
Update the model details data structure.
- void `vrna_md_set_globals` (`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

11.40.1 Detailed Description

This module provides interfaces that deal with the most basic data structures used in structure predicting and energy evaluating function of the RNAlib.

Throughout the RNAlib, a data structure, the [vrna_fold_compound](#), 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](#) data structures.

11.40.2 Macro Definition Documentation

11.40.2.1 #define VRNA_OPTION_MFE 1

Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.

See also

[vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_alii\(\)](#), [VRNA_OPTION_EVAL_ONLY](#)

11.40.2.2 #define VRNA_OPTION_PF 2

Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.

See also

[vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_alii\(\)](#), [VRNA_OPTION_EVAL_ONLY](#)

11.40.2.3 #define VRNA_OPTION_EVAL_ONLY 8

Option flag to specify that neither MFE, nor PF DP matrices are required.

Use this flag in conjunction with [VRNA_OPTION_MFE](#), and [VRNA_OPTION_PF](#) to save memory for a [vrna_fold_compound](#) obtained from [vrna_get_fold_compound\(\)](#), or [vrna_get_fold_compound_alii\(\)](#) in cases where only energy evaluation but no structure prediction is required.

See also

[vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_alii\(\)](#), [vrna_eval_structure\(\)](#)

11.40.2.4 #define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0

Default temperature for structure prediction and free energy evaluation in °C

See also

[vrna_md_t.temperature](#), [vrna_md_set_default\(\)](#)

11.40.2.5 #define VRNA_MODEL_DEFAULT_PF_SCALE -1

Default scaling factor for partition function computations.

See also

[vrna_exp_param_t.pf_scale](#), [vrna_md_set_default\(\)](#)

11.40.2.6 #define VRNA_MODEL_DEFAULT_BETA_SCALE 1.

Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.

See also

[vrna_exp_param_t.alpha](#), [vrna_md_t.betaScale](#), [vrna_md_set_default\(\)](#)

11.40.2.7 #define VRNA_MODEL_DEFAULT_DANGLES 2

Default dangling end model.

See also

[vrna_md_t.dangles](#), [vrna_md_set_default\(\)](#)

11.40.2.8 #define VRNA_MODEL_DEFAULT_SPECIAL_HP 1

Default model behavior for lookup of special tri-, tetra-, and hexa-loops.

See also

[vrna_md_t.special_hp](#), [vrna_md_set_default\(\)](#)

11.40.2.9 #define VRNA_MODEL_DEFAULT_NO_LP 0

Default model behavior for so-called 'lonely pairs'.

See also

[vrna_md_t.noLP](#), [vrna_md_set_default\(\)](#)

11.40.2.10 #define VRNA_MODEL_DEFAULT_NO_GU 0

Default model behavior for G-U base pairs.

See also

[vrna_md_t.noGU](#), [vrna_md_set_default\(\)](#)

11.40.2.11 #define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0

Default model behavior for G-U base pairs closing a loop.

See also

[vrna_md_t.noGUclosure](#), [vrna_md_set_default\(\)](#)

11.40.2.12 #define VRNA_MODEL_DEFAULT_CIRC 0

Default model behavior to treat a molecule as a circular RNA (DNA)

See also

[vrna_md_t.circ](#), [vrna_md_set_default\(\)](#)

11.40.2.13 #define VRNA_MODEL_DEFAULT_GQUAD 0

Default model behavior regarding the treatment of G-Quadruplexes.

See also

[vrna_md_t.gquad](#), [vrna_md_set_default\(\)](#)

11.40.2.14 #define VRNA_MODEL_DEFAULT_UNIQ_ML 0

Default behavior of the model regarding unique multibranch loop decomposition.

See also

[vrna_md_t.uniq_ML](#), [vrna_md_set_default\(\)](#)

11.40.2.15 #define VRNA_MODEL_DEFAULT_ENERGY_SET 0

Default model behavior on which energy set to use.

See also

[vrna_md_t.energy_set](#), [vrna_md_set_default\(\)](#)

11.40.2.16 #define VRNA_MODEL_DEFAULT_BACKTRACK 1

Default model behavior with regards to backtracking of structures.

See also

[vrna_md_t.backtrack](#), [vrna_md_set_default\(\)](#)

11.40.2.17 #define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'

Default model behavior on what type of backtracking to perform.

See also

[vrna_md_t.backtrack_type](#), [vrna_md_set_default\(\)](#)

11.40.2.18 #define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1

Default model behavior with regards to computing base pair probabilities.

See also

[vrna_md_t.compute_bpp](#), [vrna_md_set_default\(\)](#)

11.40.2.19 #define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1

Default model behavior for the allowed maximum base pair span.

See also

[vrna_md_t.max_bp_span](#), [vrna_md_set_default\(\)](#)

11.40.2.20 #define VRNA_MODEL_DEFAULT_LOG_ML 0

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

See also

[vrna_md_t.logML](#), [vrna_md_set_default\(\)](#)

11.40.2.21 #define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0

Default model behavior for consensus structure energy evaluation.

See also

[vrna_md_t.oldAliEn](#), [vrna_md_set_default\(\)](#)

11.40.2.22 #define VRNA_MODEL_DEFAULT_ALI_RIBO 0

Default model behavior for consensus structure covariance contribution assessment.

See also

[vrna_md_t.ribo](#), [vrna_md_set_default\(\)](#)

11.40.2.23 #define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.

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

See also

[vrna_md_t.cv_fact](#), [vrna_md_set_default\(\)](#)

11.40.2.24 #define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

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

See also

[#vrna_md_t.nc_fact](#), [vrna_md_set_default\(\)](#)

11.40.3 Enumeration Type Documentation

11.40.3.1 enum `vrna_mx_t`

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_LFOLD` DP matrices suitable for local structure prediction.

See also

[Lfold\(\)](#), [pfl_fold\(\)](#)

`VRNA_MX_2DFOLD` DP matrices suitable for distance class partitioned structure prediction.

See also

[vrna_TwoD_fold\(\)](#), [vrna_TwoD_pf_fold\(\)](#)

11.40.3.2 enum `vrna_vc_t`

An enumerator that is used to specify the type of a [vrna_fold_compound](#).

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)

11.40.4 Function Documentation

11.40.4.1 `vrna_fold_compound* vrna_get_fold_compound (const char * sequence, vrna_md_t * md_p, unsigned int options)`

Retrieve a [vrna_fold_compound](#) data structure for single sequences and hybridizing sequences.

This function provides an easy interface to obtain a prefilled [vrna_fold_compound](#) by passing a single sequence, or two concatenated sequences as input. For the latter, sequences need to be separated by an '&' character like this:

```
char *sequence = "GGGG&CCCC";
```

The optional parameter '`md_p`' can be used to specify the model details for computations on the `#vrna_fold_compounds` content. The third parameter '`options`' is used to specify the DP matrix requirements and the corresponding set of energy parameters. Use the macros:

- [VRNA_OPTION_MFE](#)
- [VRNA_OPTION_PF](#)
- [VRNA_OPTION_EVAL_ONLY](#)

to specify the required type of computations that will be performed with the [vrna_fold_compound](#).

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_get_fold_compound_ali\(\)](#), [vrna_md_t](#), [VRNA_OPTION_MFE](#), [VRNA_OPTION_PF](#), [VRNA_OPTION_EVAL_ONLY](#)

Parameters

<i>sequence</i>	A single sequence, or two concatenated sequences separated by an '&' character
<i>md_p</i>	An optional set of model details
<i>options</i>	The options for DP matrices memory allocation

Returns

A prefilled [vrna_fold_compound](#) that can be readily used for computations

11.40.4.2 `vrna_fold_compound* vrna_get_fold_compound_ali (const char ** sequences, vrna_md_t * md_p, unsigned int options)`

Retrieve a [vrna_fold_compound](#) data structure for sequence alignments.

This function provides an easy interface to obtain a prefilled [vrna_fold_compound](#) by passing an alignment of sequences.

The optional parameter 'md_p' can be used to specify the model details for computations on the `#vrna_fold_compound` content. The third parameter 'options' is used to specify the DP matrix requirements and the corresponding set of energy parameters. Use the macros:

- [VRNA_OPTION_MFE](#)
- [VRNA_OPTION_PF](#)
- [VRNA_OPTION_EVAL_ONLY](#)

to specify the required type of computations that will be performed with the [vrna_fold_compound](#).

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_get_fold_compound\(\)](#), [vrna_md_t](#), [VRNA_OPTION_MFE](#), [VRNA_OPTION_PF](#), [VRNA_OPTION_EVAL_ONLY](#), [read_clustal\(\)](#)

Parameters

<i>sequences</i>	A sequence alignment including 'gap' characters
<i>md_p</i>	An optional set of model details

<i>options</i>	The options for DP matrices memory allocation
----------------	---

Returns

A prefilled [vrna_fold_compound](#) that can be readily used for computations

11.40.4.3 `void vrna_params_update (vrna_fold_compound * vc, vrna_param_t * par)`

Update/Reset energy parameters data structure within a [vrna_fold_compound](#).

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.

energy_parameters

Parameters

<i>vc</i>	The vrna_fold_compound that is about to receive updated energy parameters
<i>par</i>	The energy parameters used to substitute those within vc (Maybe NULL)

11.40.4.4 `void vrna_free_fold_compound (vrna_fold_compound * vc)`

Free memory occupied by a [vrna_fold_compound](#).

See also

[vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_aliases\(\)](#), [vrna_free_mfe_matrices\(\)](#), [vrna_free_pf_matrices\(\)](#)

Parameters

<i>vc</i>	The vrna_fold_compound that is to be erased from memory
-----------	---

11.40.4.5 `void vrna_free_mfe_matrices (vrna_fold_compound * vc)`

Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.

See also

[vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_aliases\(\)](#), [vrna_free_fold_compound\(\)](#), [vrna_free_pf_matrices\(\)](#)

Parameters

<i>vc</i>	The vrna_fold_compound storing the MFE DP matrices that are to be erased from memory
-----------	--

11.40.4.6 `void vrna_free_pf_matrices (vrna_fold_compound * vc)`

Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

See also

[vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_aliases\(\)](#), [vrna_free_fold_compound\(\)](#), [vrna_free_mfe_matrices\(\)](#)

Parameters

<i>vc</i>	The vrna_fold_compound storing the PF DP matrices that are to be erased from memory
-----------	---

11.40.4.7 void vrna_md_set_default (vrna_md_t * md)

Set default model details.

Use this function if you wish to initialize a [vrna_md_t](#) data structure with its default values

Parameters

<i>md</i>	A pointer to the data structure that is about to be initialized
-----------	---

11.40.4.8 void vrna_md_update (vrna_md_t * md)

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\(\)](#)

11.40.4.9 void vrna_md_set_globals (vrna_md_t * md)

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

<i>md</i>	A pointer to the data structure that is about to be initialized
-----------	---

11.40.5 Variable Documentation

11.40.5.1 double temperature

Rescale energy parameters to a temperature in degC.

Default is 37C. You have to call the [update_..._params\(\)](#) functions after changing this parameter.

11.40.5.2 double pf_scale

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.

11.40.5.3 int dangles

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

11.40.5.4 int tetra_loop

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

default is 1.

11.40.5.5 int noLonelyPairs

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.

11.40.5.6 int canonicalBOnly

Do not use this variable, it will eventually be removed in one of the next versions

11.40.5.7 int energy_set

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.

11.40.5.8 int do_backtrack

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.

11.40.5.9 char backtrack_type

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.

11.40.5.10 char* nonstandards

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.

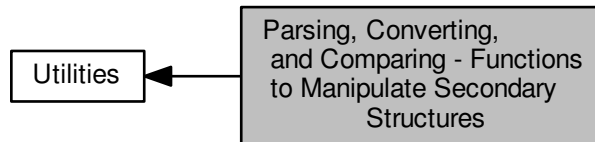
11.40.5.11 int max_bp_span

Maximum allowed base pair span.

A value of -1 indicates no restriction for distant base pairs.

11.41 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_helix](#)

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_pt_get](#) (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_pt_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_get_loop_index](#) (const short *pt)
- Get a loop index representation of a structure.*

 - char * [vrna_pt_to_db](#) (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_get_from_pr](#) (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.*

 - void [vrna_parenthesis_structure](#) (char *structure, [bondT](#) *bp, unsigned int length)
- Create a dot-bracket/parenthesis structure from backtracking stack.*

 - void [vrna_parenthesis_zuker](#) (char *structure, [bondT](#) *bp, unsigned int length)
- Create a dot-bracket/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.*

 - [plist](#) * [vrna_pl_get](#) (const char *struc, float pr)
- Create a [plist](#) from a dot-bracket string.*

 - char * [vrna_pl_to_db](#) ([plist](#) *pairs, unsigned int n)
- Convert a list of base pairs into dot-bracket notation.*

 - void [assign_plist_from_db](#) ([plist](#) **pl, const char *struc, float pr)
- Create a [plist](#) 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, [bondT](#) *bp, int length)

Create a dot-bracket/parenthesis structure from backtracking stack.
- void [parenthesis_zuker](#) (char *structure, [bondT](#) *bp, int length)

Create a dot-bracket/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.

11.41.1 Detailed Description

11.41.2 Function Documentation

11.41.2.1 char* b2HIT (const char * structure)

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

Parameters

<i>structure</i>	
------------------	--

Returns

11.41.2.2 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.

Parameters

<i>structure</i>	
------------------	--

Returns

11.41.2.3 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.

Parameters

<i>structure</i>	
------------------	--

Returns

11.41.2.4 char* add_root (const char * *structure*)

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

Parameters

<i>structure</i>	
------------------	--

Returns

11.41.2.5 char* expand_Shapiro (const char * *coarse*)

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

Parameters

<i>coarse</i>	
---------------	--

Returns

11.41.2.6 char* expand_Full (const char * *structure*)

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

Parameters

<i>structure</i>	
------------------	--

Returns

11.41.2.7 `char* unexpand_Full (const char * full)`

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

Parameters

<i>ffull</i>	
--------------	--

Returns

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

Strip weights from any weighted tree.

Parameters

<i>wcoarse</i>	
----------------	--

Returns

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

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

<i>align</i>	
--------------	--

11.41.2.10 `void parse_structure (const char * structure)`

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

<i>structure</i>	
------------------	--

Returns

11.41.2.11 `char* vrna_db_pack (const char * struc)`

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

<i>struc</i>	The secondary structure in dot-bracket notation
--------------	---

Returns

The binary encoded structure

11.41.2.12 `char* vrna_db_unpack (const char * packed)`

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

<i>packed</i>	The binary encoded packed secondary structure
---------------	---

Returns

The unpacked secondary structure in dot-bracket notation

11.41.2.13 `short* vrna_pt_get (const char * structure)`

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

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

Returns

A pointer to the created pair_table

11.41.2.14 `short* vrna_pt_pk_get (const char * structure)`

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_pt_get\(\)](#) this function also recognizes the base pairs denoted by '[' and ']' brackets.

Parameters

<i>structure</i>	The secondary structure in (extended) dot-bracket notation
------------------	--

Returns

A pointer to the created pair_table

11.41.2.15 `short* vrna_pt_copy (const short * pt)`

Get an exact copy of a pair table.

Parameters

<i>pt</i>	The pair table to be copied
-----------	-----------------------------

Returns

A pointer to the copy of 'pt'

11.41.2.16 `short* vrna_pt_snoop_get (const char * structure)`

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.

11.41.2.17 `char* vrna_pt_to_db (short * pt)`

Convert a pair table into dot-parenthesis notation.

Parameters

<i>pt</i>	The pair table to be copied
-----------	-----------------------------

Returns

A char pointer to the dot-bracket string

11.41.2.18 `int vrna_bp_distance (const char * str1, const char * str2)`

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

<i>str1</i>	First structure in dot-bracket notation
<i>str2</i>	Second structure in dot-bracket notation

Returns

The base pair distance between str1 and str2

11.41.2.19 `unsigned int* vrna_refBPcnt_matrix (const short * reference_pt, unsigned int turn)`

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!!!

11.41.2.20 `unsigned int* vrna_refBPdist_matrix (const short * pt1, const short * pt2, unsigned int turn)`

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!!!

11.41.2.21 `void vrna_parenthesis_zuker (char * structure, bondT * bp, unsigned int length)`

Create a dot-bracket/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.

Note

This function is threadsafe

11.41.2.22 `plist* vrna_pl_get (const char * struc, float pr)`

Create a [plist](#) 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

<i>struc</i>	The secondary structure in dot-bracket notation
<i>pr</i>	The probability for each base pair used in the plist

Returns

The plist array

11.41.2.23 `char* vrna_pl_to_db (plist * pairs, unsigned int n)`

Convert a list of base pairs into dot-bracket notation.

See also

[vrna_pl_get\(\)](#)

Parameters

<i>pairs</i>	A plist containing the pairs to be included in the dot-bracket string
<i>n</i>	The length of the structure (number of nucleotides)

Returns

The dot-bracket string containing the provided base pairs

11.41.2.24 `void assign_plist_from_db (plist ** pl, const char * struc, float pr)`

Create a [plist](#) 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_pl_get\(\)](#) instead

Parameters

<i>pl</i>	A pointer to the plist that is to be created
<i>struc</i>	The secondary structure in dot-bracket notation
<i>pr</i>	The probability for each base pair

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

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

<i>struc</i>	The secondary structure in dot-bracket notation
--------------	---

Returns

The binary encoded structure

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

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

<i>packed</i>	The binary encoded packed secondary structure
---------------	---

Returns

The unpacked secondary structure in dot-bracket notation

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

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_pt_get()` instead

Parameters

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

Returns

A pointer to the created `pair_table`

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

Get an exact copy of a pair table.

Deprecated Use `vrna_pt_copy()` instead

Parameters

<i>pt</i>	The pair table to be copied
-----------	-----------------------------

Returns

A pointer to the copy of 'pt'

11.41.2.29 `short* alimake_pair_table (const char * structure)`

Pair table for snoop align

Deprecated Use `vrna_pt_ali_get()` instead!

11.41.2.30 `short* make_pair_table_snoop (const char * 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. The special pseudoknotted H/ACA-mRNA structure is taken into account.

Deprecated Use `vrna_pt_snoop_get()` instead!

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

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

<i>str1</i>	First structure in dot-bracket notation
<i>str2</i>	Second structure in dot-bracket notation

Returns

The base pair distance between str1 and str2

11.41.2.32 `unsigned int* make_referenceBP_array (short * reference_pt, unsigned int turn)`

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

11.41.2.33 `unsigned int* compute_BPdifferences (short * pt1, short * pt2, unsigned int turn)`

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

11.41.2.34 `void parenthesis_structure (char * structure, bondT * bp, int length)`

Create a dot-bracket/parenthesis structure from backtracking stack.

Deprecated use `vrna_parenthesis_structure()` instead

Note

This function is threadsafe

11.41.2.35 `void parenthesis_zuker (char * structure, bondT * bp, int length)`

Create a dot-bracket/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in `cofold.c`.

Deprecated use `vrna_parenthesis_zuker` instead

Note

This function is threadsafe

11.41.2.36 `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.

Deprecated Use `vrna_db_get_from_pr()` instead!

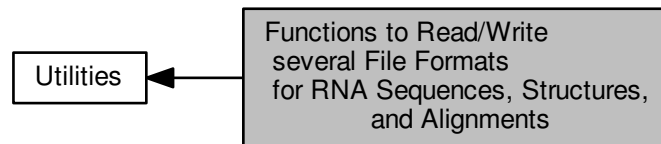
11.41.2.37 `char bppm_symbol (const float * x)`

Get a pseudo dot bracket notation for a given probability information.

Deprecated Use `vrna_bpp_symbol()` instead!

11.42 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.

Functions

- void [vrna_structure_print_hx](#) (const char *seq, const char *db, float energy, FILE *file)
Print a secondary structure as helix list.
- void [vrna_structure_print_ct](#) (const char *seq, const char *db, float energy, const char *identifier, FILE *file)
Print a secondary structure as connect table.
- void [vrna_structure_print_bpseq](#) (const char *seq, const char *db, FILE *file)
Print a secondary structure in bpseq format.
- unsigned int [vrna_read_fasta_record](#) (char **header, char **sequence, char ***rest, FILE *file, unsigned int options)
Get a (fasta) data set from a file or stdin.
- void [vrna_extract_record_rest_constraint](#) (char **cstruc, const char **lines, unsigned int option)
Extract a hard constraint encoded as pseudo dot-bracket string.
- int [vrna_read_SHAPE_file](#) (const char *file_name, int length, double default_value, char *sequence, double *values)
Read data from a given SHAPE reactivity input file.
- [plist](#) * [vrna_read_constraints_file](#) (const char *filename, unsigned int length, unsigned int options)
Read constraints from an input file.
- 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.

11.42.1 Detailed Description

11.42.2 Function Documentation

11.42.2.1 void vrna_structure_print_hx (const char * *seq*, const char * *db*, float *energy*, FILE * *file*)

Print a secondary structure as helix list.

Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

11.42.2.2 `void vrna_structure_print_ct (const char * seq, const char * db, float energy, const char * identifier, FILE * file)`

Print a secondary structure as connect table.

Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>energy</i>	The free energy of the structure
<i>identifier</i>	An optional identifier for the sequence
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

11.42.2.3 `void vrna_structure_print_bpseq (const char * seq, const char * db, FILE * file)`

Print a secondary structure in bpseq format.

Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL))

11.42.2.4 `unsigned int vrna_read_fasta_record (char ** header, char ** sequence, char *** rest, FILE * file, unsigned int options)`

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_read_fasta_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_read_fasta_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_read_fasta_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

<i>header</i>	A pointer which will be set such that it points to the header of the record
<i>sequence</i>	A pointer which will be set such that it points to the sequence of the record
<i>rest</i>	A pointer which will be set such that it points to an array of lines which also belong to the record
<i>file</i>	A file handle to read from (if NULL, this function reads from stdin)
<i>options</i>	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

11.42.2.5 void vrna_extract_record_rest_constraint (char ** cstruc, const char ** lines, unsigned int option)

Extract a hard constraint encoded as pseudo dot-bracket string.

Precondition

The argument 'lines' has to be a 2-dimensional character array as obtained by [vrna_read_fasta_record\(\)](#)

See also

[vrna_read_fasta_record\(\)](#), [VRNA_CONSTRAINT_DB_PIPE](#), [VRNA_CONSTRAINT_DB_DOT](#), [VRNA_CONSTRAINT_DB_X](#), [VRNA_CONSTRAINT_DB_ANG_BRACK](#), [VRNA_CONSTRAINT_DB_RND_BRACK](#)

Parameters

<i>cstruc</i>	A pointer to a character array that is used as pseudo dot-bracket output
<i>lines</i>	A 2-dimensional character array with the extension lines from the FASTA input
<i>option</i>	The option flags that define the behavior and recognition pattern of this function

11.42.2.6 `int vrna_read_SHAPE_file (const char * file_name, int length, double default_value, char * sequence, double * values)`

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

<i>file_name</i>	Path to the constraints file
<i>length</i>	Length of the sequence (file entries exceeding this limit will cause an error)
<i>default_value</i>	Value for missing indices
<i>sequence</i>	Pointer to an array used for storing the sequence obtained from the SHAPE reactivity file
<i>values</i>	Pointer to an array used for storing the values obtained from the SHAPE reactivity file

11.42.2.7 `plist* vrna_read_constraints_file (const char * filename, unsigned int length, unsigned int options)`

Read constraints from an input file.

This function reads constraint definitions from a file and converts them into an array of [plist](#) 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

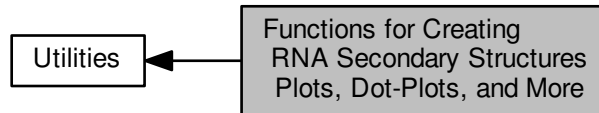
11.42.2.8 `unsigned int read_record (char ** header, char ** sequence, char *** rest, unsigned int options)`

Get a data record from stdin.

Deprecated This function is deprecated! Use [vrna_read_fasta_record\(\)](#) as a replacment.

11.43 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_layouts.h](#)
Secondary structure plot layout algorithms.
- file [PS_dot.h](#)
Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Data Structures

- struct [COORDINATE](#)
this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

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
- 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 [PS_dot_plot_list](#) (char *seq, char *filename, [plist](#) *pl, [plist](#) *mf, char *comment)
Produce a postscript dot-plot from two pair lists.
- int [aliPS_color_aln](#) (const char *structure, const char *filename, const char *seqs[], const char *names[])
- 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.

11.43.1 Detailed Description

11.43.2 Macro Definition Documentation

11.43.2.1 #define VRNA_PLOT_TYPE_SIMPLE 0

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](#), [PS_rna_plot_a\(\)](#), [PS_rna_plot\(\)](#), [svg_rna_plot\(\)](#), [gmlRNA\(\)](#), [ssv_rna_plot\(\)](#), [xrna_plot\(\)](#)

11.43.2.2 #define VRNA_PLOT_TYPE_NAVIEW 1

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](#), [PS_rna_plot_a\(\)](#), [PS_rna_plot\(\)](#), [svg_rna_plot\(\)](#), [gmlRNA\(\)](#), [ssv_rna_plot\(\)](#), [xrna_plot\(\)](#)

11.43.2.3 #define VRNA_PLOT_TYPE_CIRCULAR 2

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](#), [PS_rna_plot_a\(\)](#), [PS_rna_plot\(\)](#), [svg_rna_plot\(\)](#), [gmlRNA\(\)](#), [ssv_rna_plot\(\)](#), [xrna_plot\(\)](#)

11.43.3 Function Documentation

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

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

See also

[make_pair_table\(\)](#), [rna_plot_type](#), [simple_circplot_coordinates\(\)](#), [naview_xy_coordinates\(\)](#), [PS_rna_plot_a\(\)](#), [PS_rna_plot](#), [svg_rna_plot\(\)](#)

Parameters

<i>pair_table</i>	The pair table of the secondary structure
<i>X</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>Y</i>	a pointer to an array with enough allocated space to hold the y coordinates

Returns

length of sequence on success, 0 otherwise

11.43.3.2 `int simple_circplot_coordinates (short * pair_table, float * x, float * y)`

Calculate nucleotide coordinates for *Circular Plot*

This function calculates the coordinates of nucleotides mapped in equal distances 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. $P_x^t[i] = X[i] * rs$ and $P_y^t[i] = Y[i] * rs$.

See also

[make_pair_table\(\)](#), [rna_plot_type](#), [simple_xy_coordinates\(\)](#), [naview_xy_coordinates\(\)](#), [PS_rna_plot_a\(\)](#), [P↔S_rna_plot](#), [svg_rna_plot\(\)](#)

Parameters

<i>pair_table</i>	The pair table of the secondary structure
<i>x</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>y</i>	a pointer to an array with enough allocated space to hold the y coordinates

Returns

length of sequence on success, 0 otherwise

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

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

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>file</i>	The filename of the postscript output

Returns

1 on success, 0 otherwise

11.43.3.4 `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'.

Same as [PS_rna_plot\(\)](#) 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

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>file</i>	The filename of the postscript output
<i>pre</i>	PostScript code to appear before the secondary structure plot
<i>post</i>	PostScript code to appear after the secondary structure plot

Returns

1 on success, 0 otherwise

11.43.3.5 `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.

If 'option' is an uppercase letter the RNA sequence is used to label nodes, if 'option' equals 'X' or 'x' the resulting file will contain coordinates for an initial layout of the graph.

Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the gml output
<i>option</i>	The option flag

Returns

1 on success, 0 otherwise

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

Produce a secondary structure graph in SStructView format.

Write coord file for SStructView

Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the ssv output

Returns

1 on success, 0 otherwise

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

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

Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the svg output

Returns

1 on success, 0 otherwise

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

Produce a secondary structure plot for further editing in XRNA.

Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the xrna output

Returns

1 on success, 0 otherwise

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

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

<i>seq</i>	The RNA sequence
<i>filename</i>	A filename for the postscript output
<i>pl</i>	The base pair probability pairlist
<i>mf</i>	The mfe secondary structure pairlist
<i>comment</i>	A comment

Returns

1 if postscript was successfully written, 0 otherwise

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

PS_color_aln for duplexes

11.43.3.11 `int PS_dot_plot (char * string, char * file)`

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!

11.43.4 Variable Documentation

11.43.4.1 `int rna_plot_type`

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 Brucoleri & Heinrich (1988).

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 12

Data Structure Documentation

12.1 bondT Struct Reference

Base pair.

12.1.1 Detailed Description

Base pair.

The documentation for this struct was generated from the following file:

- /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.2 bondTEn Struct Reference

Base pair with associated energy.

12.2.1 Detailed Description

Base pair with associated energy.

The documentation for this struct was generated from the following file:

- /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.3 cofoldF Struct Reference

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

The documentation for this struct was generated from the following file:

- /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.4 ConcEnt Struct Reference

Data Fields

- double [A0](#)
start concentration A
- double [B0](#)
start concentration B
- double [ABc](#)
End concentration AB.

The documentation for this struct was generated from the following file:

- /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.5 constrain Struct Reference

constraints for cofolding

12.5.1 Detailed Description

constraints for cofolding

The documentation for this struct was generated from the following file:

- /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.6 COORDINATE Struct Reference

this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

12.6.1 Detailed Description

this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

The documentation for this struct was generated from the following file:

- /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/plot_layouts.h

12.7 cpair Struct Reference

this datastructure is used as input parameter in functions of PS_dot.c

12.7.1 Detailed Description

this datastructure is used as input parameter in functions of PS_dot.c

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.8 duplexT Struct Reference

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.9 dupVar Struct Reference

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.10 folden Struct Reference

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.11 interact Struct Reference

Data Fields

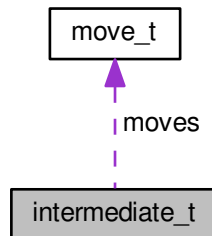
- double * [Pi](#)
probabilities of interaction
- double * [Gi](#)
free energies of interaction
- double [Gikjl](#)
full free energy for interaction between $[k,i]$ $k < i$ in longer seq and $[j,l]$ $j < l$ in shorter seq
- double [Gikjl_wo](#)
Gikjl without contributions for prob_unpaired.
- int [i](#)
 $k < i$ 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

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.12 intermediate_t Struct Reference

Collaboration diagram for intermediate_t:



Data Fields

- short * [pt](#)
pair table
- int [Sen](#)
saddle energy so far
- int [curr_en](#)
current energy
- [move_t](#) * [moves](#)
remaining moves to target

The documentation for this struct was generated from the following file:

- [/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.13 INTERVAL Struct Reference

Sequence interval stack element used in subopt.c.

12.13.1 Detailed Description

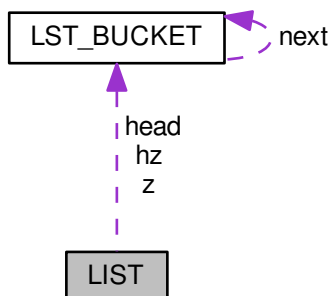
Sequence interval stack element used in subopt.c.

The documentation for this struct was generated from the following file:

- [/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.14 LIST Struct Reference

Collaboration diagram for LIST:



The documentation for this struct was generated from the following file:

- `/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/list.h`

12.15 LST_BUCKET Struct Reference

Collaboration diagram for LST_BUCKET:



The documentation for this struct was generated from the following file:

- `/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/list.h`

12.16 move_t Struct Reference

The documentation for this struct was generated from the following file:

- `/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h`

12.17 PAIR Struct Reference

Base pair data structure used in subopt.c.

12.17.1 Detailed Description

Base pair data structure used in subopt.c.

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.18 pair_info Struct Reference

A base pair info structure.

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

12.18.1 Detailed Description

A base pair info structure.

For each base pair (i,j) with i,j in [0, n-1] the structure lists:

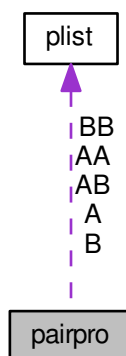
- its probability 'p'
- an entropy-like measure for its well-definedness 'ent'
- the frequency of each type of pair in 'bp[]'
 - 'bp[0]' contains the number of non-compatible sequences
 - 'bp[1]' the number of CG pairs, etc.

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.19 pairpro Struct Reference

Collaboration diagram for pairpro:



The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.20 path_t Struct Reference

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.21 plist Struct Reference

this datastructure is used as input parameter in functions of [PS_dot.h](#) and others

12.21.1 Detailed Description

this datastructure is used as input parameter in functions of [PS_dot.h](#) and others

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.22 Postorder_list Struct Reference

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/dist_vars.h

12.23 pu_contrib Struct Reference

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

12.23.1 Detailed Description

contributions to p_u

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.24 pu_out Struct Reference

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*

12.24.1 Detailed Description

Collection of all free_energy of beeing unpaired values for output.

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.25 sect Struct Reference

Stack of partial structures for backtracking.

12.25.1 Detailed Description

Stack of partial structures for backtracking.

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.26 snoopT Struct Reference

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.27 SOLUTION 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.

12.27.1 Detailed Description

Solution element from subopt.c.

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.28 struct_en Struct Reference

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/move_set.h](#)

12.29 svm_model Struct Reference

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/svm_utils.h](#)

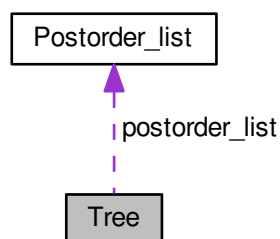
12.30 swString Struct Reference

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/dist_vars.h

12.31 Tree Struct Reference

Collaboration diagram for Tree:



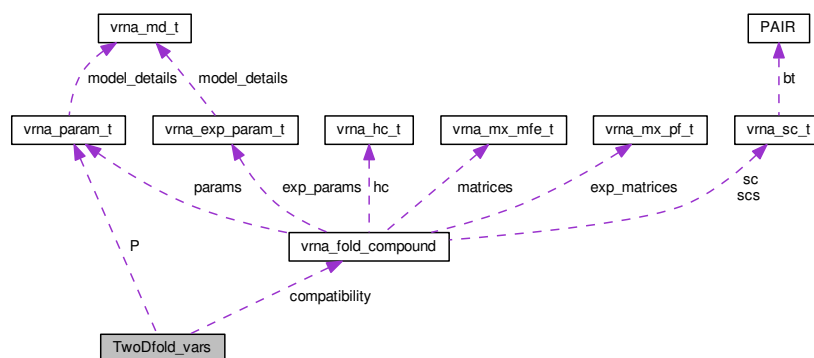
The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/dist_vars.h

12.32 TwoDfold_vars Struct Reference

Variables compound for 2Dfold MFE folding.

Collaboration diagram for TwoDfold_vars:



Data Fields

- struct [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 distance 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].

12.32.1 Detailed Description

Variables compound for 2Dfold MFE folding.

Deprecated This data structure will be removed from the library soon! Use [vrna_fold_compound](#) and the corresponding functions [vrna_get_fold_compound_2D\(\)](#), [vrna_TwoD_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

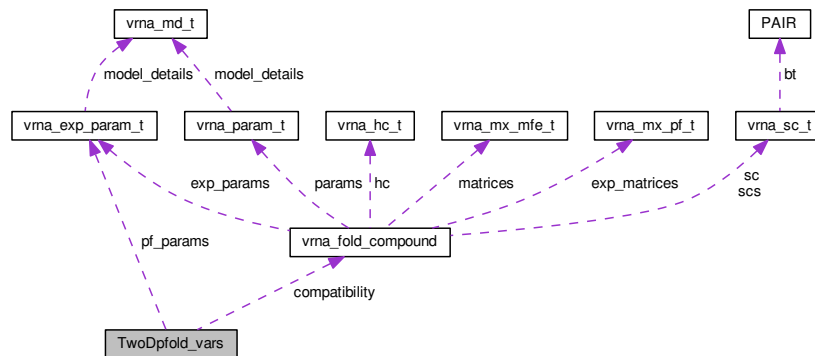
The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dfold.h](#)

12.33 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 distance 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.

12.33.1 Detailed Description

Variables compound for 2Dfold partition function folding.

Deprecated This data structure will be removed from the library soon! Use [vrna_fold_compound](#) and the corresponding functions [vrna_get_fold_compound_2D\(\)](#), [vrna_TwoD_pf_fold\(\)](#), and [vrna_free_fold_compound\(\)](#) instead!

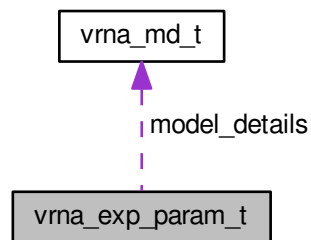
The documentation for this struct was generated from the following file:

- </home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dpfold.h>

12.34 vrna_exp_param_t Struct Reference

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

Collaboration diagram for vrna_exp_param_t:



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.

12.34.1 Detailed Description

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

12.34.2 Field Documentation

12.34.2.1 `int vrna_exp_param_t::id`

An identifier for the data structure.

Deprecated This attribute will be removed in version 3

12.34.2.2 double vrna_exp_param_t::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 $e^{-E/(\alpha \cdot K \cdot T)}$

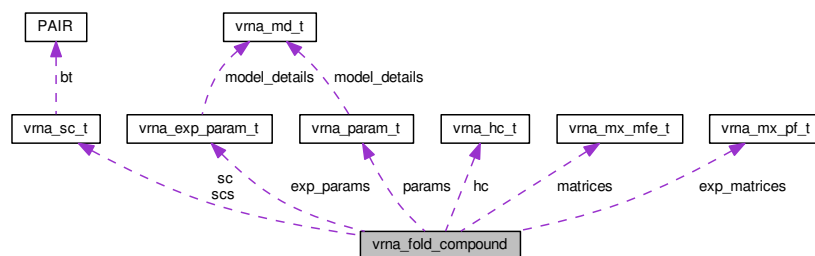
The documentation for this struct was generated from the following file:

- </home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/params.h>

12.35 vrna_fold_compound Struct Reference

The most basic data structure required by many functions throughout the RNAlib.

Collaboration diagram for vrna_fold_compound:



Data Fields

Common data fields

- [vrna_vc_t](#) type
The type of the [vrna_fold_compound](#).
- 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.
- struct [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.
- struct [vrna_param_t](#) * [params](#)
The precomputed free energy contributions for each type of loop.
- struct [vrna_exp_param_t](#) * [exp_params](#)
The precomputed free energy contributions as Boltzmann factors.
- int * [iindx](#)
DP matrix accessor.
- int * [jindx](#)
DP matrix accessor.

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
- struct [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](#)
S3[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.
- struct [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.

12.35.1 Detailed Description

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.type](#), [vrna_get_fold_compound\(\)](#), [vrna_get_fold_compound_ali\(\)](#), [vrna_free_fold_compound\(\)](#), [VRNA_VC_TYPE_SINGLE](#), [VRNA_VC_TYPE_ALIGNMENT](#)

12.35.2 Field Documentation

12.35.2.1 `vrna_vc_t vrna_fold_compound::type`

The type of the [vrna_fold_compound](#).

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](#). The value specified in this attribute dictates the set of other attributes to use within this data structure.

12.35.2.2 `char* vrna_fold_compound::sequence`

The input sequence string.

Warning

Only available if

```
type==VRNA_VC_TYPE_SINGLE
```

12.35.2.3 `short* vrna_fold_compound::sequence_encoding`

Numerical encoding of the input sequence.

See also

[vrna_sequence_encode\(\)](#)

Warning

Only available if

```
type==VRNA_VC_TYPE_SINGLE
```


12.35.2.4 char* vrna_fold_compound::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_get_indx\(\)](#), [vrna_get_ptypes\(\)](#)

12.35.2.5 char* vrna_fold_compound::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
```

12.35.2.6 struct vrna_sc_t* vrna_fold_compound::sc

The soft constraints for usage in structure prediction and evaluation.

Warning

Only available if

```
type==VRNA_VC_TYPE_SINGLE
```

12.35.2.7 char** vrna_fold_compound::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
```

12.35.2.8 unsigned int vrna_fold_compound::n_seq

The number of sequences in the alignment.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.9 char* vrna_fold_compound::cons_seq

The consensus sequence of the aligned sequences.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.10 short* vrna_fold_compound::S_cons

Numerical encoding of the consensus sequence.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.11 short vrna_fold_compound::S**

Numerical encoding of the sequences in the alignment.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.12 short vrna_fold_compound::S5**

S5[s][i] holds next base 5' of i in sequence s.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.13 short vrna_fold_compound::S3**

S3[s][i] holds next base 3' of i in sequence s.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.14 int* vrna_fold_compound::pscore

Precomputed array of pair types expressed as pairing scores.

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

12.35.2.15 struct vrna_sc_t** vrna_fold_compound::scs

A set of soft constraints (for each sequence in the alignment)

Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h

12.36 vrna_hc_t Struct Reference

The hard constraints data structure.

Data Fields

- char * [matrix](#)
Upper triangular matrix encoding 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.

12.36.1 Detailed Description

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

```
get_hard_constraints(), vrna_hc_free(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, #VRNA_CONSTRAINT_CONTEXT_EXT_LOOP_ENC, VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC
```

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/constraints.h](#)

12.37 vrna_helix Struct Reference

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/structure_utils.h](#)

12.38 vrna_md_t Struct Reference

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

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 [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 [sfact](#)
Scaling factor for partition function scaling.

12.38.1 Detailed Description

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

See also

[vrna_md_set_default\(\)](#), [vrna_md_set_globals\(\)](#), [vrna_md_update\(\)](#)

12.38.2 Field Documentation

12.38.2.1 int vrna_md_t::dangles

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

Note

Some function do not implement all dangle model but only a subset of (0,1,2,3). Read the documentaion of the particular recurrences or energy evaluation function for information about the provided dangle model.

12.38.2.2 int vrna_md_t::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.

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/model.h](#)

12.39 vrna_mx_mfe_t Struct Reference

Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).

Data Fields

Common fields for MFE matrices

- [vrna_mx_t](#) 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 multibranch 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](#)

Distance Class DP matrices

Note

These data fields are available if

```
vrna_mx_mfe_t.type == VRNA_MX_2DFOLD
```

- int *** E_F5
- int ** I_min_F5
- int ** I_max_F5
- int * k_min_F5
- int * k_max_F5
- int *** E_F3
- int ** I_min_F3
- int ** I_max_F3
- int * k_min_F3
- int * k_max_F3
- int *** E_C
- int ** I_min_C
- int ** I_max_C
- int * k_min_C
- int * k_max_C
- int *** E_M
- int ** I_min_M
- int ** I_max_M
- int * k_min_M
- int * k_max_M
- int *** E_M1
- int ** I_min_M1
- int ** I_max_M1
- int * k_min_M1
- int * k_max_M1
- int *** E_M2
- int ** I_min_M2
- int ** I_max_M2
- int * k_min_M2
- int * k_max_M2
- int ** E_Fc
- int * I_min_Fc
- int * I_max_Fc
- int k_min_Fc
- int k_max_Fc
- int ** E_FcH
- int * I_min_FcH
- int * I_max_FcH
- int k_min_FcH
- int k_max_FcH
- int ** E_Fcl
- int * I_min_Fcl
- int * I_max_Fcl
- int k_min_Fcl
- int k_max_Fcl
- int ** E_FcM
- int * I_min_FcM
- int * I_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_Fcl_rem
- int E_FcM_rem

12.39.1 Detailed Description

Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.40 vrna_mx_pf_t Struct Reference

Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).

Data Fields

Common fields for DP matrices

- [vrna_mx_t](#) type
- unsigned int **length**

Default PF matrices

Note

These data fields are available if

```
vrna_mx_pf_t.type == VRNA_MX_DEFAULT
```

- double * **q**
- double * **qb**
- double * **qm**
- double * **qm1**
- double * **probs**
- double * **q1k**
- double * **qln**
- double * **G**
- double **qo**
- double * **qm2**
- double **qho**
- double **qio**
- double **qmo**
- double * **scale**
- double * **expMLbase**

Distance Class DP matrices

Note

These data fields are available if

```
vrna_mx_pf_t.type == VRNA_MX_2DFOLD
```

- double *** **Q**
- int ** **I_min_Q**
- int ** **I_max_Q**
- int * **k_min_Q**
- int * **k_max_Q**
- double *** **Q_B**
- int ** **I_min_Q_B**
- int ** **I_max_Q_B**
- int * **k_min_Q_B**
- int * **k_max_Q_B**
- double *** **Q_M**

- int ** l_min_Q_M
- int ** l_max_Q_M
- int * k_min_Q_M
- int * k_max_Q_M
- double *** Q_M1
- int ** l_min_Q_M1
- int ** l_max_Q_M1
- int * k_min_Q_M1
- int * k_max_Q_M1
- double *** Q_M2
- int ** l_min_Q_M2
- int ** l_max_Q_M2
- int * k_min_Q_M2
- int * k_max_Q_M2
- double ** Q_c
- int * l_min_Q_c
- int * l_max_Q_c
- int k_min_Q_c
- int k_max_Q_c
- double ** Q_cH
- int * l_min_Q_cH
- int * l_max_Q_cH
- int k_min_Q_cH
- int k_max_Q_cH
- double ** Q_cl
- int * l_min_Q_cl
- int * l_max_Q_cl
- int k_min_Q_cl
- int k_max_Q_cl
- double ** Q_cM
- int * l_min_Q_cM
- int * l_max_Q_cM
- int k_min_Q_cM
- int k_max_Q_cM
- double * Q_rem
- double * Q_B_rem
- double * Q_M_rem
- double * Q_M1_rem
- double * Q_M2_rem
- double Q_c_rem
- double Q_cH_rem
- double Q_cl_rem
- double Q_cM_rem

12.40.1 Detailed Description

Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).

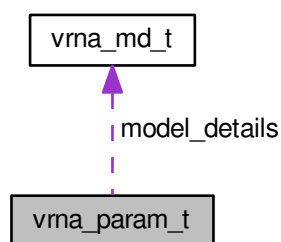
The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h](#)

12.41 vrna_param_t Struct Reference

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for `vrna_param_t`:



Data Fields

- double `temperature`
Temperature used for loop contribution scaling.
- `vrna_md_t` `model_details`
Model details to be used in the recursions.

12.41.1 Detailed Description

The datastructure that contains temperature scaled energy parameters.

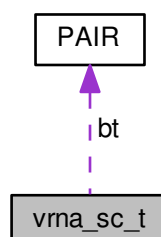
The documentation for this struct was generated from the following file:

- `/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/params.h`

12.42 `vrna_sc_t` Struct Reference

The soft constraints data structure.

Collaboration diagram for `vrna_sc_t`:



Data Fields

- `int ** free_energies`
Energy contribution for unpaired sequence stretches.
- `int * en_basepair`
Energy contribution for base pairs.
- `FLT_OR_DBL ** boltzmann_factors`
Boltzmann Factors of the energy contributions for unpaired sequence stretches.
- `FLT_OR_DBL * exp_en_basepair`
Boltzmann Factors of the energy contribution for base pairs.
- `int * en_stack`
Pseudo Energy contribution per base pair involved in a stack.
- `FLT_OR_DBL * exp_en_stack`
Boltzmann weighted pseudo energy contribution per nucleotide involved in a stack.
- `int(* f)(int, int, int, int, char, void *)`
A function pointer used for pseudo energy contribution in MFE calculations.
- `PAIR *(* bt)(int, int, int, int, char, void *)`
A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.
- `FLT_OR_DBL(* exp_f)(int, int, int, int, char, void *)`
A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.
- `void(* pre)(vrna_fold_compound *, char)`
A function pointer to some generalized soft constraints preprocessing function.
- `void(* post)(vrna_fold_compound *, char)`
A function pointer to some generalized soft constraints postprocessing function.
- `void * data`
A pointer to the data object provided for for pseudo energy contribution functions of the generalized soft constraints feature.

12.42.1 Detailed Description

The soft constraints data structure.

12.42.2 Field Documentation

12.42.2.1 `int(* vrna_sc_t::f)(int, int, int, int, char, void *)`

A function pointer used for pseudo energy contribution in MFE calculations.

See also

[vrna_sc_add_f\(\)](#)

12.42.2.2 `PAIR *(* vrna_sc_t::bt)(int, int, int, int, char, void *)`

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\(\)](#)

12.42.2.3 `FLT_OR_DBL(* vrna_sc_t::exp_f) (int, int, int, int, char, void *)`

A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.

See also

[vrna_sc_add_exp_f\(\)](#)

12.42.2.4 `void(* vrna_sc_t::pre) (vrna_fold_compound *, char)`

A function pointer to some generalized soft constraints preprocessing function.

This function will be called just before the forward recursions start

12.42.2.5 `void(* vrna_sc_t::post) (vrna_fold_compound *, char)`

A function pointer to some generalized soft constraints postprocessing function.

This function will be called right after the forward recursions or backtracking, whatever is last.

The documentation for this struct was generated from the following file:

- `/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/constraints.h`

12.43 `vrna_sol_TwoD_pf_t` Struct Reference

Solution element returned from [vrna_TwoD_pf_fold\(\)](#)

Data Fields

- `int k`
Distance to first reference.
- `int l`
Distance to second reference.
- `FLT_OR_DBL q`
partition function

12.43.1 Detailed Description

Solution element returned from [vrna_TwoD_pf_fold\(\)](#)

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_TwoD_pf_fold\(\)](#)

The documentation for this struct was generated from the following file:

- `/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dpfold.h`

12.44 vrna_sol_TwoD_t Struct Reference

Solution element returned from [vrna_TwoD_fold\(\)](#)

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.

12.44.1 Detailed Description

Solution element returned from [vrna_TwoD_fold\(\)](#)

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood. The data-structure 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* containing the secondary structure representative,

A value of [INF](#) in k denotes the end of a list

See also

[vrna_TwoD_fold\(\)](#)

The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dfold.h](#)

Chapter 13

File Documentation

13.1 `/home/mescalín/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_epars.h` File Reference

Free energy parameters for parameter file conversion.

13.1.1 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

13.2 `/home/mescalín/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_intloops.h` File Reference

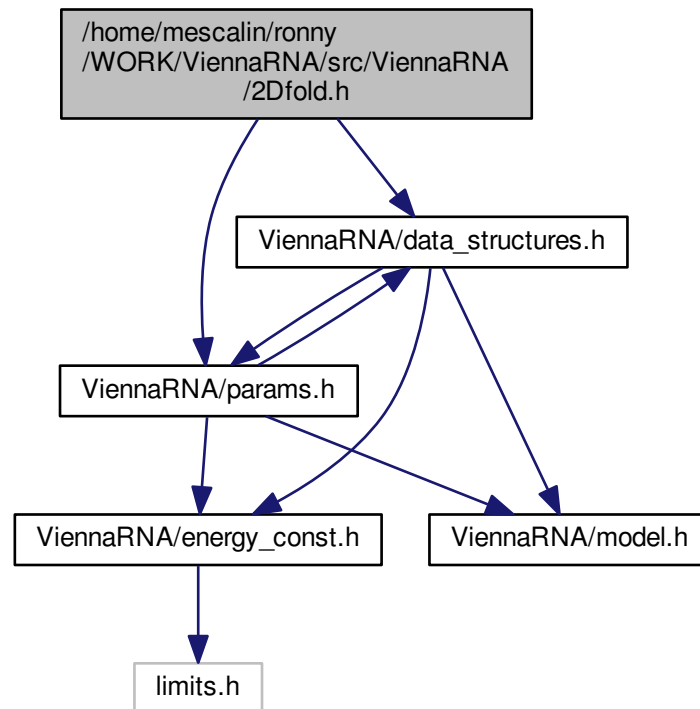
Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

13.2.1 Detailed Description

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

13.3 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dfold.h File Reference

Include dependency graph for 2Dfold.h:



Data Structures

- struct `vrna_sol_TwoD_t`
Solution element returned from `vrna_TwoD_fold()`
- struct `TwoDfold_vars`
Variables compound for 2Dfold MFE folding.

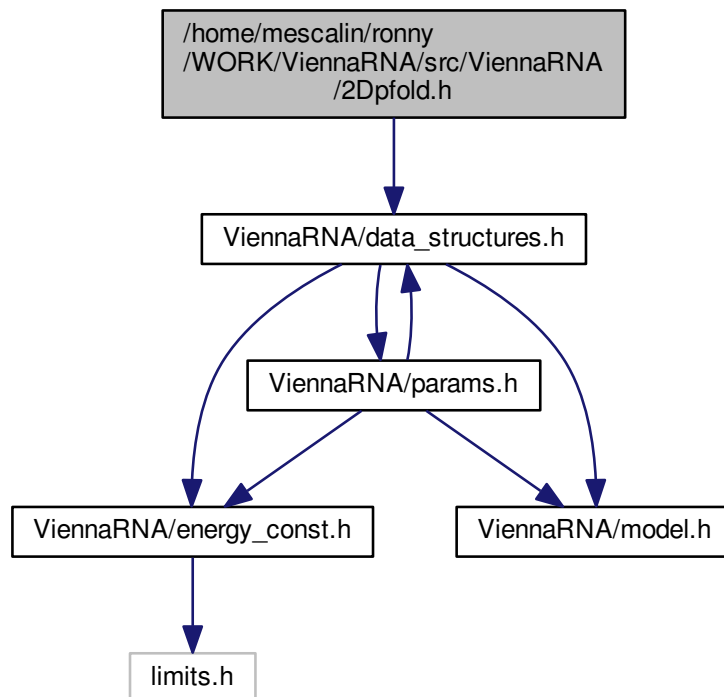
Functions

- `vrna_sol_TwoD_t * vrna_TwoD_fold (vrna_fold_compound *vc, int distance1, int distance2)`
Compute MFE's and representative for distance partitioning.
- `char * vrna_TwoD_backtrack5 (vrna_fold_compound *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.

13.4 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dpfold.h File Reference

Include dependency graph for 2Dpfold.h:



Data Structures

- struct [vrna_sol_TwoD_pf_t](#)
Solution element returned from [vrna_TwoD_pf_fold\(\)](#)
- struct [TwoDpfold_vars](#)
Variables compound for 2Dfold partition function folding.

Functions

- [vrna_sol_TwoD_pf_t](#) * [vrna_TwoD_pf_fold](#) ([vrna_fold_compound](#) *vc, int maxDistance1, int maxDistance2)
Compute the partition function for all distance classes.
- char * [vrna_TwoD_pbacktrack](#) ([vrna_fold_compound](#) *vc, int d1, int d2)
Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.
- char * [vrna_TwoD_pbacktrack5](#) ([vrna_fold_compound](#) *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.

13.4.1 Function Documentation

13.4.1.1 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` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, and `vrna_free_fold_compound()` instead!

Parameters

<i>seq</i>	the RNA sequence in uppercase format with letters from the alphabet {AUCG}
<i>structure1</i>	the first reference structure in dot-bracket notation
<i>structure2</i>	the second reference structure in dot-bracket notation
<i>circ</i>	a switch indicating if the sequence is linear (0) or circular (1)

Returns

the datastructure containing all necessary partition function attributes

13.4.1.2 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 `vrna_TwoDpfold_get_vars()` or `vrna_TwoDpfold_get_vars_from_MFE()`

Deprecated Use the new API that relies on `vrna_fold_compound` and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, and `vrna_free_fold_compound()` instead!

See also

`vrna_TwoDpfold_get_vars()`, `vrna_TwoDpfold_get_vars_from_MFE()`

Parameters

<i>vars</i>	the datastructure to be free'd
-------------	--------------------------------

13.4.1.3 `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 `vrna_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 distances 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=-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](#) and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, and `vrna_free_fold_compound()` instead!

See also

`vrna_TwoDpfold_get_vars()`, `destroy_TwoDpfold_variables()`, `#TwoDpfold_solution`

Parameters

<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
<i>maxDistance1</i>	the maximum basepair distance to reference1 (may be -1)
<i>maxDistance2</i>	the maximum basepair distance to reference2 (may be -1)

Returns

a list of partition funtions for the appropriate distance classes

13.4.1.4 `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 `vrna_TwoDpfold()` preceding this function is mandatory!

Deprecated Use the new API that relies on [vrna_fold_compound](#) and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, `vrna_TwoD_pbacktrack()`, and `vrna_free_fold_compound()` instead!

See also

`vrna_TwoDpfold()`

Parameters

<i>in</i>	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
<i>in</i>	<i>d1</i>	the distance to reference1 (may be -1)
<i>in</i>	<i>d2</i>	the distance to reference2

Returns

A sampled secondary structure in dot-bracket notation

13.4.1.5 `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 `vrna_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 `vrna_TwoDpfold()` preceding this function is mandatory!

Deprecated Use the new API that relies on [vrna_fold_compound](#) and the corresponding functions `vrna_get_fold_compound_2D()`, `vrna_TwoD_pf_fold()`, `vrna_TwoD_pbacktrack5()`, and `vrna_free_fold_compound()` instead!

See also

[TwoDpfold_pbacktrack\(\)](#), `vrna_TwoDpfold()`

Parameters

<i>in</i>	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
<i>in</i>	<i>d1</i>	the distance to reference1 (may be -1)
<i>in</i>	<i>d2</i>	the distance to reference2
<i>in</i>	<i>length</i>	the length of the structure beginning from the 5' end

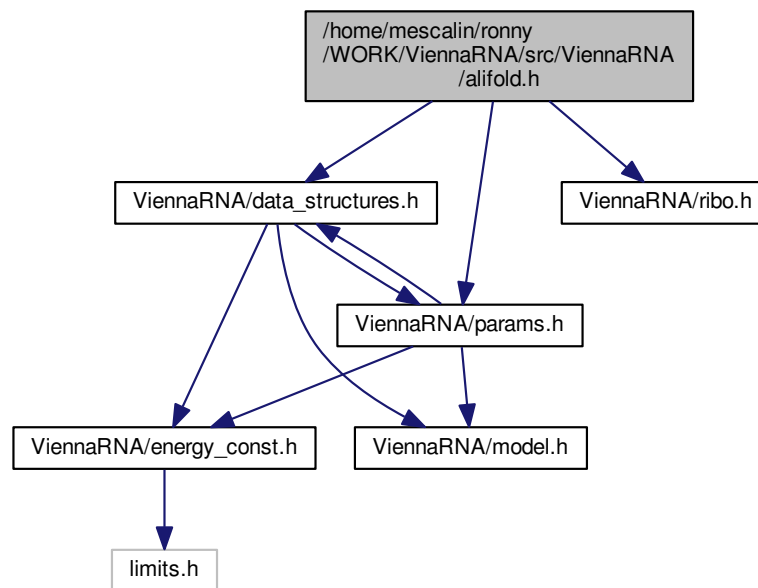
Returns

A sampled secondary structure in dot-bracket notation

13.5 /home/mescalini/ronny/WORK/ViennaRNA/src/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_ali_fold` (`vrna_fold_compound` *vc, char *structure)
Compute MFE and according consensus structure of an alignment of sequences.
- 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 `vrna_ali_pf_fold` (`vrna_fold_compound` *vc, char *structure, `plist` **pl)
Compute partition function and base pair probabilities for a sequence alignment.
- `pair_info` * `vrna_ali_get_pair_info` (`vrna_fold_compound` *vc, const char *structure, double threshold)
Retrieve an array of `pair_info` structures from precomputed pair probabilities.
- float `alipf_fold_par` (const char **sequences, char *structure, `plist` **pl, `vrna_exp_param_t` *parameters, int calculate_bppm, int is_constrained, int is_circular)
- float `alipf_fold` (const char **sequences, char *structure, `plist` **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 `pair_info` structs. The list is terminated by the first entry with `pi.i = 0`.
- float `alipf_circ_fold` (const char **sequences, char *structure, `plist` **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 * `vrna_alipf_backtrack` (`vrna_fold_compound` *vc, double *prob)

Sample a consensus secondary structure from the Boltzmann ensemble according its probability

- char * `alipf_backtrack` (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, int **pscore)

Get pointers to (almost) all relevant 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.

13.5.1 Detailed Description

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

13.5.2 Function Documentation

13.5.2.1 `pair_info`* vrna_alipf_get_pair_info (vrna_fold_compound * vc, const char * structure, double threshold)

Retrieve an array of `pair_info` structures from precomputed pair probabilities.

This array of structures contains information about positionwise pair probabilities, base pair entropy and more

See also

`pair_info`, and `vrna_alipf_fold()`

Parameters

<code>vc</code>	The <code>vrna_fold_compound</code> of type <code>VRNA_VC_TYPE_ALIGNMENT</code> with precomputed partition function matrices
<code>structure</code>	An optional structure in dot-bracket notation (Maybe NULL)
<code>threshold</code>	Do not include results with pair probabilities below threshold

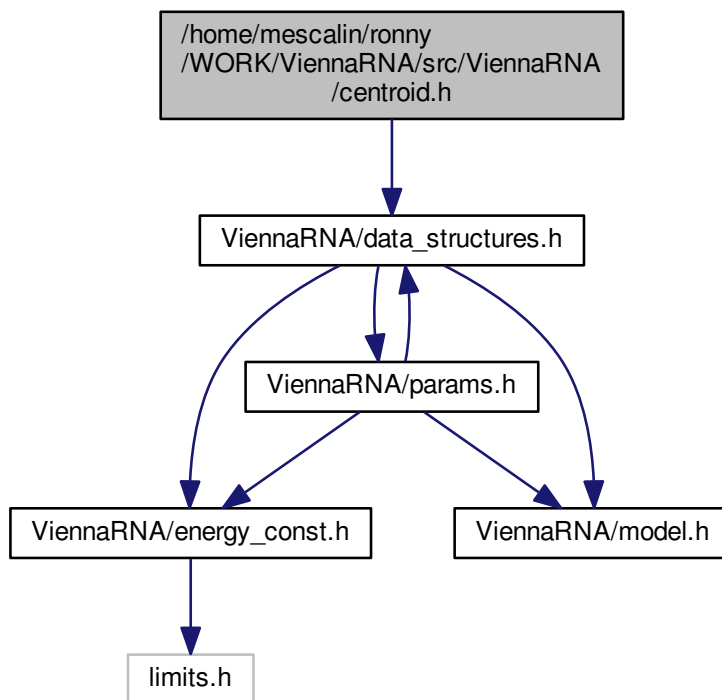
Returns

The `pair_info` array

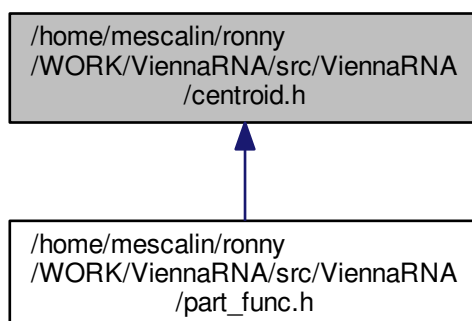
13.6 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/centroid.h File Reference

Centroid structure computation.

Include dependency graph for centroid.h:



This graph shows which files directly or indirectly include this file:



Functions

- `char * vrna_get_centroid_struct_pl` (int *length*, double **dist*, `plist` **pl*)

Get the centroid structure of the ensemble.

- `char * get_centroid_struct_pl` (int *length*, double **dist*, `plist` **pl*)

Get the centroid structure of the ensemble.

- `char * vrna_get_centroid_struct_pr` (int *length*, double **dist*, FLT_OR_DBL **pr*)

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.

- `char * vrna_get_centroid_struct` (`vrna_fold_compound` **vc*, double **dist*)

Get the centroid structure of the ensemble.

13.6.1 Detailed Description

Centroid structure computation.

13.6.2 Function Documentation

13.6.2.1 `char* get_centroid_struct_pl (int length, double * dist, plist * pl)`

Get the centroid structure of the ensemble.

Deprecated This function was renamed to `vrna_get_centroid_struct_pl()`

13.6.2.2 `char* get_centroid_struct_pr (int length, double * dist, FLT_OR_DBL * pr)`

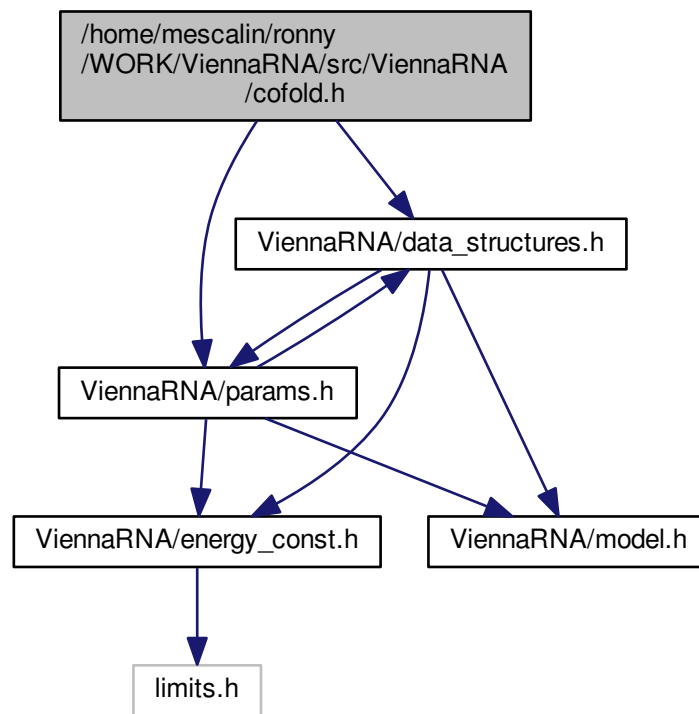
Get the centroid structure of the ensemble.

Deprecated This function was renamed to `vrna_get_centroid_struct_pr()`

13.7 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/cofold.h File Reference

MFE version of cofolding routines.

Include dependency graph for cofold.h:



Functions

- 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.
- float `vrna_cofold` (`vrna_fold_compound` *vc, char *structure)
Compute the minimum free energy of two interacting RNA molecules.
- 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 `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.
- [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.
- [SOLUTION](#) * [vrna_zukersubopt](#) ([vrna_fold_compound](#) *vc)
Compute Zuker type suboptimal structures.
- void [get_monomere_mfes](#) (float *e1, float *e2)
get_monomer_free_energies
- void [initialize_cofold](#) (int length)

13.7.1 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.

13.7.2 Function Documentation

13.7.2.1 void [get_monomere_mfes](#) (float * *e1*, float * *e2*)

[get_monomer_free_energies](#)

Export monomer free energies out of cofold arrays

Parameters

<i>e1</i>	A pointer to a variable where the energy of molecule A will be written to
<i>e2</i>	A pointer to a variable where the energy of molecule B will be written to

13.7.2.2 void [initialize_cofold](#) (int *length*)

allocate arrays for folding

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

13.8 /home/mescalini/ronny/WORK/ViennaRNA/src/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

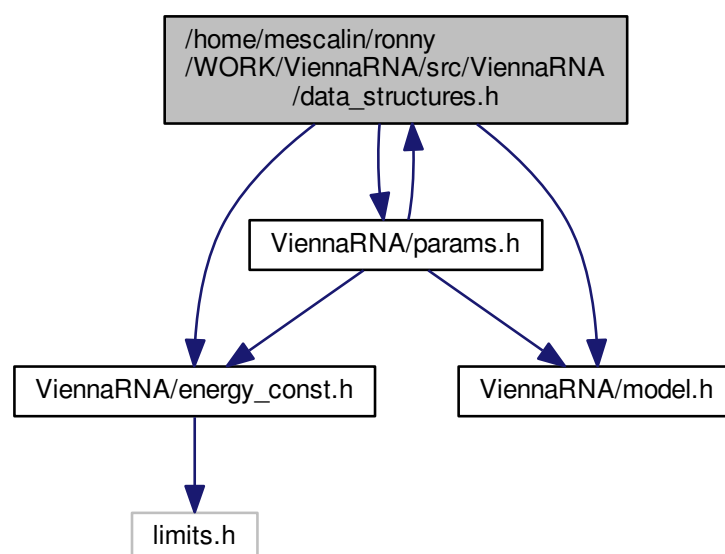
- void `convert_parameter_file` (const char *iname, const char *oname, unsigned int options)

13.8.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

13.9 /home/mescaline/ronny/WORK/ViennaRNA/src/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 [plist](#)
this datastructure is used as input parameter in functions of [PS_dot.h](#) and others
- struct [cpair](#)
this datastructure is used as input parameter in functions of [PS_dot.c](#)
- struct [sect](#)
Stack of partial structures for backtracking.
- struct [bondT](#)
Base pair.
- struct [bondTEn](#)
Base pair with associated energy.
- struct [PAIR](#)
Base pair data structure used in [subopt.c](#).
- struct [INTERVAL](#)
Sequence interval stack element used in [subopt.c](#).
- struct [SOLUTION](#)
Solution element from [subopt.c](#).
- struct [cofoldF](#)
- struct [ConcEnt](#)
- struct [pairpro](#)
- struct [pair_info](#)
A base pair info structure.
- struct [move_t](#)
- struct [intermediate_t](#)
- struct [path_t](#)
- struct [pu_contrib](#)
contributions to p_u
- struct [interact](#)
- struct [pu_out](#)
Collection of all `free_energy` of beeing unpaired values for output.
- struct [constrain](#)
constraints for cofolding
- struct [duplexT](#)
- struct [folden](#)
- struct [snoopT](#)
- struct [dupVar](#)
- struct [vrna_mx_mfe_t](#)
Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).
- struct [vrna_mx_pf_t](#)
Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna_fold_compound](#).
- struct [vrna_fold_compound](#)
The most basic data structure required by many functions throughout the [RNAlib](#).

Macros

- `#define MAXDOS 1000`
Maximum density of states discretization for subopt.
- `#define VRNA_OPTION_MFE 1`
Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.
- `#define VRNA_OPTION_PF 2`
Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.
- `#define VRNA_OPTION_EVAL_ONLY 8`
Option flag to specify that neither MFE, nor PF DP matrices are required.

Enumerations

- `enum vrna_mx_t { VRNA_MX_DEFAULT, VRNA_MX_LFOLD, VRNA_MX_2DFOLD }`
An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.
- `enum vrna_vc_t { VRNA_VC_TYPE_SINGLE, VRNA_VC_TYPE_ALIGNMENT }`
An enumerator that is used to specify the type of a `vrna_fold_compound`.

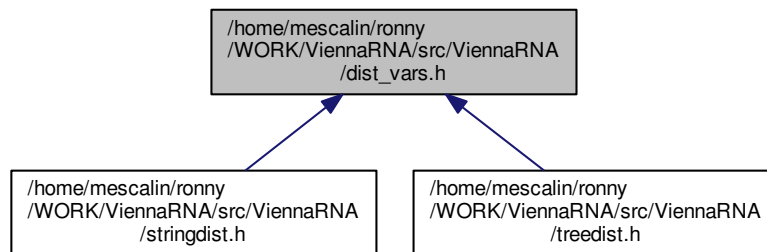
Functions

- `vrna_fold_compound * vrna_get_fold_compound (const char *sequence, vrna_md_t *md_p, unsigned int options)`
Retrieve a `vrna_fold_compound` data structure for single sequences and hybridizing sequences.
- `vrna_fold_compound * vrna_get_fold_compound_ali (const char **sequences, vrna_md_t *md_p, unsigned int options)`
Retrieve a `vrna_fold_compound` data structure for sequence alignments.
- `void vrna_params_update (vrna_fold_compound *vc, vrna_param_t *par)`
Update/Reset energy parameters data structure within a `vrna_fold_compound`.
- `void vrna_exp_params_update (vrna_fold_compound *vc, vrna_exp_param_t *params)`
Update the energy parameters for subsequent partition function computations.
- `void vrna_exp_params_rescale (vrna_fold_compound *vc, double *mfe)`
Rescale Boltzmann factors for partition function computations.
- `void vrna_free_fold_compound (vrna_fold_compound *vc)`
Free memory occupied by a `vrna_fold_compound`.
- `void vrna_free_mfe_matrices (vrna_fold_compound *vc)`
Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.
- `void vrna_free_pf_matrices (vrna_fold_compound *vc)`
Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

13.10 /home/mescalini/ronny/WORK/ViennaRNA/src/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](#)
- struct [Tree](#)
- struct [swString](#)

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.

13.10.1 Detailed Description

Global variables for Distance-Package.

13.10.2 Variable Documentation

13.10.2.1 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

13.10.2.2 int [cost_matrix](#)

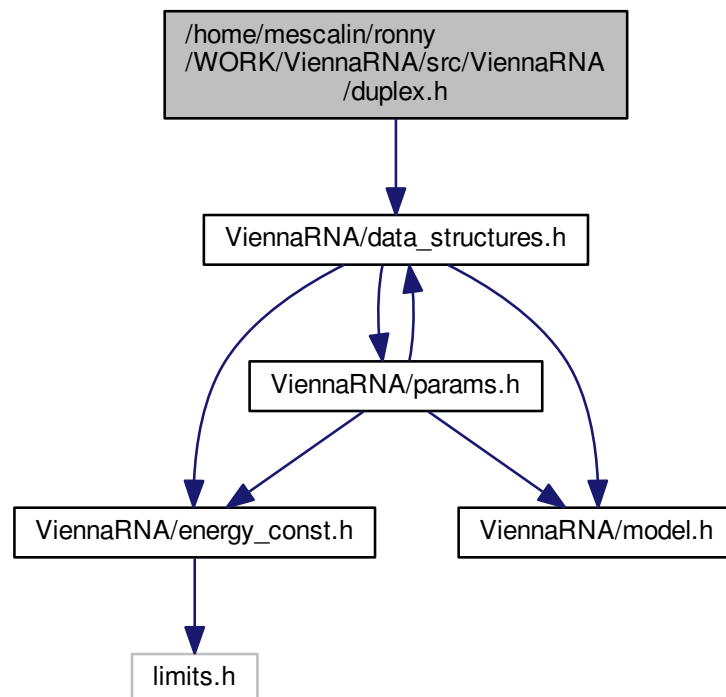
Specify the cost matrix to be used for distance calculations.

if 0, use the default cost matrix (upper matrix in example), otherwise use Shapiro's costs (lower matrix).

13.11 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/duplex.h File Reference

Duplex folding function declarations...

Include dependency graph for duplex.h:



13.11.1 Detailed Description

Duplex folding function declarations...

13.12 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/edit_cost.h File Reference

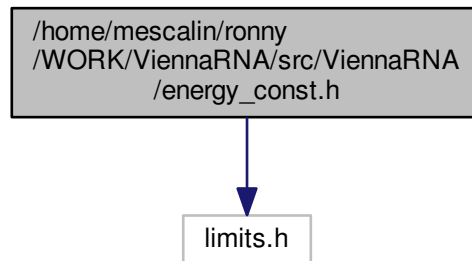
global variables for Edit Costs included by `treedist.c` and `stringdist.c`

13.12.1 Detailed Description

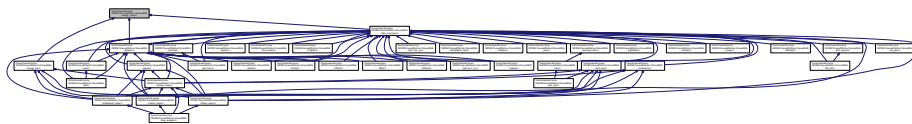
global variables for Edit Costs included by `treedist.c` and `stringdist.c`

13.13 /home/mescal/ronny/WORK/ViennaRNA/src/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`

13.13.1 Detailed Description

energy constants

13.13.2 Macro Definition Documentation

13.13.2.1 `#define GASCONST 1.98717 /* in [cal/K] */`

The gas constant

13.13.2.2 `#define K0 273.15`

0 deg Celsius in Kelvin

13.13.2.3 `#define INF 10000000 /* (INT_MAX/10) */`

Infinity as used in minimization routines

13.13.2.4 `#define FORBIDDEN 9999`

forbidden

13.13.2.5 `#define BONUS 10000`

bonus contribution

13.13.2.6 `#define NBPAIRS 7`

The number of distinguishable base pairs

13.13.2.7 `#define TURN 3`

The minimum loop length

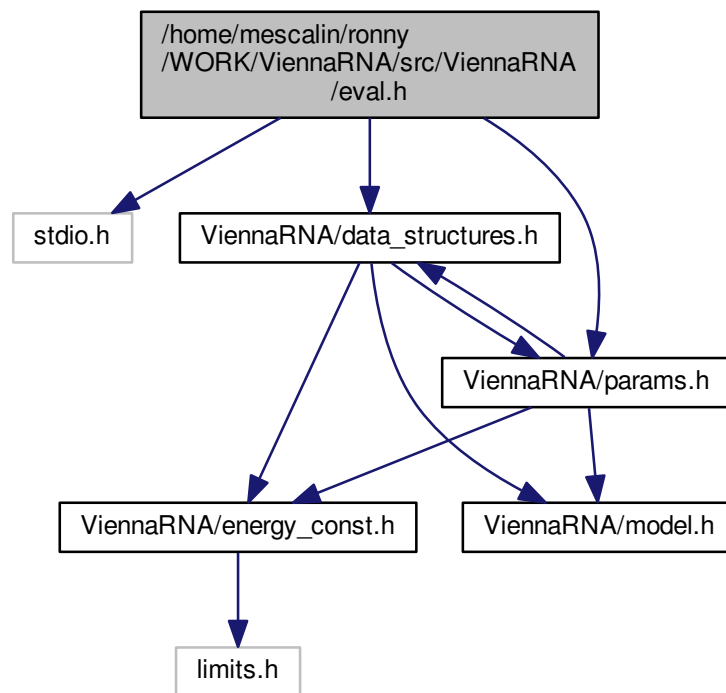
13.13.2.8 `#define MAXLOOP 30`

The maximum loop length

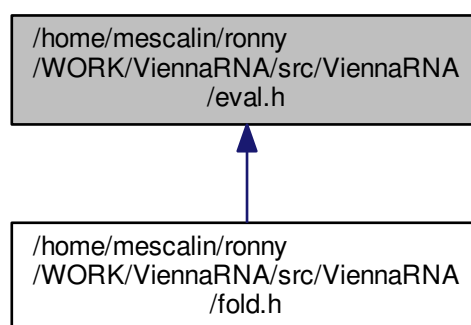
13.14 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/eval.h File Reference

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:



Functions

- float `vrna_eval_structure` (`vrna_fold_compound` *vc, const char *structure)
Calculate the free energy of an already folded RNA.

- float [vrna_eval_covar_structure](#) ([vrna_fold_compound](#) *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](#) *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_simple_verbose](#) (const char *string, const char *structure, FILE *file)
Calculate the free energy of an already folded RNA and print contributions per loop.
- int [vrna_eval_structure_pt](#) ([vrna_fold_compound](#) *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](#) *vc, const short *pt, 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_loop_pt](#) ([vrna_fold_compound](#) *vc, int i, const short *pt)
Calculate energy of a loop.
- float [vrna_eval_move](#) ([vrna_fold_compound](#) *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](#) *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)

Variables

- int [cut_point](#)
set to first pos of second seq for cofolding
- int [eos_debug](#)
verbose info from energy_of_struct

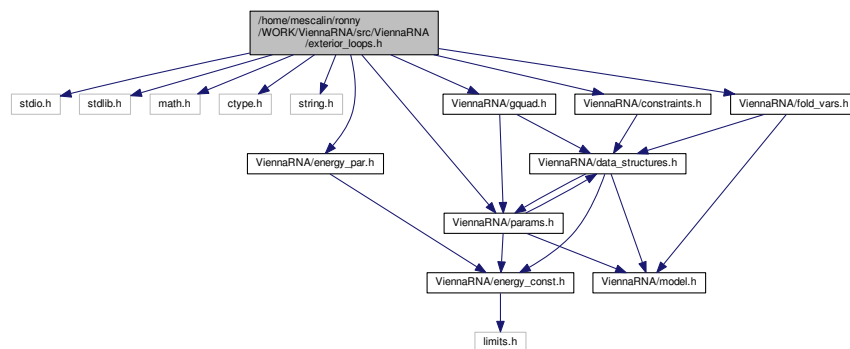
13.14.1 Detailed Description

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

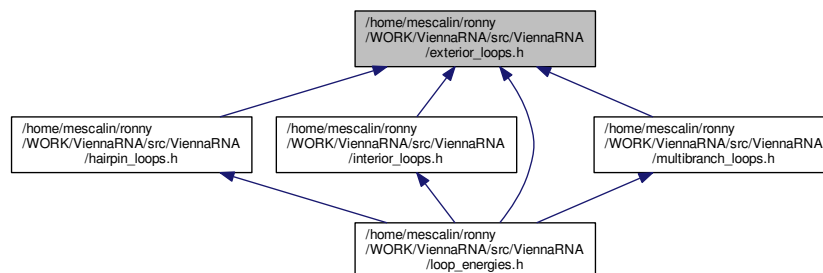
13.15 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/exterior_loops.h File Reference

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

- PRIVATE int [E_ExtLoop](#) (int type, int si1, int sj1, [vrna_param_t](#) *P)
- PRIVATE double [exp_E_ExtLoop](#) (int type, int si1, int sj1, [vrna_exp_param_t](#) *P)
- PRIVATE int [E_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna_param_t](#) *P)
- PRIVATE double [exp_E_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna_exp_param_t](#) *P)

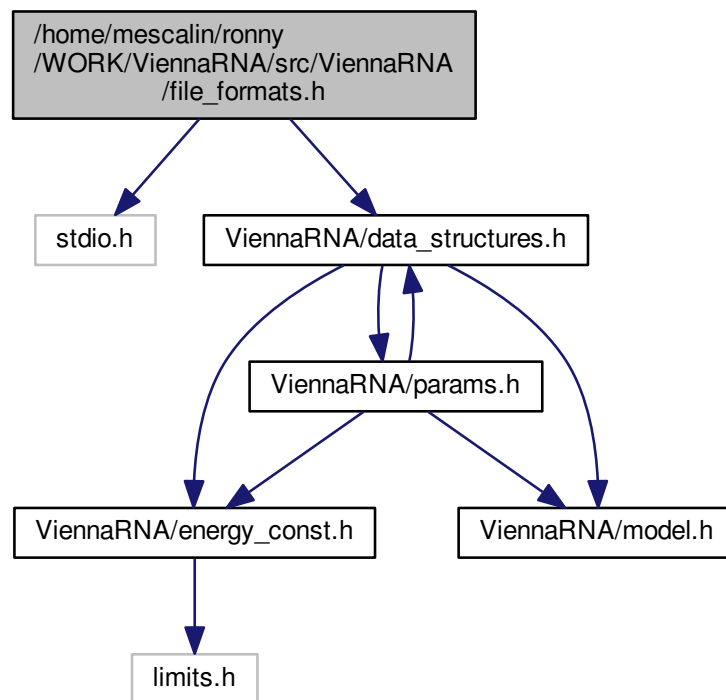
13.15.1 Detailed Description

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

13.16 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/file_formats.h File Reference

Functions dealing with file formats for RNA sequences, structures, and alignments.

Include dependency graph for file_formats.h:



Functions

- void `vrna_structure_print_hx` (const char *seq, const char *db, float energy, FILE *file)
Print a secondary structure as helix list.
- void `vrna_structure_print_ct` (const char *seq, const char *db, float energy, const char *identifier, FILE *file)
Print a secondary structure as connect table.
- void `vrna_structure_print_bpseq` (const char *seq, const char *db, FILE *file)
Print a secondary structure in bpseq format.
- unsigned int `vrna_read_fasta_record` (char **header, char **sequence, char ***rest, FILE *file, unsigned int options)
Get a (fasta) data set from a file or stdin.
- void `vrna_extract_record_rest_constraint` (char **cstruc, const char **lines, unsigned int option)
Extract a hard constraint encoded as pseudo dot-bracket string.
- int `vrna_read_SHAPE_file` (const char *file_name, int length, double default_value, char *sequence, double *values)
Read data from a given SHAPE reactivity input file.
- `plist * vrna_read_constraints_file` (const char *filename, unsigned int length, unsigned int options)
Read constraints from an input file.

- unsigned int [read_record](#) (char **header, char **sequence, char ***rest, unsigned int options)
Get a data record from stdin.

13.16.1 Detailed Description

Functions dealing with file formats for RNA sequences, structures, and alignments.

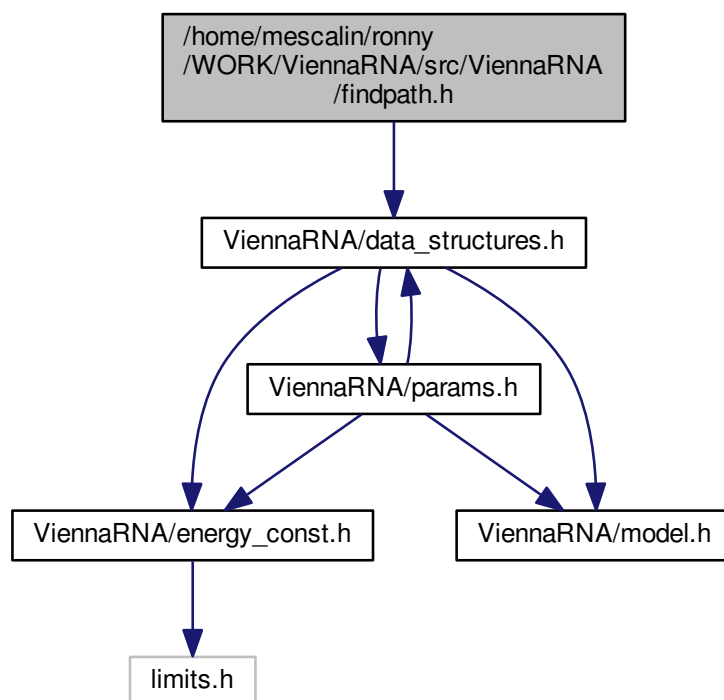
See also

[Constraints Definition File](#)

13.17 `/home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/findpath.h` File Reference

Compute direct refolding paths between two secondary structures.

Include dependency graph for `findpath.h`:



Functions

- 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)
- [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)

- void `free_path` (`path_t` *`path`)
Free memory allocated by `get_path()` function.

13.17.1 Detailed Description

Compute direct refolding paths between two secondary structures.

13.17.2 Function Documentation

13.17.2.1 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)

Parameters

<code>seq</code>	RNA sequence
<code>struc1</code>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<code>struc2</code>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<code>max</code>	integer how many structures are being kept during the search

Returns

the saddle energy in 10cal/mol

13.17.2.2 `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)

Parameters

<code>seq</code>	RNA sequence
<code>s1</code>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<code>s2</code>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<code>maxkeep</code>	integer how many structures are being kept during the search

Returns

direct refolding path between two structures

13.17.2.3 void `free_path` (`path_t` * `path`)

Free memory allocated by `get_path()` function.

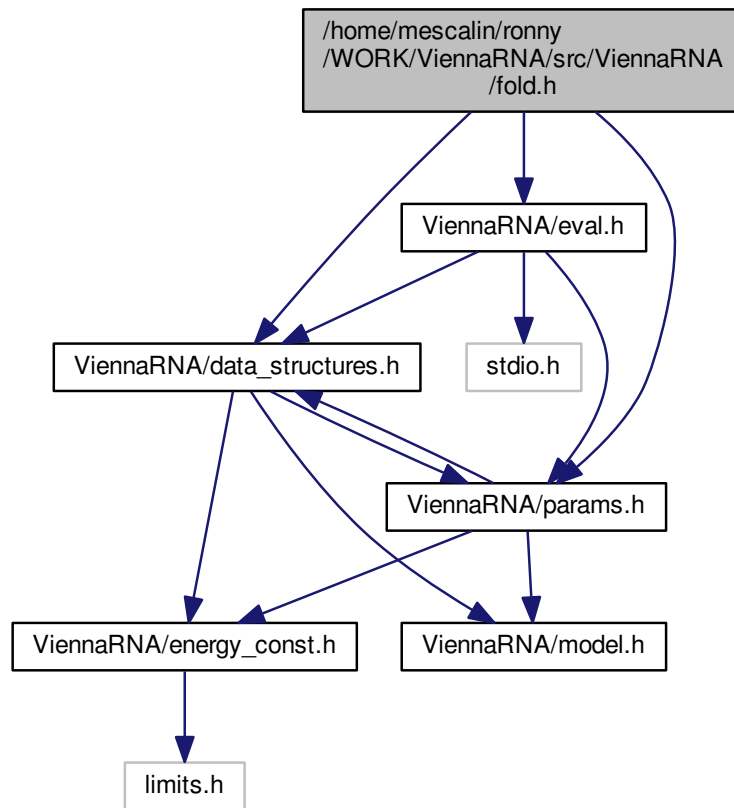
Parameters

<code>path</code>	pointer to memory to be freed
-------------------	-------------------------------

13.18 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/fold.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for fold.h:



Functions

- float `vrna_fold` (`vrna_fold_compound` *vc, char *structure)
Compute minimum free energy and an appropriate secondary structure of an 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 *Fcl_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 *Fcl_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)

13.18.1 Detailed Description

MFE calculations for single RNA sequences.

This file includes (almost) all function declarations within the RNAlib that are related to MFE folding...

13.18.2 Function Documentation

13.18.2.1 int [LoopEnergy](#) (int *n1*, int *n2*, int *type*, int *type_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*)

Deprecated {This function is deprecated and will be removed soon. Use [E_IntLoop\(\)](#) instead!}

13.18.2.2 int [HairpinE](#) (int *size*, int *type*, int *si1*, int *sj1*, const char * *string*)

Deprecated {This function is deprecated and will be removed soon. Use [E_Hairpin\(\)](#) instead!}

13.18.2.3 void [initialize_fold](#) (int *length*)

Allocate arrays for folding

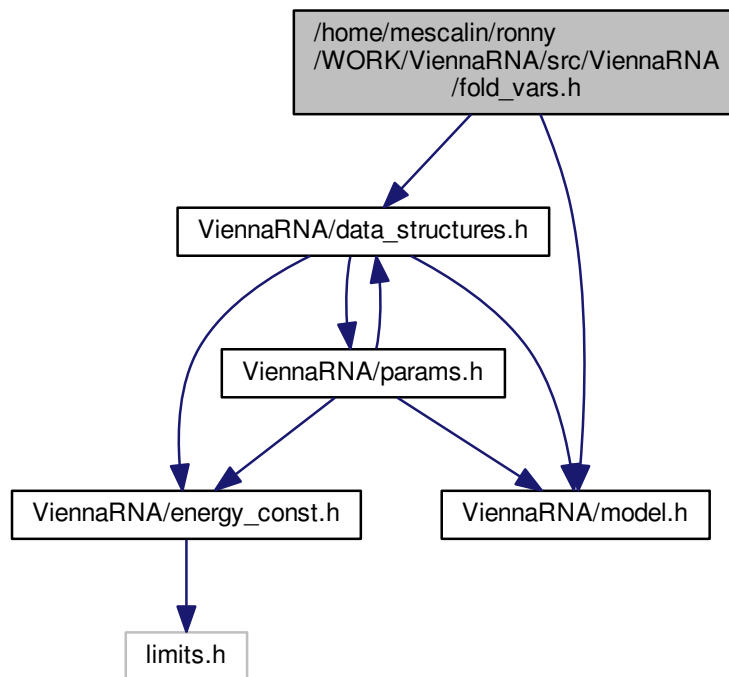
Deprecated {This function is deprecated and will be removed soon!}

See [vrna_fold\(\)](#) and [vrna_fold_compound](#) for the usage of the new API!

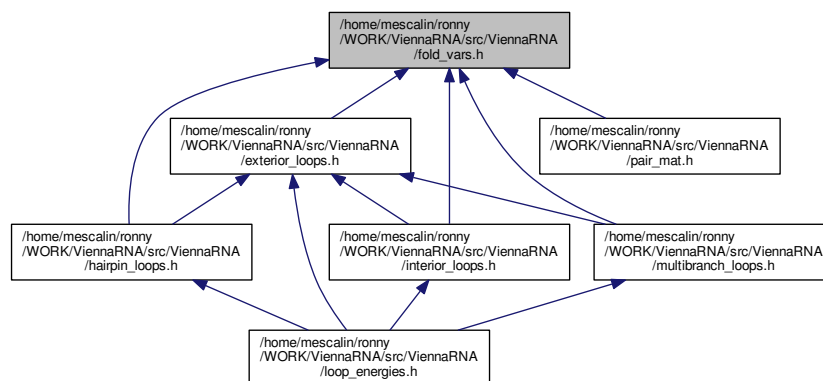
13.19 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/fold_vars.h File Reference

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 separated 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.

13.19.1 Detailed Description

Here all all declarations of the global variables used throughout RNAlib.

13.19.2 Variable Documentation

13.19.2.1 char* RibosumFile

warning this variable will vanish in the future ribosums will be compiled in instead

13.19.2.2 int james_rule

interior loops of size 2 get energy 0.8Kcal and no mismatches, default 1

13.19.2.3 int logML

use logarithmic multiloop energy function

13.19.2.4 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 two 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 [PS_rna_plot\(\)](#) and [PS_dot_plot\(\)](#) to mark the chain break in postscript plots.

13.19.2.5 [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!

13.19.2.6 FLT_OR_DBL* [pr](#)

A pointer to the base pair probability matrix.

Deprecated Do not use this variable anymore!

13.19.2.7 int* iindx

index array to move through pr.

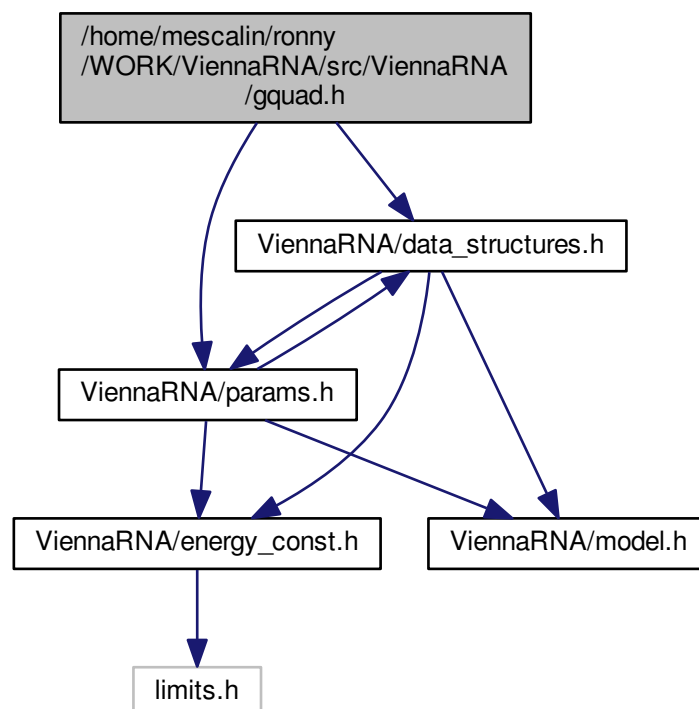
The probability for base i and j to form a pair is in pr[iindx[i]-j].

Deprecated Do not use this variable anymore!

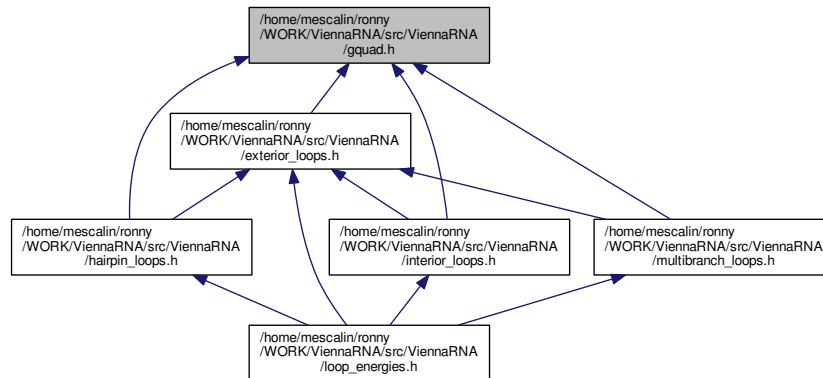
13.20 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/gquad.h File Reference

Various functions related to G-quadruplex computations.

Include dependency graph for gquad.h:



This graph shows which files directly or indirectly include this file:



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)

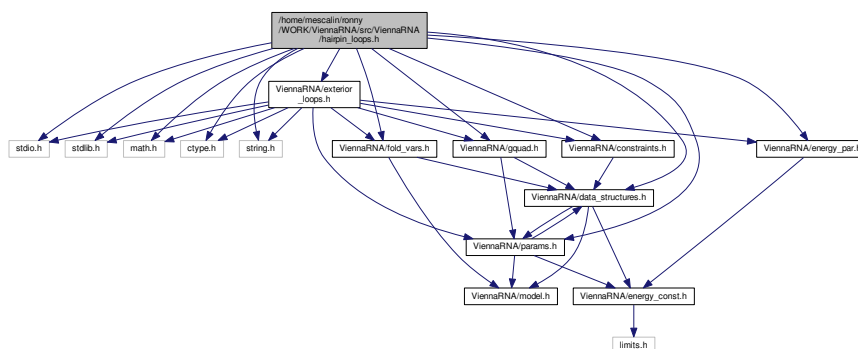
13.20.1 Detailed Description

Various functions related to G-quadruplex computations.

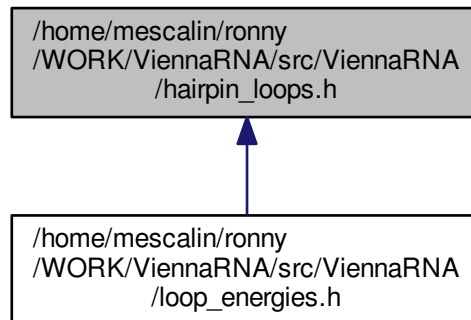
13.21 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/hairpin_loops.h File Reference

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 double [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.
- PRIVATE int [vrna_eval_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
Evaluate free energy of a hairpin loop.
- PRIVATE int [vrna_eval_ext_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
Evaluate free energy of an exterior hairpin loop.
- PRIVATE int [vrna_E_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
Evaluate the free energy of a hairpin loop and consider possible hard constraints.
- PRIVATE int [vrna_E_ext_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.
- PRIVATE double [vrna_exp_E_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j)
High-Level function for hairpin loop energy evaluation (partition function variant)
- PRIVATE int [vrna_BT_hp_loop](#) ([vrna_fold_compound](#) *vc, int i, int j, int en, [bondT](#) *bp_stack, int *stack_count)
Backtrack a hairpin loop closed by (i, j) .

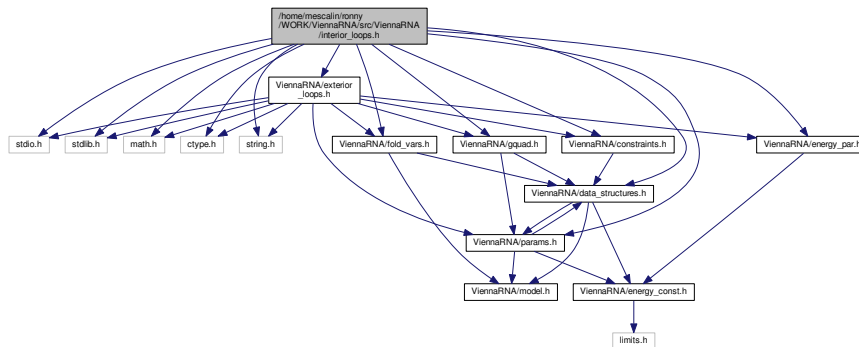
13.21.1 Detailed Description

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

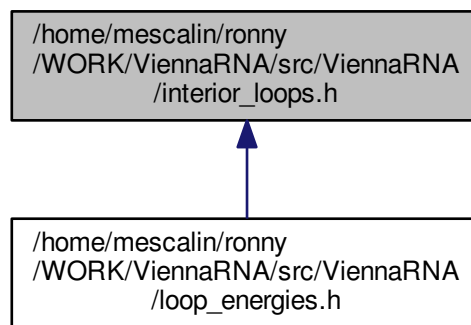
13.22 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/interior_loops.h File Reference

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 double [exp_E_IntLoop](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, [vrna_exp_param_t](#) *P)
 - PRIVATE int [E_stack](#) (int i, int j, [vrna_fold_compound](#) *vc)
- Evaluate energy of a base pair stack closed by (i,j)*
- PRIVATE int [vrna_BT_stack](#) ([vrna_fold_compound](#) *vc, int *i, int *j, int *en, [bondT](#) *bp_stack, int *stack_count)
- Backtrack a stacked pair closed by (i,j).*
- PRIVATE int [vrna_BT_int_loop](#) ([vrna_fold_compound](#) *vc, int *i, int *j, int en, [bondT](#) *bp_stack, int *stack_count)
- Backtrack an interior loop closed by (i,j).*

13.22.1 Detailed Description

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

13.23 /home/mescal/ronny/WORK/ViennaRNA/src/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`

13.23.1 Detailed Description

Inverse folding routines.

13.24 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/Lfold.h File Reference

Predicting local MFE structures of large sequences.

Functions

- 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 `aliLfold` (const char **strings, char *structure, int maxdist)

13.24.1 Detailed Description

Predicting local MFE structures of large sequences.

13.25 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/loop_energies.h File Reference

Energy evaluation for MFE and partition function calculations.


```

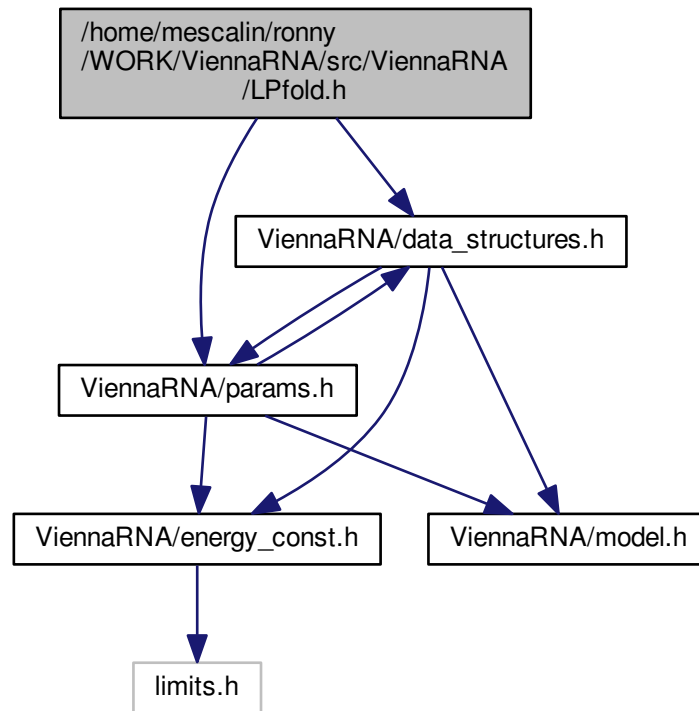
graph TD
    Root["home/masculin/vorony  
WCRK-ViennaRNA/src/ViennaRNA  
Aop_energies.h"] --> Heloin["ViennaRNA/heloin_loops.h"]
    Root --> Multibranch["ViennaRNA/multibranch_loops.h"]
    Root --> Wallpaper["ViennaRNA/wallpaper.h"]
    
    Heloin --> FoldVar["ViennaRNA/fold_var.h"]
    Heloin --> Accustruct["ViennaRNA/accustruct.h"]
    Heloin --> Params["ViennaRNA/params.h"]
    Heloin --> Model["ViennaRNA/model.h"]
    Heloin --> EnergyCons["ViennaRNA/energy_cons.h"]
    
    Multibranch --> FoldVar
    Multibranch --> Accustruct
    Multibranch --> Params
    Multibranch --> Model
    Multibranch --> EnergyCons
    
    Wallpaper --> FoldVar
    Wallpaper --> Accustruct
    Wallpaper --> Params
    Wallpaper --> Model
    Wallpaper --> EnergyCons
    
    FoldVar --> Libm["libm.h"]
    Accustruct --> Libm
    Params --> Libm
    Model --> Libm
    EnergyCons --> Libm
    EnergyPair["ViennaRNA/energy_pair.h"] --> Libm

```

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\(\)](#)].

Generated on Mon May 11 2015 11:28:29 for RNaLib-2.2.0-RC3 by Doxygen

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, struct `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, struct `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)

13.26.1 Detailed Description

Function declarations of partition function variants of the Lfold algorithm.

13.26.2 Function Documentation

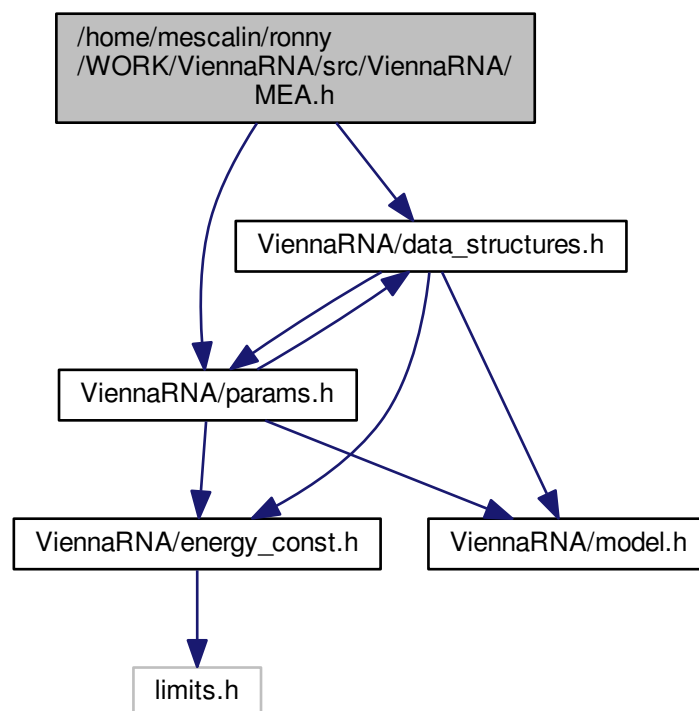
13.26.2.1 void init_pf_foldLP (int length)

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

13.27 /home/mescal/ronny/WORK/ViennaRNA/src/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.

13.27.1 Detailed Description

Computes a MEA (maximum expected accuracy) structure.

13.27.2 Function Documentation

13.27.2.1 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.

13.28 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/mm.h File Reference

Several Maximum Matching implementations.

13.28.1 Detailed Description

Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

13.29 /home/mescal/ronny/WORK/ViennaRNA/src/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_t](#)

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

Macros

- `#define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0`
Default temperature for structure prediction and free energy evaluation in °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_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.

Functions

- `void vrna_md_set_default (vrna_md_t *md)`
Set default model details.
- `void vrna_md_update (vrna_md_t *md)`
Update the model details data structure.
- `void vrna_md_set_globals (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](#)

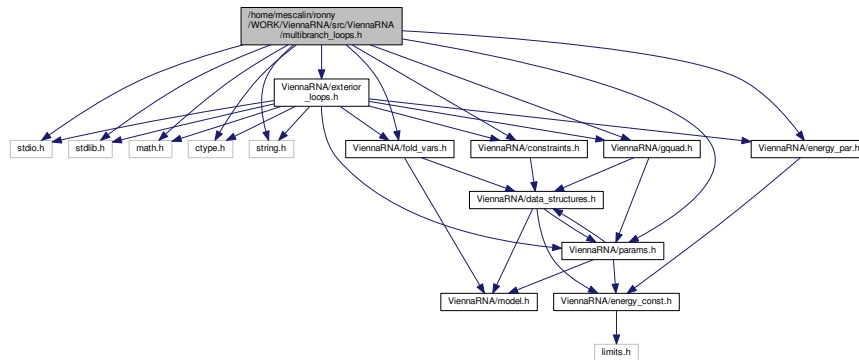
13.29.1 Detailed Description

The model details data structure and its corresponding modifiers.

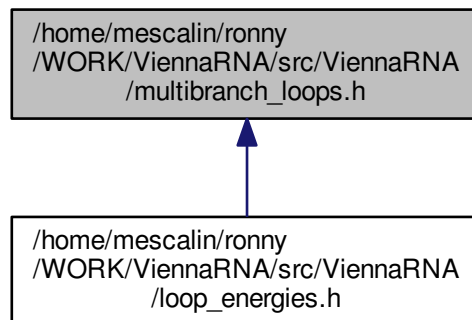
13.30 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/multibranch_loops.h File Reference

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

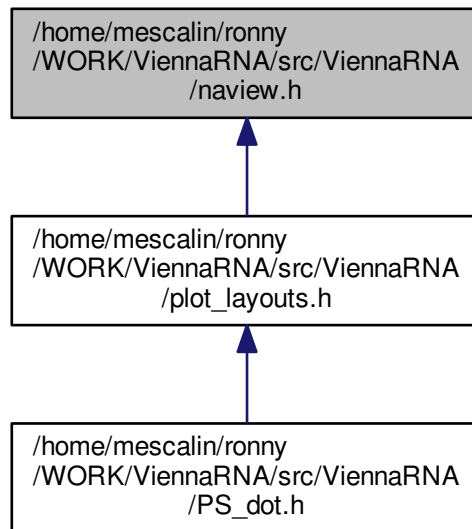
- PRIVATE int `E_mb_loop_stack` (int i, int j, `vrna_fold_compound` *vc)
Evaluate energy of a multi branch helices stacking onto closing pair (i,j)
- PRIVATE int `vrna_BT_mb_loop` (`vrna_fold_compound` *vc, int *i, int *j, int *k, int en, int *component1, int *component2)
Backtrack the decomposition of a multi branch loop closed by (i, j).

13.30.1 Detailed Description

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

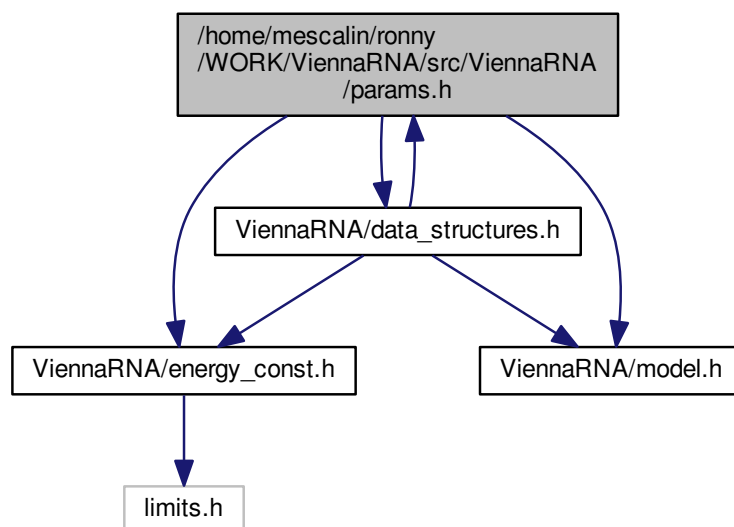
13.31 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/naview.h File Reference

This graph shows which files directly or indirectly include this file:



13.32 /home/mescal/ronny/WORK/ViennaRNA/src/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_t](#)
The datastructure that contains temperature scaled energy parameters.
- struct [vrna_exp_param_t](#)
The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.

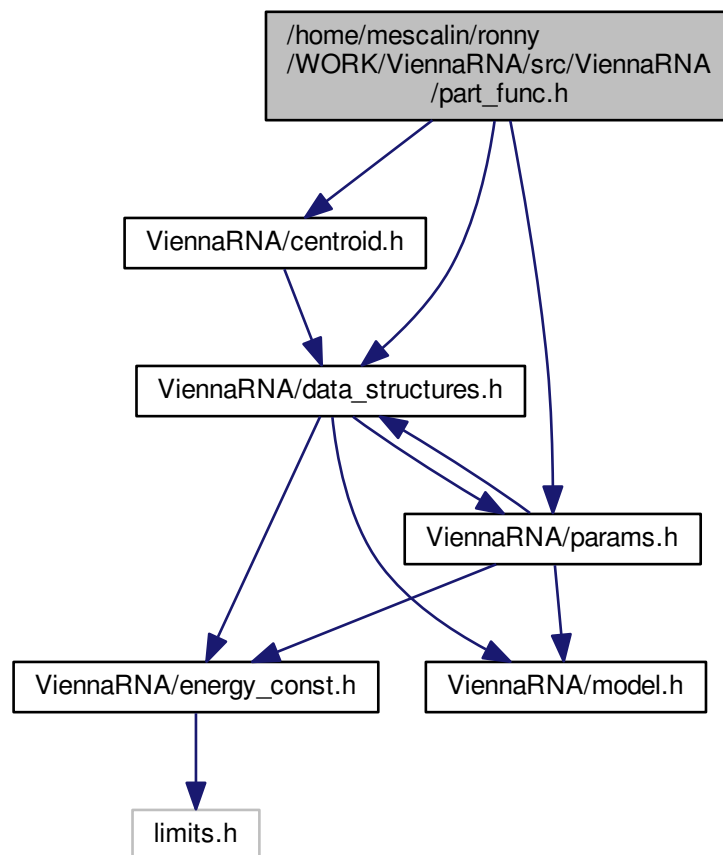
Functions

- [vrna_param_t * vrna_params_get](#) ([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_get](#) ([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_alif_get](#) (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)
- [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_alif](#) (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.

13.33 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func.h File Reference

Partition function of single RNA sequences.

Include dependency graph for `part_func.h`:



Functions

- float `vrna_pf_fold` (`vrna_fold_compound` *vc, char *structure)
Compute the partition function Q for a given RNA sequence.
- char * `vrna_pbacktrack5` (`vrna_fold_compound` *vc, int length)
Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.
- char * `vrna_pbacktrack` (`vrna_fold_compound` *vc)
Sample a secondary structure from the Boltzmann ensemble according its probability.
- 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` *vc)
Get the mean base pair distance in the thermodynamic ensemble.
- plist * `vrna_stack_prob` (`vrna_fold_compound` *vc, double cutoff)
Compute stacking probabilities.
- 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 relevant 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.*
- [plist](#) * [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 auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.*

13.33.1 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...

13.33.2 Function Documentation

13.33.2.1 [plist](#)* [stackProb](#) (double *cutoff*)

Get the probability of stacks.

Deprecated Use [vrna_stack_prob\(\)](#) instead!

13.33.2.2 void init_pf_fold (int *length*)

Allocate space for [pf_fold\(\)](#)

Deprecated This function is obsolete and will be removed soon!

13.33.2.3 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\(\)](#)

13.33.2.4 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_get_centroid_struct\(\)](#), [vrna_get_centroid_struct_pr\(\)](#), [vrna_get_centroid_struct_pl\(\)](#)

13.33.2.5 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!

13.33.2.6 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

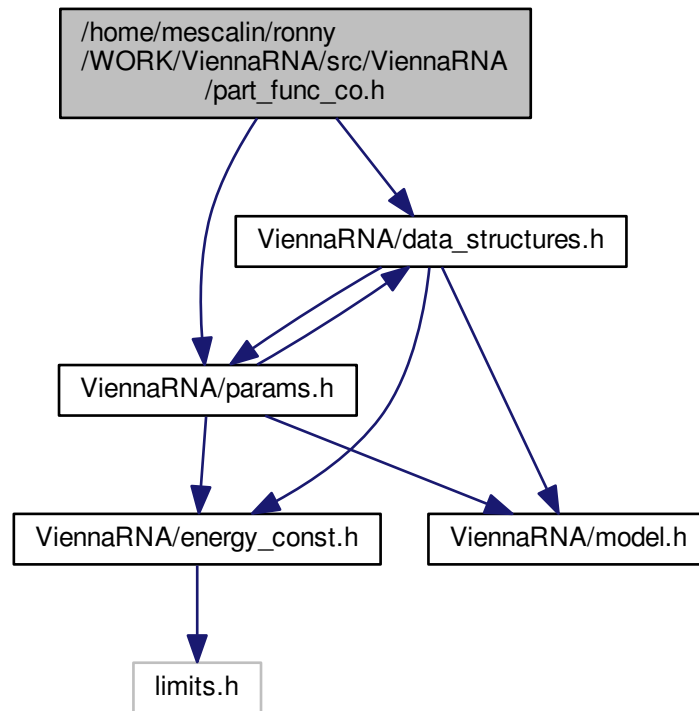
13.33.2.7 double expHairpinEnergy (int *u*, int *type*, short *si1*, short *sj1*, const char * *string*)

Deprecated Use [exp_E_Hairpin\(\)](#) from [loop_energies.h](#) instead

13.34 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_co.h File Reference

Partition function for two RNA sequences.

Include dependency graph for part_func_co.h:



Functions

- `cofoldF vrna_co_pf_fold` (`vrna_fold_compound` *vc, char *structure)
Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.
- void `vrna_co_pf_dimer_probs` (double FAB, double FA, double FB, struct `plist` *prAB, const `plist` *prA, const `plist` *prB, int Alength, const `vrna_exp_param_t` *exp_params)
Compute Boltzmann probabilities of dimerization without homodimers.
- `ConcEnt` * `vrna_co_pf_get_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.
- `cofoldF co_pf_fold` (char *sequence, char *structure)
Calculate partition function and base pair probabilities.
- `cofoldF 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.
- `plist` * `get_plist` (struct `plist` *pl, int length, double cut_off)
- void `compute_probabilities` (double FAB, double FEA, double FEB, struct `plist` *prAB, struct `plist` *prA, struct `plist` *prB, int Alength)
Compute Boltzmann probabilities of dimerization without homodimers.
- `ConcEnt` * `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.

13.34.1 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 dimers 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)

13.34.2 Function Documentation

13.34.2.1 `cofoldF 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_co_pf_fold()` instead!}

Parameters

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Will hold the structure or constraints

Returns

[cofoldF](#) structure containing a set of energies needed for concentration computations.

13.34.2.2 **cofoldF** 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.

See also

[get_boltzmann_factors\(\)](#), [co_pf_fold\(\)](#)

Parameters

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Pointer to the structure constraint
<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
<i>calculate_bppm</i>	Switch to turn Base pair probability calculations on/off (0==off)
<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)

Returns

[cofoldF](#) structure containing a set of energies needed for concentration computations.

13.34.2.3 **plist*** get_plist (struct plist * *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!

13.34.2.4 **void** compute_probabilities (double *FAB*, double *FEA*, double *FEB*, struct plist * *prAB*, struct plist * *prA*, struct plist * *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_co_pf_dimer_probs\(\)](#) instead!}

Parameters

<i>FAB</i>	free energy of dimer AB
<i>FEA</i>	free energy of monomer A
<i>FEB</i>	free energy of monomer B
<i>prAB</i>	pair probabilities for dimer
<i>prA</i>	pair probabilities monomer
<i>prB</i>	pair probabilities monomer
<i>Alength</i>	Length of molecule A

13.34.2.5 **ConcEnt*** `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 `cofoldF` struct.

Deprecated { Use `vrna_co_pf_get_concentrations()` instead!}

Parameters

<i>FEAB</i>	Free energy of AB dimer (FcAB entry)
<i>FEAA</i>	Free energy of AA dimer (FcAB entry)
<i>FEBB</i>	Free energy of BB dimer (FcAB entry)
<i>FEA</i>	Free energy of monomer A
<i>FEB</i>	Free energy of monomer B
<i>startconc</i>	List of start concentrations [a0],[b0],[a1],[b1],...,[an],[bn],[0],[0]

Returns

`ConcEnt` array containing the equilibrium energies and start concentrations

13.34.2.6 `void init_co_pf_fold (int length)`

DO NOT USE THIS FUNCTION ANYMORE

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

13.34.2.7 **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];
```

See also

`vrna_get_iindx()`

Returns

A pointer to the base pair probability array

13.34.2.8 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_update\(\)](#) instead!

Parameters

<i>length</i>	Length of the current RNA sequence
---------------	------------------------------------

13.34.2.9 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. Its 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_update\(\)](#) instead!

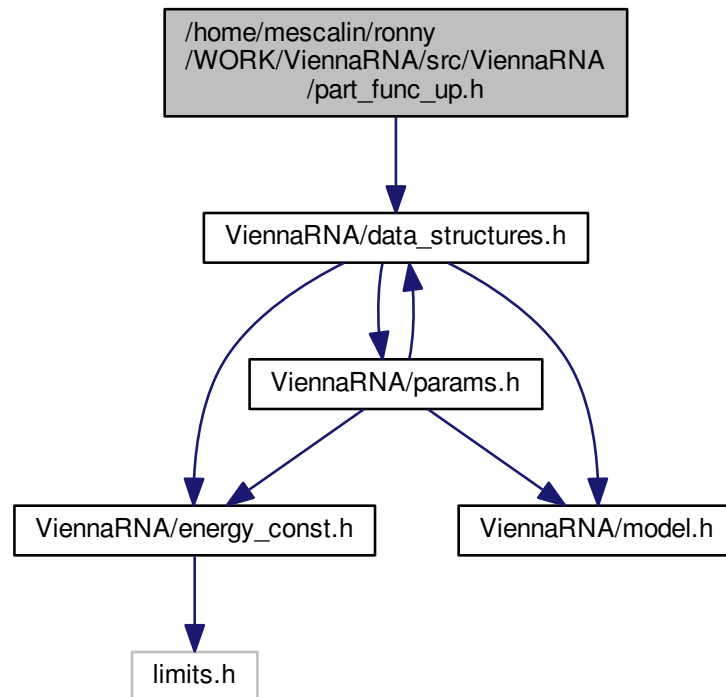
Parameters

<i>length</i>	Length of the current RNA sequence
<i>parameters</i>	data structure containing the precomputed Boltzmann factors

13.35 /home/mescalini/ronny/WORK/ViennaRNA/src/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()`.

13.35.1 Detailed Description

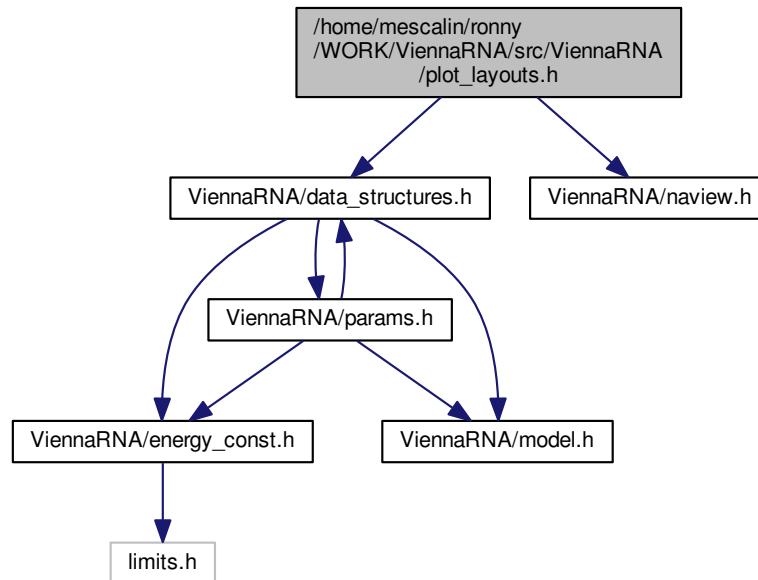
Partition Function Cofolding as stepwise process.

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.

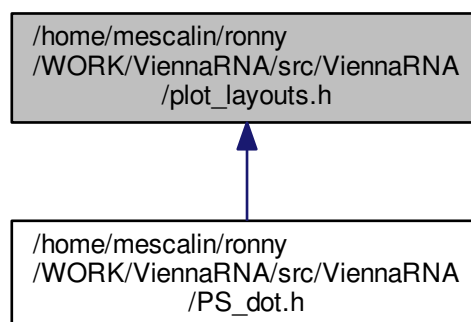
13.36 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/plot_layouts.h File Reference

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 workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

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.

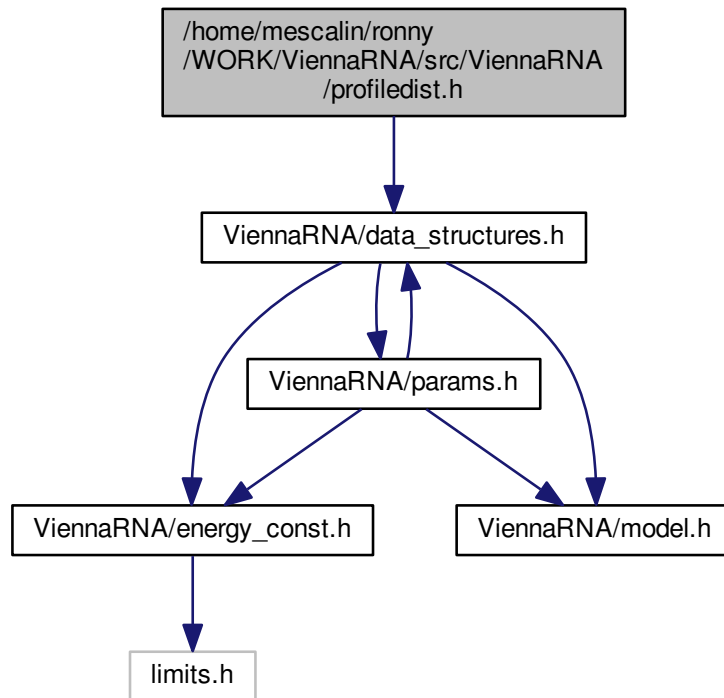
13.36.1 Detailed Description

Secondary structure plot layout algorithms.

c Ronny Lorenz The ViennaRNA Package

13.37 /home/mescalini/ronny/WORK/ViennaRNA/src/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)

13.37.1 Function Documentation

13.37.1.1 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

13.37.1.2 `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

<i>bppm</i>	A pointer to the base pair probability matrix
<i>length</i>	The length of the sequence

Returns

The bp profile

13.37.1.3 `void free_profile (float * T)`

free space allocated in `Make_bp_profile`

Backward compatibility only. You can just use plain `free()`

13.37.1.4 `float* Make_bp_profile (int length)`

Note

This function is NOT threadsafe

See also

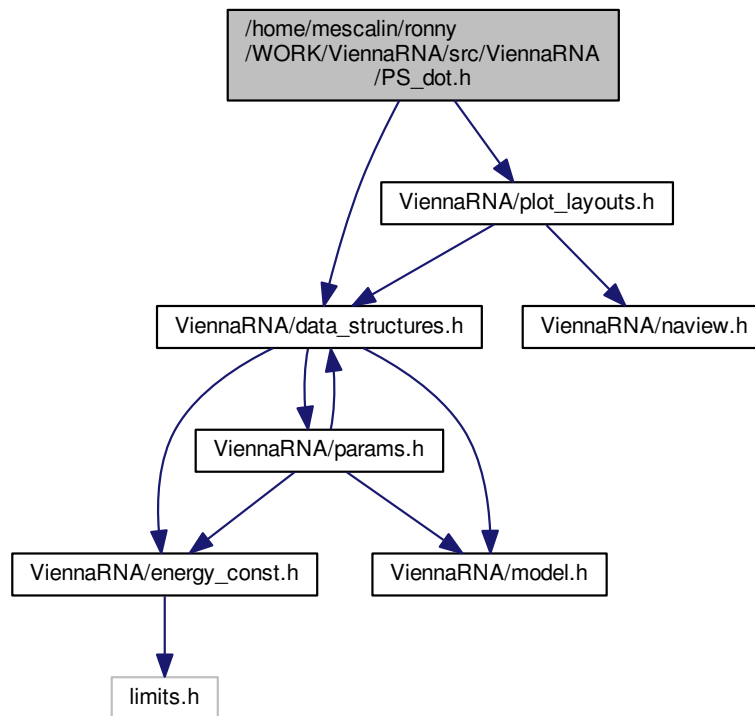
[Make_bp_profile_bppm\(\)](#)

Deprecated This function is deprecated and will be removed soon! See [Make_bp_profile_bppm\(\)](#) for a replacement

13.38 /home/mescalini/ronny/WORK/ViennaRNA/src/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_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 PS_dot_plot_list (char *seq, char *filename, plist *pl, plist *mf, char *comment)`
Produce a postscript dot-plot from two pair lists.
- `int aliPS_color_aln (const char *structure, const char *filename, const char *seqs[], const char *names[])`
- `int PS_dot_plot (char *string, char *file)`
Produce postscript dot-plot.

13.38.1 Detailed Description

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

13.39 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/read_epars.h File Reference

Functions

- void [read_parameter_file](#) (const char fname[])

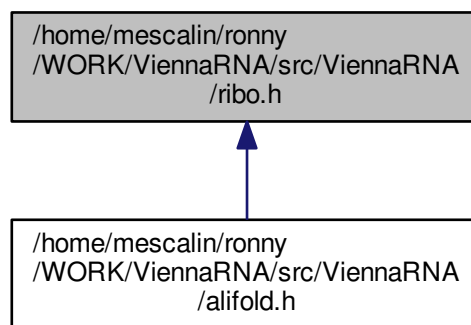
Read energy parameters from a file.
- void [write_parameter_file](#) (const char fname[])

Write energy parameters to a file.

13.40 /home/mescal/ronny/WORK/ViennaRNA/src/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.

13.40.1 Detailed Description

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

13.41 /home/mescal/ronny/WORK/ViennaRNA/src/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 *full)
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.

13.41.1 Detailed Description

Parsing and Coarse Graining of Structures.

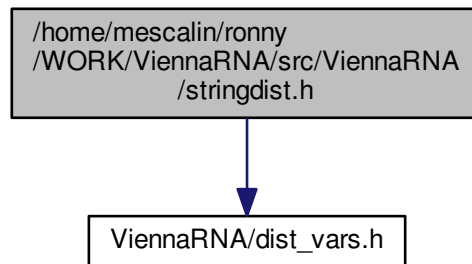
Example:

```
* .((..(((.....))..((.....))). 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)(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)
*
```

13.42 /home/mescal/ronny/WORK/ViennaRNA/src/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.

13.42.1 Detailed Description

Functions for String Alignment.

13.42.2 Function Documentation

13.42.2.1 `swString* Make_swString (char * string)`

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

Parameters

<i>string</i>	
---------------	--

Returns

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

Calculate the string edit distance of T1 and T2.

Parameters

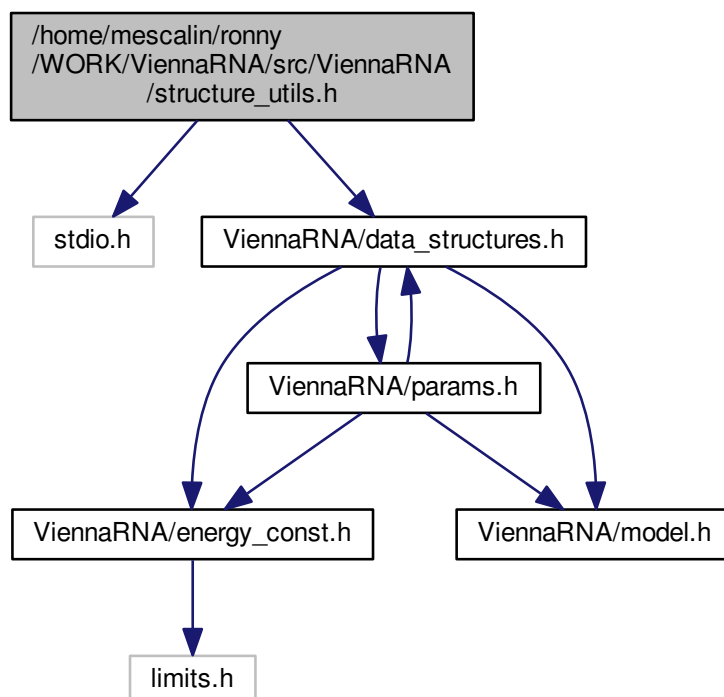
$T1$	
$T2$	

Returns

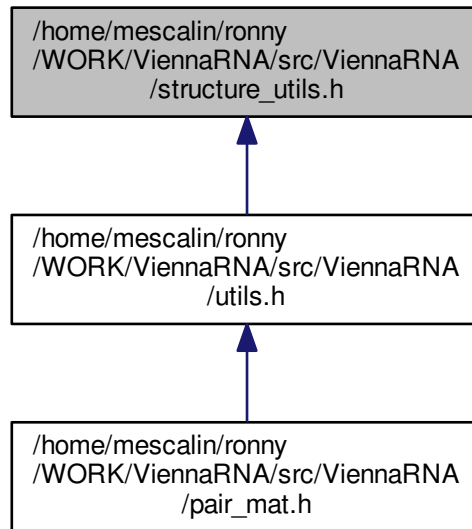
13.43 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/structure_utils.h File Reference

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_helix](#)

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_pt_get](#) (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_pt_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_get_loop_index](#) (const short *pt)
Get a loop index representation of a structure.
- char * [vrna_pt_to_db](#) (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_get_from_pr](#) (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.
- void [vrna_parenthesis_structure](#) (char *structure, [bondT](#) *bp, unsigned int length)
Create a dot-bracket/parenthesis structure from backtracking stack.
- void [vrna_parenthesis_zuker](#) (char *structure, [bondT](#) *bp, unsigned int length)
Create a dot-bracket/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.
- [plist](#) * [vrna_pl_get](#) (const char *struc, float pr)
Create a [plist](#) from a dot-bracket string.
- [plist](#) * [vrna_pl_get_from_pr](#) ([vrna_fold_compound](#) *vc, double cut_off)
Create a [plist](#) from base pair probability matrix.
- char * [vrna_pl_to_db](#) ([plist](#) *pairs, unsigned int n)
Convert a list of base pairs into dot-bracket notation.
- void [assign_plist_from_db](#) ([plist](#) **pl, const char *struc, float pr)
Create a [plist](#) from a dot-bracket string.
- char * [pack_structure](#) (const char *struc)
Pack 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](#) ([plist](#) **pl, FLT_OR_DBL *probs, int length, double cutoff)
Create a [plist](#) from a probability matrix.
- void [parenthesis_structure](#) (char *structure, [bondT](#) *bp, int length)
Create a dot-bracket/parenthesis structure from backtracking stack.
- void [parenthesis_zuker](#) (char *structure, [bondT](#) *bp, int length)
Create a dot-bracket/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.

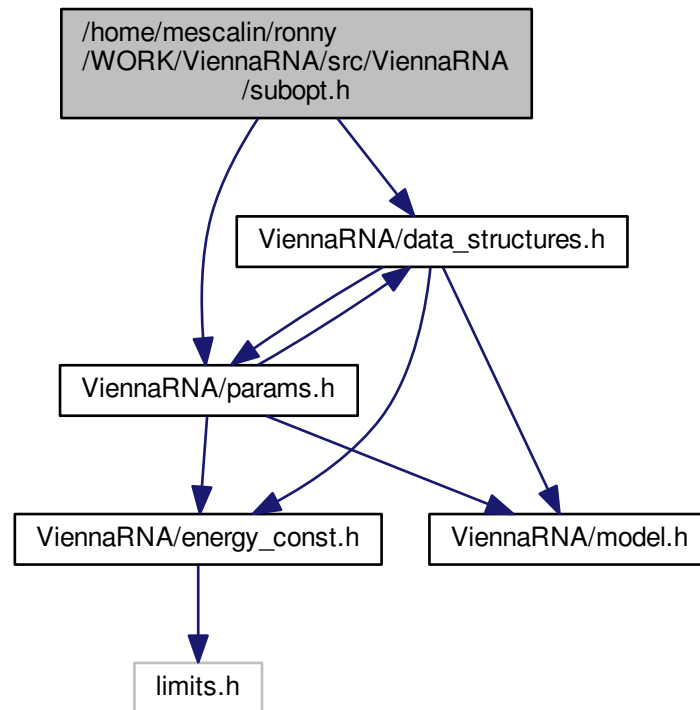
13.43.1 Detailed Description

Various utility- and helper-functions for secondary structure parsing, converting, etc.

13.44 /home/mescal/ronny/WORK/ViennaRNA/src/ViennaRNA/subopt.h File Reference

RNAsubopt and density of states declarations.

Include dependency graph for subopt.h:



Functions

- **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

- int `subopt_sorted`
Sort output by energy.
- double `print_energy`
printing threshold for use with logML
- int `density_of_states` [MAXDOS+1]
The Density of States.

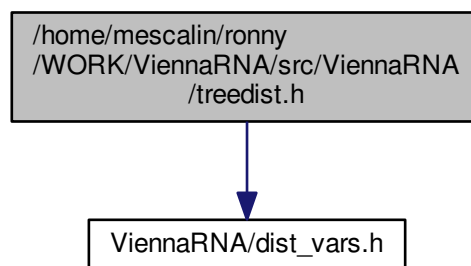
13.44.1 Detailed Description

RNAsubopt and density of states declarations.

13.45 /home/mescal/ronny/WORK/ViennaRNA/src/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.

13.45.1 Detailed Description

Functions for [Tree](#) Edit Distances.

13.45.2 Function Documentation

13.45.2.1 [Tree](#)* [make_tree](#) (char * struc)

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

Parameters

<i>struc</i>	may be any rooted structure representation.
--------------	---

Returns

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

Calculates the edit distance of the two trees.

Parameters

<i>T1</i>	
<i>T2</i>	

Returns

13.45.2.3 void free_tree (Tree * *t*)

Free the memory allocated for [Tree](#) *t*.

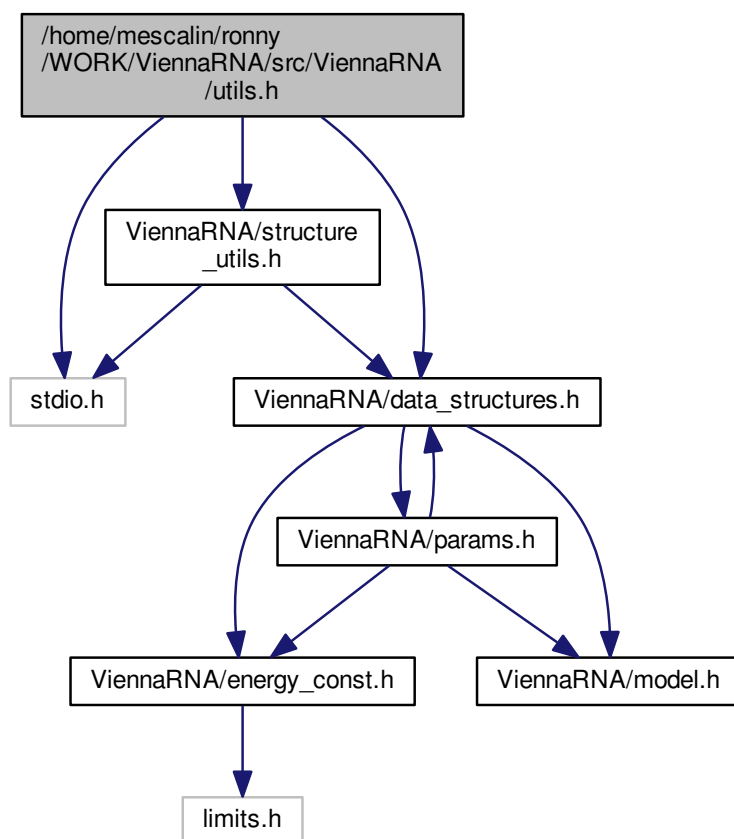
Parameters

<i>t</i>	
----------	--

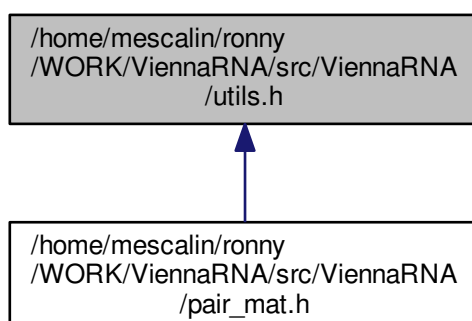
13.46 /home/mescalini/ronny/WORK/ViennaRNA/src/ViennaRNA/Utils.h File Reference

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 occurred, 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 truncate the line by eliminating white spaces at end of line".
- `#define VRNA_INPUT_NO_REST 512U`
Input switch for `vrna_read_fasta_record()`: "do fill rest array".
- `#define VRNA_INPUT_NO_SPAN 1024U`
Input switch for `vrna_read_fasta_record()`: "never allow data to span more than one line".
- `#define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U`
Input switch for `vrna_read_fasta_record()`: "do not skip empty lines".
- `#define VRNA_INPUT_BLANK_LINE 4096U`
Output flag for `vrna_read_fasta_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_read_fasta_record()`: "read a comment".
- `#define VRNA_OPTION_MULTILINE 32U`
Tell a function that an input is assumed to span several lines.
- `#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.
- `#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

- 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 * [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.
- 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.
- 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.
- int * [vrna_get_iindx](#) (unsigned int length)
Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.
- int * [vrna_get_indx](#) (unsigned int length)
Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.
- 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.
- char * [vrna_get_ptypes](#) (const short *S, [vrna_md_t](#) *md)
Get an array of the numerical encoding for each possible base pair (i,j)

- void [str_uppercase](#) (char *sequence)
Convert an input sequence to uppercase.
- void [str_DNA2RNA](#) (char *sequence)
Convert a DNA input sequence to RNA alphabet.
- 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.
- char * [random_string](#) (int l, const char symbols[])
Create a random string using characters from a specified symbol set.
- void [filecopy](#) (FILE *from, FILE *to)
Inefficient `cp`
- char * [time_stamp](#) (void)
Get a timestamp.
- 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.

Variables

- unsigned short [xsubi](#) [3]
Current 48 bit random number.

13.46.1 Detailed Description

General utility- and helper-functions used throughout the *ViennaRNA Package*.

13.46.2 Function Documentation

13.46.2.1 void str_uppercase (char * sequence)

Convert an input sequence to uppercase.

Deprecated Use [vrna_seq_toupper\(\)](#) instead!

13.46.2.2 void str_DNA2RNA (char * *sequence*)

Convert a DNA input sequence to RNA alphabet.

Deprecated Use [vrna_seq_toRNA\(\)](#) instead!

13.46.2.3 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!

13.46.2.4 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!

13.46.2.5 void warn_user (const char *message*[])

Print a warning message.

Print a warning message to *stderr*

Deprecated Use [vrna_message_warning\(\)](#) instead!

13.46.2.6 void nerror (const char *message*[])

Die with an error message.

Deprecated Use [vrna_message_error\(\)](#) instead!

13.46.2.7 void* space (unsigned *size*)

Allocate space safely.

Deprecated Use [vrna_alloc\(\)](#) instead!

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

Reallocate space safely.

Deprecated Use [vrna_realloc\(\)](#) instead!

13.46.2.9 void init_rand (void)

Make random number seeds.

Deprecated Use [vrna_init_rand\(\)](#) instead!

13.46.2.10 double urn (void)

get a random number from [0..1]

Deprecated Use [vrna_urn\(\)](#) instead!

13.46.2.11 int int_urn (int *from*, int *to*)

Generates a pseudo random integer in a specified range.

Deprecated Use [vrna_int_urn\(\)](#) instead!

13.46.2.12 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!

13.46.2.13 void filecopy (FILE * *from*, FILE * *to*)

Inefficient `cp`

Deprecated Use [vrna_file_copy\(\)](#) instead!

13.46.2.14 char* time_stamp (void)

Get a timestamp.

Deprecated Use [vrna_time_stamp\(\)](#) instead!

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

Calculate hamming distance between two sequences.

Deprecated Use [vrna_hamming_distance\(\)](#) instead!

13.46.2.16 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!

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. [120](#)
- [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. [115](#)
- [3] Katherine E. Deigan, Tian W. Li, David H. Mathews, and Kevin M. Weeks. Accurate SHAPE-directed RNA structure determination. *PNAS*, 106:97–102, 2009. [167](#)
- [4] 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 combinatorial landscapes. *Physical review E*, 47(3):2083, 1993. [3](#)
- [5] 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. [120](#)
- [6] 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. [1](#)
- [7] I.L. Hofacker and P.F. Stadler. Memory efficient folding algorithms for circular RNA secondary structures. *Bioinformatics*, 22(10):1172–1176, 2006. [85](#)
- [8] 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. [1](#)
- [9] 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. [145](#)
- [10] 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. [138](#)
- [11] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. [90](#)
- [12] B.A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer applications in the biosciences: CABIOS*, 4(3):387–393, 1988. [3](#)
- [13] 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](#)
- [14] 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. [138](#)
- [15] 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. [169](#)
- [16] 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. [167](#)

- [17] 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. [85](#)

Index

/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_epars.h, [239](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/1.8.4_intloops.h, [239](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dfold.h, [240](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/2Dpfold.h, [241](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/LPfold.h, [273](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/Lfold.h, [272](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/MEA.h, [275](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/PS_dot.h, [294](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/RNAstruct.h, [296](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/alifold.h, [244](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/centroid.h, [247](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/cofold.h, [248](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/convert_epars.h, [250](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/data_structures.h, [251](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/dist_vars.h, [253](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/duplex.h, [255](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/edit_cost.h, [255](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/energy_const.h, [256](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/eval.h, [257](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/energy_const.h, [256](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/eval.h, [257](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/exterior_loops.h, [260](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/file_formats.h, [261](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/findpath.h, [262](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/fold.h, [263](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/fold_vars.h, [265](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/gquad.h, [268](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/hairpin_loops.h, [269](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/interior_loops.h, [270](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/inverse.h, [272](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/loop_energies.h, [272](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/mm.h, [276](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/model.h, [276](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/multibranch_loops.h, [279](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/naview.h, [280](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/params.h, [280](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func.h, [281](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_co.h, [284](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/part_func_up.h, [289](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/plot_layouts.h, [291](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/profiledist.h, [293](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/read_epars.h, [296](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/ribo.h, [296](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/stringdist.h, [298](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/structure_utils.h, [299](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/subopt.h, [302](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/treedist.h, [303](#)
/home/mescalin/ronny/WORK/ViennaRNA/src/ViennaRNA/Utils.h, [304](#)
2Dpfold.h
 destroy_TwoDpfold_variables, [242](#)
 get_TwoDpfold_variables, [242](#)
 TwoDpfold_pbacktrack, [243](#)
 TwoDpfold_pbacktrack5, [244](#)

- TwoDpfoldList, [243](#)
- add_root
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [189](#)
- aliLfold
 - Local MFE consensus structures for Sequence Alignments, [137](#)
- aliPS_color_aln
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [208](#)
- alifold
 - MFE Consensus Structures for Sequence Alignment(s), [125](#)
- alifold.h
 - vrna_aln_get_pair_info, [246](#)
- alimake_pair_table
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [196](#)
- alipbacktrack
 - Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), [130](#)
- alipf_circ_fold
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [128](#)
- alipf_fold
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [128](#)
- alipf_fold_par
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [128](#)
- alloc_sequence_arrays
 - Predicting Consensus Structures from Alignment(s), [122](#)
- alpha
 - vrna_exp_param_t, [221](#)
- assign_plist_from_db
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [194](#)
- assign_plist_from_pr
 - Computing Partition Functions and Pair Probabilities, [97](#)
- b2C
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [188](#)
- b2HIT
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [188](#)
- b2Shapiro
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [189](#)
- BONUS
 - energy_const.h, [257](#)
- backtrack_GQuad_IntLoop
 - Processing and Evaluating Decomposed Loops, [58](#)
- backtrack_GQuad_IntLoop_L
 - Processing and Evaluating Decomposed Loops, [58](#)
- backtrack_type
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- base_pair
 - fold_vars.h, [267](#)
- Basic Data Structures for Structure Prediction and Evaluation, [173](#)
 - backtrack_type, [184](#)
 - canonicalBPonly, [184](#)
 - dangles, [183](#)
 - do_backtrack, [184](#)
 - energy_set, [184](#)
 - max_bp_span, [185](#)
 - noLonelyPairs, [184](#)
 - nonstandards, [184](#)
 - pf_scale, [183](#)
 - temperature, [183](#)
 - tetra_loop, [184](#)
 - VRNA_MODEL_DEFAULT_ALI_CV_FACT, [179](#)
 - VRNA_MODEL_DEFAULT_ALI_NC_FACT, [179](#)
 - VRNA_MODEL_DEFAULT_ALI_OLD_EN, [179](#)
 - VRNA_MODEL_DEFAULT_ALI_RIBO, [179](#)
 - VRNA_MODEL_DEFAULT_BACKTRACK, [178](#)
 - VRNA_MODEL_DEFAULT_BACKTRACK_TYPE, [178](#)
 - VRNA_MODEL_DEFAULT_BETA_SCALE, [177](#)
 - VRNA_MODEL_DEFAULT_CIRC, [178](#)
 - VRNA_MODEL_DEFAULT_COMPUTE_BPP, [178](#)
 - VRNA_MODEL_DEFAULT_DANGLES, [177](#)
 - VRNA_MODEL_DEFAULT_ENERGY_SET, [178](#)
 - VRNA_MODEL_DEFAULT_GQUAD, [178](#)
 - VRNA_MODEL_DEFAULT_LOG_ML, [179](#)
 - VRNA_MODEL_DEFAULT_MAX_BP_SPAN, [179](#)
 - VRNA_MODEL_DEFAULT_NO_GU, [177](#)
 - VRNA_MODEL_DEFAULT_NO_GU_CLOSURE, [177](#)
 - VRNA_MODEL_DEFAULT_NO_LP, [177](#)
 - VRNA_MODEL_DEFAULT_PF_SCALE, [177](#)
 - VRNA_MODEL_DEFAULT_SPECIAL_HP, [177](#)
 - VRNA_MODEL_DEFAULT_TEMPERATURE, [176](#)
 - VRNA_MODEL_DEFAULT_UNIQ_ML, [178](#)
 - VRNA_MX_2DFOLD, [180](#)
 - VRNA_MX_DEFAULT, [180](#)
 - VRNA_MX_LFOLD, [180](#)
 - VRNA_OPTION_EVAL_ONLY, [176](#)
 - VRNA_OPTION_MFE, [176](#)
 - VRNA_OPTION_PF, [176](#)
 - VRNA_VC_TYPE_ALIGNMENT, [180](#)
 - VRNA_VC_TYPE_SINGLE, [180](#)
 - vrna_free_fold_compound, [182](#)
 - vrna_free_mfe_matrices, [182](#)
 - vrna_free_pf_matrices, [182](#)
 - vrna_get_fold_compound, [180](#)
 - vrna_get_fold_compound_aln, [181](#)
 - vrna_md_set_default, [183](#)
 - vrna_md_set_globals, [183](#)
 - vrna_md_update, [183](#)
 - vrna_mx_t, [180](#)
 - vrna_params_update, [182](#)

- vrna_vc_t, [180](#)
- bondT, [209](#)
- bondTEn, [209](#)
- bp_distance
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [196](#)
- bppm_symbol
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [197](#)
- bppm_to_structure
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [197](#)
- bt
 - vrna_sc_t, [235](#)
- COORDINATE, [210](#)
- Calculate Partition Functions of a Distance Based Partitioning, [150](#)
 - vrna_TwoD_pf_fold, [150](#)
- Calculate Secondary Structures of two RNAs upon Dimerization, [109](#)
- Calculating MFE representatives of a Distance Based Partitioning, [146](#)
 - destroy_TwoDfold_variables, [148](#)
 - get_TwoDfold_variables, [148](#)
 - TwoDfold_backtrack_f5, [149](#)
 - TwoDfoldList, [148](#)
 - vrna_TwoD_backtrack5, [147](#)
 - vrna_TwoD_fold, [147](#)
- canonicalBPonly
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- centroid
 - part_func.h, [284](#)
- centroid.h
 - get_centroid_struct_pl, [248](#)
 - get_centroid_struct_pr, [248](#)
- circularfold
 - MFE Consensus Structures for Sequence Alignment(s), [125](#)
- circfold
 - Computing Minimum Free Energy (MFE) Structures, [86](#)
- Classified Dynamic Programming, [144](#)
- co_pf_fold
 - part_func_co.h, [286](#)
- co_pf_fold_par
 - part_func_co.h, [287](#)
- cofold
 - MFE Structures of two hybridized Sequences, [111](#)
- cofold.h
 - get_monomere_mfes, [250](#)
 - initialize_cofold, [250](#)
- cofold_par
 - MFE Structures of two hybridized Sequences, [111](#)
- cofoldF, [209](#)
- Compute the centroid structure, [99](#)
 - vrna_get_centroid_struct, [100](#)
 - vrna_get_centroid_struct_pl, [99](#)
 - vrna_get_centroid_struct_pr, [99](#)
- Compute the Density of States, [154](#)
 - density_of_states, [154](#)
- Compute the structure with maximum expected accuracy (MEA), [98](#)
- compute_BPdifferences
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [196](#)
- compute_probabilities
 - part_func_co.h, [287](#)
- Computing Minimum Free Energy (MFE) Structures, [84](#)
 - circfold, [86](#)
 - export_circfold_arrays, [87](#)
 - export_circfold_arrays_par, [88](#)
 - export_fold_arrays, [87](#)
 - export_fold_arrays_par, [87](#)
 - fold, [86](#)
 - fold_par, [85](#)
 - free_arrays, [87](#)
 - update_fold_params, [87](#)
 - update_fold_params_par, [87](#)
 - vrna_fold, [85](#)
- Computing Partition Functions and Pair Probabilities, [89](#)
 - assign_plist_from_pr, [97](#)
 - export_bppm, [95](#)
 - free_pf_arrays, [94](#)
 - get_pf_arrays, [95](#)
 - mean_bp_distance, [96](#)
 - mean_bp_distance_pr, [96](#)
 - pf_circ_fold, [94](#)
 - pf_fold, [93](#)
 - pf_fold_par, [92](#)
 - update_pf_params, [95](#)
 - update_pf_params_par, [95](#)
 - vrna_mean_bp_distance, [91](#)
 - vrna_mean_bp_distance_pr, [91](#)
 - vrna_pf_fold, [91](#)
 - vrna_pl_get_from_pr, [97](#)
 - vrna_stack_prob, [92](#)
- ConcEnt, [210](#)
- cons_seq
 - vrna_fold_compound, [226](#)
- constrain, [210](#)
- Constraining the Secondary Structure Recursions, [155](#)
 - VRNA_CONSTRAINT_DB, [157](#)
 - VRNA_CONSTRAINT_DB_ANG_BRACK, [156](#)
 - VRNA_CONSTRAINT_DB_DOT, [156](#)
 - VRNA_CONSTRAINT_DB_ENFORCE_BP, [157](#)
 - VRNA_CONSTRAINT_DB_GQUAD, [157](#)
 - VRNA_CONSTRAINT_DB_INTERMOL, [157](#)
 - VRNA_CONSTRAINT_DB_INTRAMOL, [157](#)
 - VRNA_CONSTRAINT_DB_PIPE, [156](#)
 - VRNA_CONSTRAINT_DB_RND_BRACK, [157](#)
 - VRNA_CONSTRAINT_DB_X, [156](#)
 - VRNA_CONSTRAINT_FILE, [158](#)
 - vrna_add_constraints, [158](#)
 - vrna_message_constraint_options, [158](#)
 - vrna_message_constraints_all, [158](#)

- convert_parameter_file
 - Converting Energy Parameter Files, [142](#)
- Converting Energy Parameter Files, [140](#)
 - convert_parameter_file, [142](#)
 - VRNA_CONVERT_OUTPUT_ALL, [141](#)
 - VRNA_CONVERT_OUTPUT_BULGE, [142](#)
 - VRNA_CONVERT_OUTPUT_DANGLE3, [141](#)
 - VRNA_CONVERT_OUTPUT_DANGLE5, [141](#)
 - VRNA_CONVERT_OUTPUT_DUMP, [142](#)
 - VRNA_CONVERT_OUTPUT_HP, [141](#)
 - VRNA_CONVERT_OUTPUT_INT, [142](#)
 - VRNA_CONVERT_OUTPUT_INT_11, [141](#)
 - VRNA_CONVERT_OUTPUT_INT_21, [141](#)
 - VRNA_CONVERT_OUTPUT_INT_22, [142](#)
 - VRNA_CONVERT_OUTPUT_MISC, [142](#)
 - VRNA_CONVERT_OUTPUT_ML, [142](#)
 - VRNA_CONVERT_OUTPUT_MM_EXT, [141](#)
 - VRNA_CONVERT_OUTPUT_MM_HP, [141](#)
 - VRNA_CONVERT_OUTPUT_MM_INT, [141](#)
 - VRNA_CONVERT_OUTPUT_MM_INT_1N, [141](#)
 - VRNA_CONVERT_OUTPUT_MM_INT_23, [141](#)
 - VRNA_CONVERT_OUTPUT_MM_MULTI, [141](#)
 - VRNA_CONVERT_OUTPUT_NINIO, [142](#)
 - VRNA_CONVERT_OUTPUT_SPECIAL_HP, [142](#)
 - VRNA_CONVERT_OUTPUT_STACK, [141](#)
 - VRNA_CONVERT_OUTPUT_VANILLA, [142](#)
- copy_pair_table
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [195](#)
- cost_matrix
 - dist_vars.h, [254](#)
- cpair, [210](#)
- cut_point
 - fold_vars.h, [267](#)
- cv_fact
 - Predicting Consensus Structures from Alignment(s), [123](#)
- dangles
 - Basic Data Structures for Structure Prediction and Evaluation, [183](#)
 - vrna_md_t, [229](#)
- Data Structures and Preprocessor Macros, [72](#)
- density_of_states
 - Compute the Density of States, [154](#)
- destroy_TwoDfold_variables
 - Calculating MFE representatives of a Distance Based Partitioning, [148](#)
- destroy_TwoDpfold_variables
 - 2Dpfold.h, [242](#)
- dist_vars.h
 - cost_matrix, [254](#)
 - edit_backtrack, [254](#)
- Distance based partitioning of the Secondary Structure Space, [145](#)
- do_backtrack
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- dupVar, [211](#)
- duplexT, [211](#)
- E_ExtLoop
 - Processing and Evaluating Decomposed Loops, [56](#)
- E_Hairpin
 - Processing and Evaluating Decomposed Loops, [59](#)
- E_IntLoop
 - Processing and Evaluating Decomposed Loops, [61](#)
- E_Stem
 - Processing and Evaluating Decomposed Loops, [56](#)
- E_mb_loop_stack
 - Processing and Evaluating Decomposed Loops, [62](#)
- edit_backtrack
 - dist_vars.h, [254](#)
- encode_ali_sequence
 - Predicting Consensus Structures from Alignment(s), [122](#)
- Energy Parameter Sets and Boltzmann Factors, [65](#)
 - get_boltzmann_factor_copy, [70](#)
 - get_boltzmann_factors, [69](#)
 - get_boltzmann_factors_ali, [70](#)
 - get_scaled_alipf_parameters, [70](#)
 - get_scaled_parameters, [70](#)
 - get_scaled_pf_parameters, [69](#)
 - scale_parameters, [70](#)
 - vrna_exp_params_ali_get, [68](#)
 - vrna_exp_params_copy, [68](#)
 - vrna_exp_params_get, [68](#)
 - vrna_exp_params_rescale, [66](#)
 - vrna_exp_params_update, [66](#)
 - vrna_params_copy, [67](#)
 - vrna_params_get, [67](#)
- energy_const.h
 - BONUS, [257](#)
 - FORBIDDEN, [257](#)
 - GASCONST, [256](#)
 - INF, [256](#)
 - K0, [256](#)
 - MAXLOOP, [257](#)
 - NBPAIRS, [257](#)
 - TURN, [257](#)
- energy_of_alistruct
 - Predicting Consensus Structures from Alignment(s), [120](#)
- energy_of_circ_struct
 - Free Energy Evaluation for given Sequence / Structure Pairs, [53](#)
- energy_of_circ_struct_par
 - Free Energy Evaluation for given Sequence / Structure Pairs, [50](#)
- energy_of_circ_structure
 - Free Energy Evaluation for given Sequence / Structure Pairs, [49](#)
- energy_of_move
 - Free Energy Evaluation for given Sequence / Structure Pairs, [51](#)
- energy_of_move_pt
 - Free Energy Evaluation for given Sequence / Structure Pairs, [52](#)

- energy_of_struct
 - Free Energy Evaluation for given Sequence / Structure Pairs, [52](#)
- energy_of_struct_par
 - Free Energy Evaluation for given Sequence / Structure Pairs, [49](#)
- energy_of_struct_pt
 - Free Energy Evaluation for given Sequence / Structure Pairs, [53](#)
- energy_of_struct_pt_par
 - Free Energy Evaluation for given Sequence / Structure Pairs, [51](#)
- energy_of_structure
 - Free Energy Evaluation for given Sequence / Structure Pairs, [48](#)
- energy_of_structure_pt
 - Free Energy Evaluation for given Sequence / Structure Pairs, [50](#)
- energy_set
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- Enumerating Suboptimal Structures, [101](#)
- exp_E_ExtLoop
 - Processing and Evaluating Decomposed Loops, [56](#)
- exp_E_Hairpin
 - Processing and Evaluating Decomposed Loops, [60](#)
- exp_E_IntLoop
 - Processing and Evaluating Decomposed Loops, [62](#)
- exp_E_Stem
 - Processing and Evaluating Decomposed Loops, [57](#)
- exp_f
 - vrna_sc_t, [235](#)
- expHairpinEnergy
 - part_func.h, [284](#)
- expLoopEnergy
 - part_func.h, [284](#)
- expand_Full
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [189](#)
- expand_Shapiro
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [189](#)
- export_al_i_bppm
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [129](#)
- export_bppm
 - Computing Partition Functions and Pair Probabilities, [95](#)
- export_circfold_arrays
 - Computing Minimum Free Energy (MFE) Structures, [87](#)
- export_circfold_arrays_par
 - Computing Minimum Free Energy (MFE) Structures, [88](#)
- export_co_bppm
 - part_func_co.h, [288](#)
- export_cofold_arrays
 - MFE Structures of two hybridized Sequences, [113](#)
- export_cofold_arrays_gq
 - MFE Structures of two hybridized Sequences, [112](#)
- export_fold_arrays
 - Computing Minimum Free Energy (MFE) Structures, [87](#)
- export_fold_arrays_par
 - Computing Minimum Free Energy (MFE) Structures, [87](#)
- f
 - vrna_sc_t, [235](#)
- FILENAME_ID_LENGTH
 - Utilities, [77](#)
- FILENAME_MAX_LENGTH
 - Utilities, [77](#)
- FORBIDDEN
 - energy_const.h, [257](#)
- filecopy
 - utils.h, [310](#)
- final_cost
 - Inverse Secondary Structure Prediction, [41](#)
- find_saddle
 - findpath.h, [263](#)
- findpath.h
 - find_saddle, [263](#)
 - free_path, [263](#)
 - get_path, [263](#)
- fold
 - Computing Minimum Free Energy (MFE) Structures, [86](#)
- fold.h
 - HairpinE, [265](#)
 - initialize_fold, [265](#)
 - LoopEnergy, [265](#)
- fold_par
 - Computing Minimum Free Energy (MFE) Structures, [85](#)
- fold_vars.h
 - base_pair, [267](#)
 - cut_point, [267](#)
 - iindx, [267](#)
 - james_rule, [267](#)
 - logML, [267](#)
 - pr, [267](#)
 - RibosumFile, [267](#)
- folden, [211](#)
- Free Energy Evaluation for given Sequence / Structure Pairs, [42](#)
 - energy_of_circ_struct, [53](#)
 - energy_of_circ_struct_par, [50](#)
 - energy_of_circ_structure, [49](#)
 - energy_of_move, [51](#)
 - energy_of_move_pt, [52](#)
 - energy_of_struct, [52](#)
 - energy_of_struct_par, [49](#)
 - energy_of_struct_pt, [53](#)
 - energy_of_struct_pt_par, [51](#)
 - energy_of_structure, [48](#)
 - energy_of_structure_pt, [50](#)

- loop_energy, 52
- vrna_eval_covar_structure, 43
- vrna_eval_hp_loop, 54
- vrna_eval_loop_pt, 47
- vrna_eval_move, 47
- vrna_eval_move_pt, 48
- vrna_eval_structure, 43
- vrna_eval_structure_pt, 45
- vrna_eval_structure_pt_simple, 46
- vrna_eval_structure_pt_simple_verbose, 47
- vrna_eval_structure_pt_verbose, 46
- vrna_eval_structure_simple, 44
- vrna_eval_structure_simple_verbose, 45
- vrna_eval_structure_verbose, 44
- free_alifold_arrays
 - MFE Consensus Structures for Sequence Alignment(s), 125
- free_alipf_arrays
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), 129
- free_arrays
 - Computing Minimum Free Energy (MFE) Structures, 87
- free_co_arrays
 - MFE Structures of two hybridized Sequences, 112
- free_path
 - findpath.h, 263
- free_pf_arrays
 - Computing Partition Functions and Pair Probabilities, 94
- free_profile
 - profiledist.h, 294
- free_sequence_arrays
 - Predicting Consensus Structures from Alignment(s), 123
- free_tree
 - treedist.h, 304
- Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 203
 - aliPS_color_aln, 208
 - gmlRNA, 206
 - PS_dot_plot, 208
 - PS_dot_plot_list, 207
 - PS_rna_plot, 205
 - PS_rna_plot_a, 206
 - rna_plot_type, 208
 - simple_circplot_coordinates, 205
 - simple_xy_coordinates, 205
 - ssv_rna_plot, 206
 - svg_rna_plot, 207
 - VRNA_PLOT_TYPE_CIRCULAR, 204
 - VRNA_PLOT_TYPE_NAVIEW, 204
 - VRNA_PLOT_TYPE_SIMPLE, 204
 - xrna_plot, 207
- Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 198
 - read_record, 202
 - vrna_extract_record_rest_constraint, 201
 - vrna_read_SHAPE_file, 202
 - vrna_read_constraints_file, 202
 - vrna_read_fasta_record, 200
 - vrna_structure_print_bpseq, 200
 - vrna_structure_print_ct, 200
 - vrna_structure_print_hx, 199
- GASCONST
 - energy_const.h, 256
- Generalized Soft Constraints, 170
 - VRNA_SC_GEN_MFE, 170
 - VRNA_SC_GEN_PF, 171
 - vrna_sc_add_bt, 171
 - vrna_sc_add_exp_f, 171
 - vrna_sc_add_f, 171
 - vrna_sc_add_post, 172
 - vrna_sc_add_pre, 172
- get_TwoDfold_variables
 - Calculating MFE representatives of a Distance Based Partitioning, 148
- get_TwoDpfold_variables
 - 2Dpfold.h, 242
- get_alipf_arrays
 - Predicting Consensus Structures from Alignment(s), 120
- get_boltzmann_factor_copy
 - Energy Parameter Sets and Boltzmann Factors, 70
- get_boltzmann_factors
 - Energy Parameter Sets and Boltzmann Factors, 69
- get_boltzmann_factors_ali
 - Energy Parameter Sets and Boltzmann Factors, 70
- get_centroid_struct_gquad_pr
 - part_func.h, 284
- get_centroid_struct_pl
 - centroid.h, 248
- get_centroid_struct_pr
 - centroid.h, 248
- get_concentrations
 - part_func_co.h, 288
- get_gquad_matrix
 - Processing and Evaluating Decomposed Loops, 57
- get_input_line
 - Utilities, 80
- get_line
 - Utilities, 79
- get_monomere_mfes
 - cofold.h, 250
- get_mpi
 - Predicting Consensus Structures from Alignment(s), 121
- get_path
 - findpath.h, 263
- get_pf_arrays
 - Computing Partition Functions and Pair Probabilities, 95
- get_plist
 - part_func_co.h, 287
- get_scaled_alipf_parameters
 - Energy Parameter Sets and Boltzmann Factors, 70

- get_scaled_parameters
 - Energy Parameter Sets and Boltzmann Factors, 70
- get_scaled_pf_parameters
 - Energy Parameter Sets and Boltzmann Factors, 69
- give_up
 - Inverse Secondary Structure Prediction, 41
- gmIRNA
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 206
- HairpinE
 - fold.h, 265
- hamming
 - utils.h, 310
- hamming_bound
 - utils.h, 310
- Hard Constraints, 160
 - vrna_hc_add_bp, 161
 - vrna_hc_add_bp_nonspecific, 162
 - vrna_hc_add_up, 161
 - vrna_hc_free, 162
 - vrna_hc_init, 161
- INF
 - energy_const.h, 256
- INTERVAL, 212
- id
 - vrna_exp_param_t, 221
- iindx
 - fold_vars.h, 267
- init_co_pf_fold
 - part_func_co.h, 288
- init_pf_fold
 - part_func.h, 283
- init_pf_foldLP
 - LPfold.h, 274
- init_rand
 - utils.h, 309
- initialize_cofold
 - cofold.h, 250
- initialize_fold
 - fold.h, 265
- int_urn
 - utils.h, 310
- interact, 211
- intermediate_t, 212
- inv_verbose
 - Inverse Secondary Structure Prediction, 41
- Inverse Secondary Structure Prediction, 40
 - final_cost, 41
 - give_up, 41
 - inv_verbose, 41
 - inverse_fold, 40
 - inverse_pf_fold, 40
- inverse_fold
 - Inverse Secondary Structure Prediction, 40
- inverse_pf_fold
 - Inverse Secondary Structure Prediction, 40
- james_rule
 - fold_vars.h, 267
- K0
 - energy_const.h, 256
- LIST, 213
- LPfold.h
 - init_pf_foldLP, 274
- LST_BUCKET, 213
- Lfold
 - Local MFE structure Prediction and Z-scores, 133
- Lfoldz
 - Local MFE structure Prediction and Z-scores, 133
- Local MFE consensus structures for Sequence Alignments, 137
 - aliLfold, 137
- Local MFE structure Prediction and Z-scores, 133
 - Lfold, 133
 - Lfoldz, 133
- logML
 - fold_vars.h, 267
- loop_energy
 - Free Energy Evaluation for given Sequence / Structure Pairs, 52
- LoopEnergy
 - fold.h, 265
- MAXLOOP
 - energy_const.h, 257
- MEA
 - MEA.h, 275
- MEA.h
 - MEA, 275
- MFE Consensus Structures for Sequence Alignment(s), 124
 - alifold, 125
 - circularifold, 125
 - free_alifold_arrays, 125
 - vrna_alifold, 124
- MFE Structures of two hybridized Sequences, 110
 - cofold, 111
 - cofold_par, 111
 - export_cofold_arrays, 113
 - export_cofold_arrays_gq, 112
 - free_co_arrays, 112
 - vrna_cofold, 111
 - vrna_cut_point_insert, 111
 - vrna_cut_point_remove, 112
- Make_bp_profile
 - profiledist.h, 294
- Make_bp_profile_bppm
 - profiledist.h, 294
- make_pair_table
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 195
- make_pair_table_snoop
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 196

- make_referenceBP_array
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [196](#)
- Make_swString
 - stringdist.h, [298](#)
- make_tree
 - treedist.h, [303](#)
- max_bp_span
 - Basic Data Structures for Structure Prediction and Evaluation, [185](#)
- mean_bp_dist
 - part_func.h, [284](#)
- mean_bp_distance
 - Computing Partition Functions and Pair Probabilities, [96](#)
- mean_bp_distance_pr
 - Computing Partition Functions and Pair Probabilities, [96](#)
- min_loop_size
 - vrna_md_t, [229](#)
- move_t, [213](#)
- n_seq
 - vrna_fold_compound, [225](#)
- NBPAIRS
 - energy_const.h, [257](#)
- nc_fact
 - Predicting Consensus Structures from Alignment(s), [123](#)
- noLonelyPairs
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- nonstandards
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- nrrerror
 - utils.h, [309](#)
- PAIR, [213](#)
- PS_dot_plot
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [208](#)
- PS_dot_plot_list
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [207](#)
- PS_rna_plot
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [205](#)
- PS_rna_plot_a
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [206](#)
- pack_structure
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [195](#)
- pair_info, [214](#)
- pairpro, [215](#)
- parenthesis_structure
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [197](#)
- parenthesis_zuker
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [197](#)
- parse_gquad
 - Processing and Evaluating Decomposed Loops, [58](#)
- parse_structure
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [191](#)
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [186](#)
 - add_root, [189](#)
 - alimake_pair_table, [196](#)
 - assign_plist_from_db, [194](#)
 - b2C, [188](#)
 - b2HIT, [188](#)
 - b2Shapiro, [189](#)
 - bp_distance, [196](#)
 - bppm_symbol, [197](#)
 - bppm_to_structure, [197](#)
 - compute_BPdifferences, [196](#)
 - copy_pair_table, [195](#)
 - expand_Full, [189](#)
 - expand_Shapiro, [189](#)
 - make_pair_table, [195](#)
 - make_pair_table_snoop, [196](#)
 - make_referenceBP_array, [196](#)
 - pack_structure, [195](#)
 - parenthesis_structure, [197](#)
 - parenthesis_zuker, [197](#)
 - parse_structure, [191](#)
 - unexpand_Full, [189](#)
 - unexpand_aligned_F, [191](#)
 - unpack_structure, [195](#)
 - unweight, [191](#)
 - vrna_bp_distance, [193](#)
 - vrna_db_pack, [191](#)
 - vrna_db_unpack, [192](#)
 - vrna_parenthesis_zuker, [193](#)
 - vrna_pl_get, [194](#)
 - vrna_pl_to_db, [194](#)
 - vrna_pt_copy, [192](#)
 - vrna_pt_get, [192](#)
 - vrna_pt_pk_get, [192](#)
 - vrna_pt_snoop_get, [193](#)
 - vrna_pt_to_db, [193](#)
 - vrna_refBPcnt_matrix, [193](#)
 - vrna_refBPdist_matrix, [193](#)
- part_func.h
 - centroid, [284](#)
 - expHairpinEnergy, [284](#)
 - expLoopEnergy, [284](#)
 - get_centroid_struct_gquad_pr, [284](#)
 - init_pf_fold, [283](#)
 - mean_bp_dist, [284](#)
 - stackProb, [283](#)
- part_func_co.h
 - co_pf_fold, [286](#)
 - co_pf_fold_par, [287](#)

- compute_probabilities, [287](#)
- export_co_bppm, [288](#)
- get_concentrations, [288](#)
- get_plist, [287](#)
- init_co_pf_fold, [288](#)
- update_co_pf_params, [288](#)
- update_co_pf_params_par, [289](#)
- Partition Function and Base Pair Probabilities for Sequence Alignment(s), [127](#)
 - alipf_circ_fold, [128](#)
 - alipf_fold, [128](#)
 - alipf_fold_par, [128](#)
 - export_ali_bppm, [129](#)
 - free_alipf_arrays, [129](#)
 - vrna_ali_pf_fold, [127](#)
- Partition Function for two hybridized Sequences, [114](#)
 - vrna_co_pf_dimer_probs, [115](#)
 - vrna_co_pf_fold, [115](#)
 - vrna_co_pf_get_concentrations, [115](#)
- Partition Function for two hybridized Sequences as a stepwise Process, [117](#)
 - pf_interact, [118](#)
 - pf_unstru, [117](#)
- Partition functions for locally stable secondary structures, [134](#)
 - pfl_fold, [134](#)
 - putoutpU_prob, [135](#)
 - putoutpU_prob_bin, [135](#)
 - update_pf_paramsLP, [134](#)
- path_t, [215](#)
- pbacktrack
 - Stochastic backtracking in the Ensemble, [107](#)
- pbacktrack_circ
 - Stochastic backtracking in the Ensemble, [107](#)
- pf_circ_fold
 - Computing Partition Functions and Pair Probabilities, [94](#)
- pf_fold
 - Computing Partition Functions and Pair Probabilities, [93](#)
- pf_fold_par
 - Computing Partition Functions and Pair Probabilities, [92](#)
- pf_interact
 - Partition Function for two hybridized Sequences as a stepwise Process, [118](#)
- pf_scale
 - Basic Data Structures for Structure Prediction and Evaluation, [183](#)
- pf_unstru
 - Partition Function for two hybridized Sequences as a stepwise Process, [117](#)
- pfl_fold
 - Partition functions for locally stable secondary structures, [134](#)
- plist, [215](#)
- post
 - vrna_sc_t, [236](#)
- Postorder_list, [215](#)
- pr
 - fold_vars.h, [267](#)
- pre
 - vrna_sc_t, [236](#)
- Predicting Consensus Structures from Alignment(s), [119](#)
 - alloc_sequence_arrays, [122](#)
 - cv_fact, [123](#)
 - encode_ali_sequence, [122](#)
 - energy_of_alistruct, [120](#)
 - free_sequence_arrays, [123](#)
 - get_alipf_arrays, [120](#)
 - get_mpi, [121](#)
 - nc_fact, [123](#)
 - update_alifold_params, [121](#)
 - vrna_ali_get_mpi, [121](#)
- Predicting Locally stable structures of large sequences, [132](#)
- print_tty_input_seq
 - utils.h, [309](#)
- print_tty_input_seq_str
 - utils.h, [309](#)
- Processing and Evaluating Decomposed Loops, [55](#)
 - backtrack_GQuad_IntLoop, [58](#)
 - backtrack_GQuad_IntLoop_L, [58](#)
 - E_ExtLoop, [56](#)
 - E_Hairpin, [59](#)
 - E_IntLoop, [61](#)
 - E_Stem, [56](#)
 - E_mb_loop_stack, [62](#)
 - exp_E_ExtLoop, [56](#)
 - exp_E_Hairpin, [60](#)
 - exp_E_IntLoop, [62](#)
 - exp_E_Stem, [57](#)
 - get_gquad_matrix, [57](#)
 - parse_gquad, [58](#)
 - vrna_BT_hp_loop, [61](#)
 - vrna_BT_mb_loop, [62](#)
 - vrna_E_hp_loop, [60](#)
 - vrna_exp_E_hp_loop, [60](#)
- profile_edit_distance
 - profiledist.h, [293](#)
- profiledist.h
 - free_profile, [294](#)
 - Make_bp_profile, [294](#)
 - Make_bp_profile_bppm, [294](#)
 - profile_edit_distance, [293](#)
- progress_callback
 - Soft Constraints, [165](#)
- pscore
 - vrna_fold_compound, [226](#)
- ptype
 - vrna_fold_compound, [224](#)
- ptype_pf_compat
 - vrna_fold_compound, [225](#)
- pu_contrib, [216](#)
- pu_out, [216](#)
- putoutpU_prob

- Partition functions for locally stable secondary structures, [135](#)
- putoutpU_prob_bin
 - Partition functions for locally stable secondary structures, [135](#)
- RNA Secondary Structure Prediction, [37](#)
- random_string
 - utils.h, [310](#)
- read_parameter_file
 - Reading/Writing Energy Parameter Sets from/to File, [138](#)
- read_record
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [202](#)
- Reading/Writing Energy Parameter Sets from/to File, [138](#)
 - read_parameter_file, [138](#)
 - write_parameter_file, [138](#)
- RibosumFile
 - fold_vars.h, [267](#)
- rna_plot_type
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [208](#)
- S
 - vrna_fold_compound, [226](#)
- S3
 - vrna_fold_compound, [226](#)
- S5
 - vrna_fold_compound, [226](#)
- S_cons
 - vrna_fold_compound, [226](#)
- SOLUTION, [217](#)
- sc
 - vrna_fold_compound, [225](#)
- scale_parameters
 - Energy Parameter Sets and Boltzmann Factors, [70](#)
- scs
 - vrna_fold_compound, [227](#)
- sect, [217](#)
- sequence
 - vrna_fold_compound, [224](#)
- sequence_encoding
 - vrna_fold_compound, [224](#)
- sequences
 - vrna_fold_compound, [225](#)
- simple_circplot_coordinates
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [205](#)
- simple_xy_coordinates
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [205](#)
- snoopT, [217](#)
- Soft Constraints, [163](#)
 - progress_callback, [165](#)
 - VRNA_MINIMIZER_CONJUGATE_FR, [164](#)
 - VRNA_MINIMIZER_CONJUGATE_PR, [164](#)
 - VRNA_MINIMIZER_STEEPEST_DESCENT, [165](#)
 - VRNA_MINIMIZER_VECTOR_BFGS, [165](#)
 - VRNA_MINIMIZER_VECTOR_BFGS2, [165](#)
 - VRNA_OBJECTIVE_FUNCTION_ABSOLUTE, [164](#)
 - VRNA_OBJECTIVE_FUNCTION_QUADRATIC, [164](#)
 - vrna_sc_SHAPE_add_deigan, [166](#)
 - vrna_sc_SHAPE_add_deigan_ali, [167](#)
 - vrna_sc_SHAPE_add_zarringhalam, [167](#)
 - vrna_sc_SHAPE_to_pr, [168](#)
 - vrna_sc_add_bp, [166](#)
 - vrna_sc_add_up, [166](#)
 - vrna_sc_free, [166](#)
 - vrna_sc_init, [165](#)
 - vrna_sc_minimize_perturbation, [168](#)
 - vrna_sc_remove, [166](#)
- space
 - utils.h, [309](#)
- ssv_rna_plot
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [206](#)
- st_back
 - Stochastic backtracking in the Ensemble, [108](#)
- stackProb
 - part_func.h, [283](#)
- Stochastic backtracking in the Ensemble, [106](#)
 - pbacktrack, [107](#)
 - pbacktrack_circ, [107](#)
 - st_back, [108](#)
 - vrna_pbacktrack, [107](#)
 - vrna_pbacktrack5, [106](#)
- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), [130](#)
 - alipbacktrack, [130](#)
 - vrna_ali_pbacktrack, [130](#)
- Stochastic Backtracking of Structures from Distance Based Partitioning, [152](#)
 - vrna_TwoD_pbacktrack, [152](#)
 - vrna_TwoD_pbacktrack5, [153](#)
- str_DNA2RNA
 - utils.h, [308](#)
- str_uppercase
 - utils.h, [308](#)
- string_edit_distance
 - stringdist.h, [298](#)
- stringdist.h
 - Make_swString, [298](#)
 - string_edit_distance, [298](#)
- struct_en, [217](#)
- subopt
 - Suboptimal structures within an energy band around the MFE, [104](#)
- subopt_circ
 - Suboptimal structures within an energy band around the MFE, [105](#)
- Suboptimal structures according to Zuker et al. 1989, [102](#)

- vrna_zukersubopt, [102](#)
- zukersubopt, [102](#)
- zukersubopt_par, [102](#)
- Suboptimal structures within an energy band around the MFE, [104](#)
- subopt, [104](#)
- subopt_circ, [105](#)
- svg_rna_plot
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [207](#)
- svm_model, [217](#)
- swString, [218](#)
- TURN
 - energy_const.h, [257](#)
- temperature
 - Basic Data Structures for Structure Prediction and Evaluation, [183](#)
- tetra_loop
 - Basic Data Structures for Structure Prediction and Evaluation, [184](#)
- time_stamp
 - utils.h, [310](#)
- Tree, [218](#)
- tree_edit_distance
 - treedist.h, [304](#)
- treedist.h
 - free_tree, [304](#)
 - make_tree, [303](#)
 - tree_edit_distance, [304](#)
- TwoDfold_backtrack_f5
 - Calculating MFE representatives of a Distance Based Partitioning, [149](#)
- TwoDfold_vars, [218](#)
- TwoDfoldList
 - Calculating MFE representatives of a Distance Based Partitioning, [148](#)
- TwoDpfold_pbacktrack
 - 2Dpfold.h, [243](#)
- TwoDpfold_pbacktrack5
 - 2Dpfold.h, [244](#)
- TwoDpfold_vars, [219](#)
- TwoDpfoldList
 - 2Dpfold.h, [243](#)
- type
 - vrna_fold_compound, [224](#)
- unexpand_Full
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [189](#)
- unexpand_aligned_F
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [191](#)
- unpack_structure
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [195](#)
- unweight
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [191](#)
- update_alifold_params
 - Predicting Consensus Structures from Alignment(s), [121](#)
- update_co_pf_params
 - part_func_co.h, [288](#)
- update_co_pf_params_par
 - part_func_co.h, [289](#)
- update_fold_params
 - Computing Minimum Free Energy (MFE) Structures, [87](#)
- update_fold_params_par
 - Computing Minimum Free Energy (MFE) Structures, [87](#)
- update_pf_params
 - Computing Partition Functions and Pair Probabilities, [95](#)
- update_pf_params_par
 - Computing Partition Functions and Pair Probabilities, [95](#)
- update_pf_paramsLP
 - Partition functions for locally stable secondary structures, [134](#)
- urn
 - utils.h, [310](#)
- Utilities, [74](#)
 - FILENAME_ID_LENGTH, [77](#)
 - FILENAME_MAX_LENGTH, [77](#)
 - get_input_line, [80](#)
 - get_line, [79](#)
 - VRNA_INPUT_CONSTRAINT, [76](#)
 - VRNA_INPUT_FASTA_HEADER, [76](#)
 - VRNA_OPTION_MULTILINE, [77](#)
 - vrna_alloc, [77](#)
 - vrna_get_iindx, [81](#)
 - vrna_get_indx, [81](#)
 - vrna_get_ptypes, [82](#)
 - vrna_hamming_distance, [79](#)
 - vrna_hamming_distance_bound, [79](#)
 - vrna_int_urn, [78](#)
 - vrna_message_error, [77](#)
 - vrna_message_input_seq, [80](#)
 - vrna_message_input_seq_simple, [80](#)
 - vrna_message_warning, [78](#)
 - vrna_nucleotide_decode, [82](#)
 - vrna_nucleotide_encode, [82](#)
 - vrna_random_string, [79](#)
 - vrna_realloc, [77](#)
 - vrna_seq_toRNA, [80](#)
 - vrna_seq_toupper, [81](#)
 - vrna_time_stamp, [78](#)
 - vrna_urn, [78](#)
 - xsubi, [83](#)
- utils.h
 - filecopy, [310](#)
 - hamming, [310](#)
 - hamming_bound, [310](#)
 - init_rand, [309](#)
 - int_urn, [310](#)

- nrerror, [309](#)
- print_tty_input_seq, [309](#)
- print_tty_input_seq_str, [309](#)
- random_string, [310](#)
- space, [309](#)
- str_DNA2RNA, [308](#)
- str_uppercase, [308](#)
- time_stamp, [310](#)
- urn, [310](#)
- warn_user, [309](#)
- xrealloc, [309](#)
- VRNA_CONSTRAINT_DB
 - Constraining the Secondary Structure Recursions, [157](#)
- VRNA_CONSTRAINT_DB_ANG_BRACK
 - Constraining the Secondary Structure Recursions, [156](#)
- VRNA_CONSTRAINT_DB_DOT
 - Constraining the Secondary Structure Recursions, [156](#)
- VRNA_CONSTRAINT_DB_ENFORCE_BP
 - Constraining the Secondary Structure Recursions, [157](#)
- VRNA_CONSTRAINT_DB_GQUAD
 - Constraining the Secondary Structure Recursions, [157](#)
- VRNA_CONSTRAINT_DB_INTERMOL
 - Constraining the Secondary Structure Recursions, [157](#)
- VRNA_CONSTRAINT_DB_INTRAMOL
 - Constraining the Secondary Structure Recursions, [157](#)
- VRNA_CONSTRAINT_DB_PIPE
 - Constraining the Secondary Structure Recursions, [156](#)
- VRNA_CONSTRAINT_DB_RND_BRACK
 - Constraining the Secondary Structure Recursions, [157](#)
- VRNA_CONSTRAINT_DB_X
 - Constraining the Secondary Structure Recursions, [156](#)
- VRNA_CONSTRAINT_FILE
 - Constraining the Secondary Structure Recursions, [158](#)
- VRNA_CONVERT_OUTPUT_ALL
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_BULGE
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_DANGLE3
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_DANGLE5
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_DUMP
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_HP
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_INT
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_INT_11
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_INT_21
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_INT_22
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_MISC
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_ML
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_MM_EXT
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_MM_HP
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_MM_INT
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_MM_INT_1N
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_MM_INT_23
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_MM_MULTI
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_NINIO
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_SPECIAL_HP
 - Converting Energy Parameter Files, [142](#)
- VRNA_CONVERT_OUTPUT_STACK
 - Converting Energy Parameter Files, [141](#)
- VRNA_CONVERT_OUTPUT_VANILLA
 - Converting Energy Parameter Files, [142](#)
- VRNA_INPUT_CONSTRAINT
 - Utilities, [76](#)
- VRNA_INPUT_FASTA_HEADER
 - Utilities, [76](#)
- VRNA_MINIMIZER_CONJUGATE_FR
 - Soft Constraints, [164](#)
- VRNA_MINIMIZER_CONJUGATE_PR
 - Soft Constraints, [164](#)
- VRNA_MINIMIZER_STEEPEST_DESCENT
 - Soft Constraints, [165](#)
- VRNA_MINIMIZER_VECTOR_BFGS
 - Soft Constraints, [165](#)
- VRNA_MINIMIZER_VECTOR_BFGS2
 - Soft Constraints, [165](#)
- VRNA_MODEL_DEFAULT_ALI_CV_FACT
 - Basic Data Structures for Structure Prediction and Evaluation, [179](#)
- VRNA_MODEL_DEFAULT_ALI_NC_FACT
 - Basic Data Structures for Structure Prediction and Evaluation, [179](#)
- VRNA_MODEL_DEFAULT_ALI_OLD_EN
 - Basic Data Structures for Structure Prediction and Evaluation, [179](#)
- VRNA_MODEL_DEFAULT_ALI_RIBO
 - Basic Data Structures for Structure Prediction and Evaluation, [179](#)
- VRNA_MODEL_DEFAULT_BACKTRACK

- Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MODEL_DEFAULT_BACKTRACK_TYPE
 - Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MODEL_DEFAULT_BETA_SCALE
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_CIRC
 - Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MODEL_DEFAULT_COMPUTE_BPP
 - Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MODEL_DEFAULT_DANGLES
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_ENERGY_SET
 - Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MODEL_DEFAULT_GQUAD
 - Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MODEL_DEFAULT_LOG_ML
 - Basic Data Structures for Structure Prediction and Evaluation, [179](#)
- VRNA_MODEL_DEFAULT_MAX_BP_SPAN
 - Basic Data Structures for Structure Prediction and Evaluation, [179](#)
- VRNA_MODEL_DEFAULT_NO_GU
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_NO_GU_CLOSURE
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_NO_LP
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_PF_SCALE
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_SPECIAL_HP
 - Basic Data Structures for Structure Prediction and Evaluation, [177](#)
- VRNA_MODEL_DEFAULT_TEMPERATURE
 - Basic Data Structures for Structure Prediction and Evaluation, [176](#)
- VRNA_MODEL_DEFAULT_UNIQ_ML
 - Basic Data Structures for Structure Prediction and Evaluation, [178](#)
- VRNA_MX_2DFOLD
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- VRNA_MX_DEFAULT
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- VRNA_MX_LFOLD
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- VRNA_OBJECTIVE_FUNCTION_ABSOLUTE
 - Soft Constraints, [164](#)
- VRNA_OBJECTIVE_FUNCTION_QUADRATIC
 - Soft Constraints, [164](#)
- VRNA_OPTION_EVAL_ONLY
 - Basic Data Structures for Structure Prediction and Evaluation, [176](#)
- VRNA_OPTION_MFE
 - Basic Data Structures for Structure Prediction and Evaluation, [176](#)
- VRNA_OPTION_MULTILINE
 - Utilities, [77](#)
- VRNA_OPTION_PF
 - Basic Data Structures for Structure Prediction and Evaluation, [176](#)
- VRNA_PLOT_TYPE_CIRCULAR
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [204](#)
- VRNA_PLOT_TYPE_NAVIEW
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [204](#)
- VRNA_PLOT_TYPE_SIMPLE
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [204](#)
- VRNA_SC_GEN_MFE
 - Generalized Soft Constraints, [170](#)
- VRNA_SC_GEN_PF
 - Generalized Soft Constraints, [171](#)
- VRNA_VC_TYPE_ALIGNMENT
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- VRNA_VC_TYPE_SINGLE
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- vrna_BT_hp_loop
 - Processing and Evaluating Decomposed Loops, [61](#)
- vrna_BT_mb_loop
 - Processing and Evaluating Decomposed Loops, [62](#)
- vrna_E_hp_loop
 - Processing and Evaluating Decomposed Loops, [60](#)
- vrna_TwoD_backtrack5
 - Calculating MFE representatives of a Distance Based Partitioning, [147](#)
- vrna_TwoD_fold
 - Calculating MFE representatives of a Distance Based Partitioning, [147](#)
- vrna_TwoD_pbacktrack
 - Stochastic Backtracking of Structures from Distance Based Partitioning, [152](#)
- vrna_TwoD_pbacktrack5
 - Stochastic Backtracking of Structures from Distance Based Partitioning, [153](#)
- vrna_TwoD_pf_fold
 - Calculate Partition Functions of a Distance Based Partitioning, [150](#)
- vrna_add_constraints

- Constraining the Secondary Structure Recursions, 158
- `vrna_ali_fold`
 - MFE Consensus Structures for Sequence Alignment(s), 124
- `vrna_ali_get_mpi`
 - Predicting Consensus Structures from Alignment(s), 121
- `vrna_ali_get_pair_info`
 - `alifold.h`, 246
- `vrna_ali_pbacktrack`
 - Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), 130
- `vrna_ali_pf_fold`
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), 127
- `vrna_alloc`
 - Utilities, 77
- `vrna_bp_distance`
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 193
- `vrna_co_pf_dimer_probs`
 - Partition Function for two hybridized Sequences, 115
- `vrna_co_pf_fold`
 - Partition Function for two hybridized Sequences, 115
- `vrna_co_pf_get_concentrations`
 - Partition Function for two hybridized Sequences, 115
- `vrna_cofold`
 - MFE Structures of two hybridized Sequences, 111
- `vrna_cut_point_insert`
 - MFE Structures of two hybridized Sequences, 111
- `vrna_cut_point_remove`
 - MFE Structures of two hybridized Sequences, 112
- `vrna_db_pack`
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 191
- `vrna_db_unpack`
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 192
- `vrna_eval_covar_structure`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 43
- `vrna_eval_hp_loop`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 54
- `vrna_eval_loop_pt`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 47
- `vrna_eval_move`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 47
- `vrna_eval_move_pt`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 48
- `vrna_eval_structure`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 43
- `vrna_eval_structure_pt`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 45
- `vrna_eval_structure_pt_simple`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 46
- `vrna_eval_structure_pt_simple_verbose`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 47
- `vrna_eval_structure_pt_verbose`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 46
- `vrna_eval_structure_simple`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 44
- `vrna_eval_structure_simple_verbose`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 45
- `vrna_eval_structure_verbose`
 - Free Energy Evaluation for given Sequence / Structure Pairs, 44
- `vrna_exp_E_hp_loop`
 - Processing and Evaluating Decomposed Loops, 60
- `vrna_exp_param_t`, 221
 - `alpha`, 221
 - `id`, 221
- `vrna_exp_params_ali_get`
 - Energy Parameter Sets and Boltzmann Factors, 68
- `vrna_exp_params_copy`
 - Energy Parameter Sets and Boltzmann Factors, 68
- `vrna_exp_params_get`
 - Energy Parameter Sets and Boltzmann Factors, 68
- `vrna_exp_params_rescale`
 - Energy Parameter Sets and Boltzmann Factors, 66
- `vrna_exp_params_update`
 - Energy Parameter Sets and Boltzmann Factors, 66
- `vrna_extract_record_rest_constraint`
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 201
- `vrna_fold`
 - Computing Minimum Free Energy (MFE) Structures, 85
- `vrna_fold_compound`, 222
 - `cons_seq`, 226
 - `n_seq`, 225
 - `pscore`, 226
 - `ptype`, 224
 - `ptype_pf_compat`, 225
 - `S`, 226
 - `S3`, 226
 - `S5`, 226
 - `S_cons`, 226
 - `sc`, 225
 - `scs`, 227
 - `sequence`, 224

- sequence_encoding, [224](#)
- sequences, [225](#)
- type, [224](#)
- vrna_free_fold_compound
 - Basic Data Structures for Structure Prediction and Evaluation, [182](#)
- vrna_free_mfe_matrices
 - Basic Data Structures for Structure Prediction and Evaluation, [182](#)
- vrna_free_pf_matrices
 - Basic Data Structures for Structure Prediction and Evaluation, [182](#)
- vrna_get_centroid_struct
 - Compute the centroid structure, [100](#)
- vrna_get_centroid_struct_pl
 - Compute the centroid structure, [99](#)
- vrna_get_centroid_struct_pr
 - Compute the centroid structure, [99](#)
- vrna_get_fold_compound
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- vrna_get_fold_compound_aln
 - Basic Data Structures for Structure Prediction and Evaluation, [181](#)
- vrna_get_iindx
 - Utilities, [81](#)
- vrna_get_indx
 - Utilities, [81](#)
- vrna_get_ptypes
 - Utilities, [82](#)
- vrna_hamming_distance
 - Utilities, [79](#)
- vrna_hamming_distance_bound
 - Utilities, [79](#)
- vrna_hc_add_bp
 - Hard Constraints, [161](#)
- vrna_hc_add_bp_nonspecific
 - Hard Constraints, [162](#)
- vrna_hc_add_up
 - Hard Constraints, [161](#)
- vrna_hc_free
 - Hard Constraints, [162](#)
- vrna_hc_init
 - Hard Constraints, [161](#)
- vrna_hc_t, [227](#)
- vrna_helix, [228](#)
- vrna_int_urn
 - Utilities, [78](#)
- vrna_md_set_default
 - Basic Data Structures for Structure Prediction and Evaluation, [183](#)
- vrna_md_set_globals
 - Basic Data Structures for Structure Prediction and Evaluation, [183](#)
- vrna_md_t, [228](#)
 - dangles, [229](#)
 - min_loop_size, [229](#)
- vrna_md_update
 - Basic Data Structures for Structure Prediction and Evaluation, [183](#)
- vrna_mean_bp_distance
 - Computing Partition Functions and Pair Probabilities, [91](#)
- vrna_mean_bp_distance_pr
 - Computing Partition Functions and Pair Probabilities, [91](#)
- vrna_message_constraint_options
 - Constraining the Secondary Structure Recursions, [158](#)
- vrna_message_constraints_all
 - Constraining the Secondary Structure Recursions, [158](#)
- vrna_message_error
 - Utilities, [77](#)
- vrna_message_input_seq
 - Utilities, [80](#)
- vrna_message_input_seq_simple
 - Utilities, [80](#)
- vrna_message_warning
 - Utilities, [78](#)
- vrna_mx_mfe_t, [230](#)
- vrna_mx_pf_t, [232](#)
- vrna_mx_t
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- vrna_nucleotide_decode
 - Utilities, [82](#)
- vrna_nucleotide_encode
 - Utilities, [82](#)
- vrna_param_t, [233](#)
- vrna_params_copy
 - Energy Parameter Sets and Boltzmann Factors, [67](#)
- vrna_params_get
 - Energy Parameter Sets and Boltzmann Factors, [67](#)
- vrna_params_update
 - Basic Data Structures for Structure Prediction and Evaluation, [182](#)
- vrna_parenthesis_zuker
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [193](#)
- vrna_pbacktrack
 - Stochastic backtracking in the Ensemble, [107](#)
- vrna_pbacktrack5
 - Stochastic backtracking in the Ensemble, [106](#)
- vrna_pf_fold
 - Computing Partition Functions and Pair Probabilities, [91](#)
- vrna_pl_get
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [194](#)
- vrna_pl_get_from_pr
 - Computing Partition Functions and Pair Probabilities, [97](#)
- vrna_pl_to_db
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [194](#)

- vrna_pt_copy
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [192](#)
- vrna_pt_get
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [192](#)
- vrna_pt_pk_get
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [192](#)
- vrna_pt_snoop_get
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [193](#)
- vrna_pt_to_db
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [193](#)
- vrna_random_string
 - Utilities, [79](#)
- vrna_read_SHAPE_file
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [202](#)
- vrna_read_constraints_file
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [202](#)
- vrna_read_fasta_record
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [200](#)
- vrna_realloc
 - Utilities, [77](#)
- vrna_refBPcnt_matrix
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [193](#)
- vrna_refBPdist_matrix
 - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [193](#)
- vrna_sc_SHAPE_add_deigan
 - Soft Constraints, [166](#)
- vrna_sc_SHAPE_add_deigan_al
 - Soft Constraints, [167](#)
- vrna_sc_SHAPE_add_zarringhalam
 - Soft Constraints, [167](#)
- vrna_sc_SHAPE_to_pr
 - Soft Constraints, [168](#)
- vrna_sc_add_bp
 - Soft Constraints, [166](#)
- vrna_sc_add_bt
 - Generalized Soft Constraints, [171](#)
- vrna_sc_add_exp_f
 - Generalized Soft Constraints, [171](#)
- vrna_sc_add_f
 - Generalized Soft Constraints, [171](#)
- vrna_sc_add_post
 - Generalized Soft Constraints, [172](#)
- vrna_sc_add_pre
 - Generalized Soft Constraints, [172](#)
- vrna_sc_add_up
 - Soft Constraints, [166](#)
- vrna_sc_free
 - Soft Constraints, [166](#)
- vrna_sc_init
 - Soft Constraints, [165](#)
- vrna_sc_minimize_perturbation
 - Soft Constraints, [168](#)
- vrna_sc_remove
 - Soft Constraints, [166](#)
- vrna_sc_t, [234](#)
 - bt, [235](#)
 - exp_f, [235](#)
 - f, [235](#)
 - post, [236](#)
 - pre, [236](#)
- vrna_seq_toRNA
 - Utilities, [80](#)
- vrna_seq_toupper
 - Utilities, [81](#)
- vrna_sol_TwoD_pf_t, [236](#)
- vrna_sol_TwoD_t, [237](#)
- vrna_stack_prob
 - Computing Partition Functions and Pair Probabilities, [92](#)
- vrna_structure_print_bpseq
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [200](#)
- vrna_structure_print_ct
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [200](#)
- vrna_structure_print_hx
 - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [199](#)
- vrna_time_stamp
 - Utilities, [78](#)
- vrna_urn
 - Utilities, [78](#)
- vrna_vc_t
 - Basic Data Structures for Structure Prediction and Evaluation, [180](#)
- vrna_zukersubopt
 - Suboptimal structures according to Zuker et al. 1989, [102](#)
- warn_user
 - utils.h, [309](#)
- write_parameter_file
 - Reading/Writing Energy Parameter Sets from/to File, [138](#)
- xrealloc
 - utils.h, [309](#)
- xrna_plot
 - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [207](#)
- xsubi

Utilities, [83](#)

zuckersubopt

Suboptimal structures according to Zuker et al.
1989, [102](#)

zuckersubopt_par

Suboptimal structures according to Zuker et al.
1989, [102](#)