Nucleic Acid Sequence Alignments of Partly Coding Regions

DISSERTATION
zur Erlangung des akademischen Grades
Doctor rerum naturalium

Vorgelegt der
Fakultät für Naturwissenschaften und Mathematik
der Universität Wien

von
Mag. Roman Stocsits

am Institut für
Theoretische Chemie und Molekulare Strukturbiologie

Februar 2003

Abstract

High quality sequence alignments of RNA and DNA sequences are a prerequisite for the comparative analysis of genomic sequence data. The high level of sequence heterogeneity, as compared to proteins, makes good alignments of nucleic acid sequences often impossible. In many cases, the nucleic acid sequences under consideration, or parts of them, code for proteins. While protein sequences can still show substantial homology, the corresponding nucleic acid sequences have already evolutionarily diverged, thus they are essentially randomized. This is caused by the inherent redundancy of the genetic code: Most amino acids have more than one codon on the level of nucleic acid. This specific problem leads to gaps and incorrectly aligned segments within coding regions.

In the thesis a multiple nucleic acid alignment procedure was implemented that uses genetic information about coding and non-coding regions as part of the scoring function in order to improve the resulting alignment. Our algorithm combines (mis)match scores for nucleic acids with those for the underlying amino acids in the case of open reading frames and exons. The program makes explicit use of information about overlapping open reading frames, as they occur in virus sequences, to further improve the reliability and quality of the nucleic acid alignment.

The implementation is realized in the program package `code2aln` which is freely available.

`Code2aln` is based upon a Gotoh-type dynamic programming algorithm with affine gap penalties, and features more complex scoring functions for coding regions that combine nucleic acid with amino acid scores.

Alignments computed with `code2aln` have a significantly improved quality in coding regions compared to other methods for nucleic acids. In particular, disruptions of codons are reduced. `Code2aln` alignments are shown to improve the sensitivity of a method for the detection of conserved RNA structures.

An application of `code2aln` to two unrelated groups of viruses is described. We processed the alignments as input for the procedure of `alidot` for detecting conserved RNA secondary structure elements in RNA genomes of Leviviridae and the pregenomic RNA of human hepatitis B virus. Virus genomes contain various (partially overlapping) open reading frames and are

an ideal test case for a procedure that makes usage of information about (overlapping) coding regions to improve the input alignments and, therefore, the identification of conserved secondary structures.

We find a number of highly significant secondary structure elements, not being described in the literature so far, and some well known elements like the ε-elements and two important elements of the HPRE region in hepatitis B virus. Also the results of the Levivirus group are of particular interest: We detect various secondary structure elements that are strongly confirmed by compensatory mutations and gain novel insight into the structural organization of Levivirus genomes.

Zusammenfassung

Nukleinsäuresequenz-Alignments hoher Qualität sind von großer Bedeutung für die vergleichende Analyse genomischer Sequenzdaten. Die höhere Sequenzheterogenität auf der Ebene von Nukleinsäuren, verglichen mit Proteinsequenzen, macht es oft unmöglich, gute Nukleinsäure-Alignments zu erreichen. In vielen Fällen codieren die betrachteten Sequenzen, oder Teile davon, für Proteine. Während Proteinsequenzen weitgehend Homologie zeigen, können die entsprechenden Nukleinsäuresequenzen evolutionäre Divergenz bis hin zur völligen Heterogenität zeigen. Der Grund ist die inhärente Redundanz des genetischen Codes: die meisten Aminosäuren haben mehr als ein Nukleinsäure-Codon. Das führt zu 'Gaps' und schlecht alignierten Teilen innerhalb codierender Regionen.

Wir haben einen multiplen Nukleinsäure-Alignmentalgorithmus entwikkelt, der genetische Information über codierende und nicht codierende Regionen als Teil der Scoring-Funktion nutzt, um die resultierenden Alignments zu verbessern. Unsere Implementation kombiniert (Mis)match-Scores von Nukleinsäuren mit denen der entsprechenden Aminosäuren innerhalb codierender Bereiche und Exons. Der Algorithmus zieht explizit Nutzen aus der Information über überlappende ORFs, wie sie in Virussequenzen oft vorkommen, um die Qualität der Nukleinsäure-Alignments weiter zu optimieren.

Der Algorithmus ist implementiert in dem Programmpaket `code2aln`, das frei erhältlich ist.

`Code2aln` ist eine Version eines Dynamic-Programming-Algorithmus nach dem 'Gotoh-Typ' mit affinen Gap-Penalties und einer komplexeren Scoring-Funktion, die Nukleinsäure- mit Aminosäure-Scores kombiniert.

Wir konnten zeigen, dass die Alignments tatsächlich signifikant verbessert wurden. Wir fanden die starke Tendenz von `code2aln`, Codons innerhalb codierender Regionen nicht durch das Einfügen von Gaps zu unterbrechen.

Wir haben die Resultate von `code2aln` als Input für die Detektion konservierter RNA-Sekundärstrukturelemente in RNA-Genomen von Leviviridae und prägenomischen RNA-Sequenzen von humanen Hepatitis B-Viren durch `alidot` angewandt.

Virusgenome sind ideale Testfälle für eine Methode, die Information über (überlappende) codierende Regionen nutzt, um deren Alignments zu verbessern, weil sie oft mehrere (auch überlappende) ORFs enthalten.

Wir fanden etliche hochsignifikante Sekundärstrukturelemente, die bis dato in der Literatur nicht beschrieben sind, sowie auch einige bekannte Elemente, wie das ε-Element und Elemente der HPRE-Region in Hepatitis B-Viren.

Auch die Resultate für die Levivirus-Gruppe sind hochinteressant: wir detektierten verschiedene Elemente, die durch kompensatorische Mutationen deutlich bestätigt sind, und wir konnten neue Einblicke in die strukturelle Organisation des Genus Levivirus gewinnen.

Danksagung

Ich danke Peter Stadler für die hervorragende und freundschaftliche Betreuung meiner Arbeit und die Möglichkeit, bei ihm diese Dissertation fertigzustellen. Danke an Ivo Hofacker für viele sehr hilfreiche Ratschläge und Hinweise.
Ich danke Peter Schuster für die freundliche Aufnahme an seinem Institut.
Vielen Dank auch an Christoph Flamm für seine immerwährende Hilfsbereitschaft.
Danke an alle meine Freunde und Kollegen: Michael Wolfinger, Stefanie Widder, Daniela Dorigoni, Michael Kospach, Kurt Grünberger, Judith Ivansits, Caroline Thurner, Christina Witwer, Stephan Bernhart, Stefan Müller, Andreas Svrcek-Seiler (die unerschöpfliche Quelle weiser Danksagungen), Andreas Wernitznig, Günther Weberndorfer, Bärbel Stadler, Jan Cupal, Ulli Mückstein, Uli Langhammer, Gil Benkö, Camille Stephan Otto Attolini, Jörg Hackermüller, Ingrid Abfalter, Sonja Prohaska, Claudia Fried.
Zuletzt vielen Dank an meine Eltern, die mir dieses Studium ermöglicht haben.

Contents

1 Introduction

During the last decades, research in the fields of molecular biology and biochemistry has provided the scientific community with huge amounts of sequence data sets. These data are available as entries in data banks such as `GenBank`. In many cases, however, there are no satisfactory tools to process the data [117]. One of the most basic and essential tools for data analysis in molecular biology is the alignment of nucleotide or amino acid sequences. In principle, it is used to tell whether two or more sequences are related and to give an impression how close a relationship is in terms of sequence similarity.

Multiple sequence alignments are used, for instance, to find diagnostic patterns that characterize protein families; to detect or demonstrate homology between new sequences and existing families of sequences; to help predict the secondary and tertiary structures of new sequences; to suggest oligonucleotide primers for PCR; and as an essential prelude to molecular evolutionary analysis [62].

Beside the basic necessity to process existing data, the rate of appearance of new sequence data is steadily increasing and the development of efficient and accurate automatic methods for multiple sequence alignments is of major importance [1, 3].

In fact, many advanced techniques of sequence analysis are dependent upon the availablity of high quality multiple sequence alignments. A procedure for extracting conserved secondary structure elements from a moderate size sample of related RNA sequences will be one example in this thesis. A procedure like this needs essentially a high quality alignment of nucleic acid sequences.

Recent research in our group aims at finding conserved secondary structure elements that are part of the genomes of RNA viruses and the pregenomic RNA intermediates of some DNA viruses like the Hepatitis B viruses

```
HBA_HUMAN    GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL
             ++ ++++H+ KV    + +A  ++           +L+ L+++H+ K
LGB2_LUPLU   NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
```

Figure 1: This figure shows a protein sequence alignment between a fragment of human alpha globin and leghaemoglobin from yellow lupin. Some identities are shown and some similar positions which have a positive score in the substitution matrix (indicated by '+'). This is a biologically meaningful alignment, in that we know that these two sequences are evolutionarily related.

```
ADI-MAL         AU-AGUACAUGGCAGAAUAAUGGUGC----AAGA------CU-AA--GU----AAUAGCACAGAGUC---AACUGGUAGUAUCACACUCCCAUG
AE-90CF402      AU-AGUACUUGGAUA---------------AAUGGAACCAUGCAGGAGGUU--AAUGGCACAAACUC---A---GGCAAUAUCACACUUCCAUG
B-896           AU-AGUACUUGGAAU-------G-------U-UA------CUGGAGGGACA--AAUGGCACUGAAGG---AAAUGACAUAAUCACACUCCAAUG
B-ACH320A       AU-AGUACUUGG------AAUGAUACUGGGAAUGUUA---CUGAAAGGUCA--AAUAACAAUGA------AAAU------AUCACACUCCCAUG
B-D31           AU-AGUACUUGGAAU-----------------GAUA---CUAAAGAGUCA--AAUAACACAAAU---------GGAACUAUCACACUCCCAUG
B-JRCSF         AU-AGUACUUGGAAU-------G-------A-UA------CUGAAAAGUCA--AGUGGCACUGAAGG---AAAUGACACCAUCAUACUCCCAUG
B-LAI           AU-AGUACUUGGUUU---AAUAGUACUUGGAGUA------CUGAAGGGUCA--AAUAACACUGAAGG---AAGUGACACAAUCACACUCCCAUG
B-MANC          AU-AGUACUUGGAAUACUGGG---------AAUGAUA---CUAGAGAGUCA--AAUGACACAAAUAA---UACUGGAAAUAUCACACUCCCAUG
B-OYI           AU-AGUACUUGGAAU-----------------GAUA---CUACAAGGGCA--AAUAGCACUGAA---------GUAACUAUCACACUCCCAUG
B-YU2           -------CUUGG------AAUGAUACUAGAAA--------------GUUA--AAUAACACUGGAAG---AAAU------AUCACACUCCCAUG
D-NDK           AU-AGUACAUGGAAU----------CA--GACUAAUAG---UACAGGGUUC--AAUAAUGGCACAG--------------UCACACUCCCAUG
O-ANT70         AUUA-UACCUUU-UCA----------UGUAACGGAACCACCUGUAGUGUUAGUAAUGUUAGUCAAGG------UAACAAUGGCACUCUACCUUG
SIVCPZGAB       CUGACAACAUUA-----------------------------------CA--AAUGGCAUU-----------------AUAAUACUGCCAUG
```

Figure 2: This figure shows part of a `clustalw` nucleic acid alignment of HIV-1 sequences. The region shown here is a coding region located at the center of the `env` gene. The alignment is highly disrupted. We see various gaps where they are not really necessary, because the biologically important part of the system is protein in this region of the genome.

[79, 91]. For this reason predicted secondary structures of genomic RNA sequences have to be compared on the basis of a reliable multiple sequence alignment.

As far as we know, almost all RNA molecules and consequently also almost all subsequences of a large RNA molecule form secondary structures. But the presence of secondary structure in itself therefore does not indicate any functional significance. Extensive computer simulations [30] showed that a small number of point mutations is very likely to cause large changes in the

```
ADI-MAL         CUCUAUACAACAGGGAUAGUAGGA--------------GAUAUAAGAAGAGCAUAUUGUACU
AE-90CF402      UUCCAUACAACAGGAAACAUAAAUGGU-----------GAUAUAAGAAAAGCAUAUUGUGAA
AE-CM240        UUCUAUAGAACAGGAGAUAUAAUAGGA-----------AAUAUAAGAAAAGCAUAUUGUGAG
B-896           UUUUAUGCAAGAAGAAACAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-ACH320A       UUUUAUGCAACAGGACAAAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-BCSG3         UAUUAUACAACAGGAGAAAUAGUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-CAM1          GUUUAUGCAACAGACAGAAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-D31           UUUUAUACAAAAGGAAAAAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-HIV1AD8       UUUUAUACAACAGGAGACAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGCAAC
B-HXB2          UUUGUUACAAUAGGAAAA---AUAGGA-----------AAUAUGAGACAAGCACAUUGUAAC
B-JRCSF         UUUUAUACAACAGGAGAAAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-LAI           UUUGUUACAAUAGGAAAA---AUAGGA-----------AAUAUGAGACAAGCACAUUGUAAC
B-MANC          UUUCAUGUAACAAGAGCCGUAACAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-OYI           UUUCAUACAACAAAACAAAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-SF2           UUUCAUACAACAGGAAGAAUAAUAGGA-----------GAUAUAAGAAAAGCACAUUGUAAC
B-WEAU          CUUUAUACAACAGGAGAAAUAAUAGGA-----------GAUAUAAGACGAGCACAUUGUAAC
B-YU2           UUGUAUACAACAGGAGAAAUAAUAGGA-----------GAUAUAAGACAAGCACAUUGUAAC
B-pNL43         UUUGUUACAAUAGGAAAA---AUAGGA-----------AAUAUGAGACAAGCACAUUGUAAC
D-ELI           CUCUAUACUACAAGAUCAAGAUCA---------------AUAAUAGGACAAGCACAUUGUAAU
D-NDK           CUCUAUACAAUAACAGGAAAAAAGAAGAAAACAGGA---UACAUAGGACAAGCACAUUGUAAA
O-ANT70         UACAGCAUGGGAAUAGGGGGAACAGCAGGAAACAGC------UCAAGGGCAGCUUAUUGCAAG
O-MVP5180       CGCAGUAUGACACUUAAAAGAAGUAACAAUACAUCACCAAGAUCAAGGGUAGCUUAUUGUACA
SIVCPZGAB       UUUUAUAAUAUAGAAAAUGUAGUAGGA-----------GAUACCAGAUCUGCCUACUGUAAG
```

Figure 3: A part of a nucleic acid alignment of HIV-1 sequences, also located in the `env` gene, but produced by another alignment algorithm using `env` protein sequence information for improving the alignment of the underlying nucleic acid sequences (`ralign`, see below). This region was aligned on the level of amino acids and reverse translated. Many identities are shown and make this alignment a biologically meaningful one, in that we see that these sequences are evolutionarily closely related.

secondary structures. A difference in the nucleic acid sequence of only 10% leads almost surely to unrelated structures if the mutated sequence positions are chosen randomly. Secondary structure elements that are consistently present in a group of sequences with less than, say 95%, average pairwise identity are therefore most likely the result of stabilizing selection, not a consequence of sequence homology. If selection acts to preserve a structural element, then it must of course have some function [29, 99, 98].

The development and implementation of computational methods capable of reliably predicting functional structural elements on the basis of sequence alignment information will provide immense benefits in terms of our understanding of the relationship between sequence and structure [11, 24, 30, 60]. Since only functional secondary structures are likely to be conserved, a procedure that detects and highlights conserved structural elements based solely

on already available sequence data could be used widely, e.g. to guide experimental mutagenesis or deletion studies [37, 38, 61].

Such methods will also be useful in tasks such as drug discovery or the study of molecular evolution [32]. Quite easily they could be applied to huge quantities of sequence data at our disposal nowadays to discover important structural motifs and trends in various macromolecules, without measuring the 3D structure of each macromolecule which is very laborious and expensive [4, 10, 16, 75, 97, 96].

It is not surprising that the quality of the input sequence alignment is important for this method. However, sequence heterogenity on the level of nucleic acids makes good alignments often infeasible even for phylogenetically closely related sequences. In many cases one observes too many gaps. This is caused by the inherent redundancy of the genetic code: most amino acids have more than one codon on the level of nucleic acids. As a result it is possible that two different nucleic acid sequences code for the same protein sequence. All three nucleotides in two codons might be different and, in spite of that, those codons might code for the same amino acid. In a protein alignment these amino acids would match each other while the differences on the level of nucleic acids can produce gaps in a nucleic acid alignment. This specific problem leads to various gaps within coding regions where they are not really necessary, because the biologically important part of the system is protein in this region of the genome. On the level of protein alignments many of these gaps could have been avoided.

The purpose of this thesis was to develop an alignment algorithm and a suitable scoring system to improve the quality of nucleic acid sequence alignments; especially nucleic acid alignments of RNA virus genomes.

Furthermore, procedures are described that utilize the information contained in the amino acid sequences of coding regions to construct an improved multiple alignment of the underlying nucleic acid sequences.

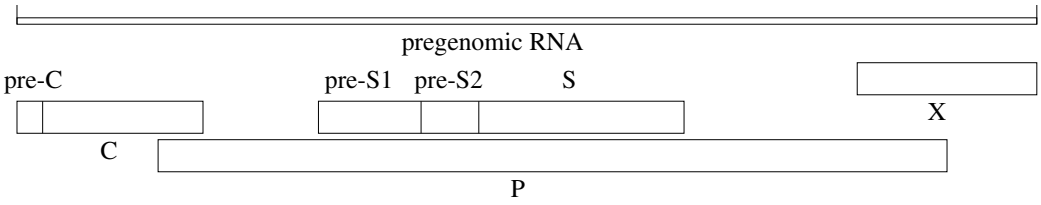


Figure 4: Diagram of the genome organization of hepatitis B virus with the four open reading frames (C, P, S, X). We can see the overlapping coding regions which are used as information for improving the alignments of hepatitis B virus strains on the level of nucleic acid.

Currently, there are two very different implementations of program packages which function according to this principle: The first algorithm, which consists of a combination of amino acid and nucleic acid based partial alignments, is implemented in the program package `ralign`. This program detects possible coding regions in all input data sets and aligns them on amino acid level while non-coding regions are handled as nucleic acids. Finally, the various alignments of the different parts of the input sequences are combined. `Ralign` is the only algorithm of this type currently available. (The procedure is described in detail in [101, 102, 103].)

The second algorithm processes the complete nucleic acid sequences and produces a 'real' nucleic acid alignment that, however, uses the information on coding and non-coding regions as part of the scoring function: `code2aln`.

The idea behind both the combined amino acid and nucleic acid based alignments (`ralign`) and the alignments produced by `code2aln` is that coding regions vary less on the level of protein than on the level of nucleic acid, because most amino acids are coded by more than one codon (base triplet) and some different nucleic acid sequences can produce the same protein sequence after translation. Protein sequences can still show substantial homology when the corresponding nucleic acid sequences are already essentially randomized.

In the following some examples will be shown how alignments of viral genomes can be improved, and, furthermore, how the number of gaps can be reduced without using unrealistically large gap parameters. Both programs have been applied to producing input for the program `alidot` [51] for finding conserved RNA secondary structures. The complete procedure was applied to pregenomic RNA of human hepatitis B virus and the bacteriophage Levivirus sequences. Secondary structures of ssRNA viruses or (as in the case of Hepatitis B virus) pregenomic RNA intermediates are known to play an important role in the regulation of the viral life cycle [39, 74]. Much research is performed to find conserved secondary structure elements as part of the genomes of RNA viruses (e.g. Levivirus) and the pregenomic RNA intermediates of some DNA viruses (e.g. the Hepatitis B viruses) [79, 91].

Secondary structure prediction was performed using the algorithms of the `Vienna RNA package` [52, 54], such as `RNAfold`, which calculates the minimum free energy structure and the partition function and base pairing probability matrix of a RNA sequence.

The final step of the procedure is the detection of structural conservation by means of compensatory mutations. `Alidot` detects conserved secondary structure elements in relatively small sets of RNAs by combining multiple sequence alignments and secondary structure predictions. The starting point of this approach is the list of all predicted base pairs. The multiple sequence alignment is very useful to establish which base pairs from different sequences correspond to each other. Then the individual base pairs are ranked by certain filtering procedures. The procedure is described in detail in [53, 51].

As expected, the performance of `alidot` depends not only on the quality of the RNA secondary structure prediction but also crucially on the quality of the input multiple sequence alignment. It will be shown that the alignments are indeed improved by usage of genetic information.

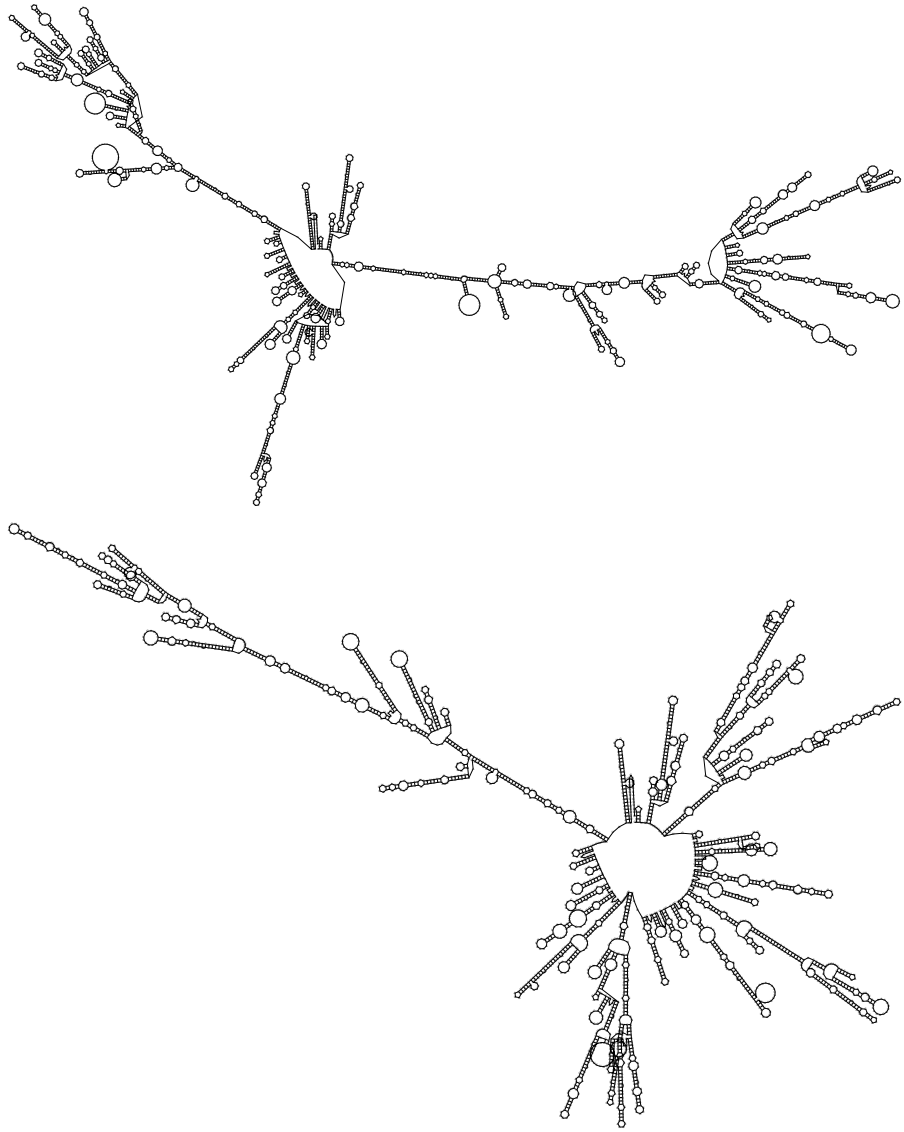


Figure 5: Two predicted secondary structures of the complete RNA pregenomes of (closely related) human hepatitis B viruses (AB014360 and HBD50521). Note that there is hardly similarity between the predictions: little changes in the nucleic acid sequences cause large changes in secondary structures.

2 Theoretical Background

2.1 Alignments in Principle

An alignment is the most basic sequence analysis task. It is used to tell whether two or more sequences are related and to give an impression how close their relationship is in terms of sequence similarity. To find the best possible alignment of sequences is of central importance for bioinformatics and data processing after routine laboratory procedures like sequencing nucleic acids. Some alignment algorithms exist which are used to find an optimal (global or local) alignment, and, of course, a scoring system is necessary to rank alignments. In principle, all known algorithms are based on two criteria, (i) maximum similarity or (ii) minimum (Hamming-) distance [28, 33, 50].

For evaluating the difference between two sequences we have three possibilities of pairs of opposite symbols: (i) identity, (ii) substitution or mismatch and (iii) insertion or deletion. The procedure is usually done by first aligning the sequences and then deciding whether that alignment is more likely to have occurred because the sequences are related, or just by chance. In any case the scoring system should help to answer this question regarding to identical and similar positions in the alignment. (Similar pairs of residues in amino acid alignments are those which have a positive score in the substitution matrix used to score the alignment, e.g. aspartate-glutamate pairs, D-E, both negatively charged amino acids.)

2.2 The Scoring of Alignments

Careful thought must be given to the scoring system used to evaluate an alignment when we are looking for evidence that they have diverged from a common ancestor by a process of mutation and selection. The basic mutational processes that are considered are substitutions, which change residues

in a sequence, and insertions and deletions, which add or remove residues and are together referred to as 'gaps'. The total score of an alignment is a sum of terms for each aligned pair of residues, plus contributions for each gap. Informally, using an additive scoring system we expect identities and conservative substitutions to be more likely in good (biologically relevant) alignments than we expect by chance, and so they should contribute positive score terms. And on the other hand, non-conservative changes are expected to be observed less frequently in real alignments than we expect by chance, and so these contribute negative score terms. This system also corresponds to the assumption that we can consider mutations at different sites in a sequence to have occurred independently (treating a gap of arbitrary length as a single event). All alignment algorithms depend crucially on such a scoring scheme and from a biological point of view the assumption of independence appears to be a reasonable approximation for DNA and protein sequences, although we know that intramolecular interactions between residues of a protein play a very important role in determining protein structure. Regarding the secondary structures of RNAs, where base pairing introduces very critical long range dependencies, the model of independent mutations is biologically inaccurate [59, 64, 67].

We need score terms for each aligned residue (or base) pair. We derive substitution scores from a probabilistic model that gives a measure of the relative likelyhood that the sequences are related as opposed to being unrelated. We do this by having models that assign a probability to the alignment in each of the two cases. Then we consider the ratio of the two probabilities. The *random* model R assumes that a letter in the sequence (for proteins an amino acid or one of the four bases in the case of DNA or RNA) occurs independently with some frequency q, and hence the probability of the two

sequences is the product of the probabilities of each amino acid (or base):

$$P(x, y|R) = \prod_i q_{x_i} \prod_j q_{y_j} \tag{1}$$

where x and y is a pair of sequences, x_i is the ith symbol in x and y_j is the jth symbol in y. These symbols come from an alphabet (A, G, C, T, U in the case of nucleic acids or an amino acid in the case of protein). In the alternative *match* model M, aligned pairs of residues occur with a joint probability p_{ab}. This value p_{ab} can be thought of as the probability that the residues a and b have each independently been derived from some unknown original residue c in their common ancestor (c might be the same as a and/or b). This yields a probability for the whole alignment:

$$P(x, y|M) = \prod_i p_{x_i y_i} \tag{2}$$

The ratio of these two likelihoods is the 'odds ratio',

$$\frac{P(x, y|M)}{P(x, y|R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}} \tag{3}$$

We want to arrive at an additive scoring system, so we have to take the logarithm of this ratio, known as the 'log-odds ratio',

$$S = \sum_i s(x_i, y_i) \tag{4}$$

where

$$s(a, b) = \log(\frac{p_{ab}}{q_a q_b}) \tag{5}$$

is the log-likelihood ratio of the residue pair(a, b) occurring as an really valid aligned pair, as opposed to an unaligned pair (or by chance joined pair of residues or nucleic acids). We can see that S in this equation is a sum of individual scores $s(a, b)$ for each aligned pair of residues. And these individual scores, these log-odds values can be rounded to the nearest integer

for purposes of computational efficiency and then arranged in a matrix. For instance, in the case of proteins the matrix is a 20×20 matrix which gives an individual score $s(a_i, b_j)$ for sequences a and b in position i and j. The highest positive entries in the matrix are given for identical residue pairs, lower, but also positive, values do the conservative substitutions have while non conservative substitutions give a negative score. So it is possible to derive scores, in fact $s(a, b)$ in the above equation, for every pair of residues in the alignment. Any matrix like this is making a statement about the probability of observing ab pairs in real (biologically relevant) alignments and is called substitution matrix or score matrix or weight matrix. Examples of substitution matrices are the `BLOSUM50`, the `BLOSUM62` [47] or the `PAM250` matrix [25].

The next point is penalising gaps. There are two possibilities: the standard cost associated with a gap of length g could be given by a linear score

$$\gamma(g) = -gd \tag{6}$$

where d is called the gap open penalty. But it seems to be more legitimate to make a difference whether a gap is newly opened or an existing gap is just extended. A type of score could be used which is known as the affine score

$$\gamma(g) = -d - (g-1)e \tag{7}$$

where e is called the gap extension penalty. This penalty should be set to something less than the gap open penalty d, so that extension of existing insertions (or deletions) is penalised less than opening further gaps (as it would be by the linear gap cost). Gap penalties also correspond to a probabilistic model of alignment. We assume that the probability of a gap occurring at a particular site in a given sequence is the product of a function $f(g)$ of the length of the gap, and the combined probability of the set of inserted

residues,

$$P(\text{gap}) = f(g) \prod_{i \in \text{gap}} q_{x_i}. \tag{8}$$

The form of this equation as a product of $f(g)$ with the q_{x_i} terms corresponds to an assumption that the length of the gap is not correlated to the residues it contains. The natural values for the q_a probabilities here are the same as those used in the random model above, because they both correspond to unmatched independent residues. When we divide by the probability of this region according to the random model to form the odds ratio, the q_{x_i} terms cancel out. This leaves us with a term dependent on length $\gamma(g) = \log(f(g))$. Gap penalties correspond to the log probability of a gap of that length.

On the other hand, if there is evidence for a different distribution of residues in gap regions then there should be residue-specific scores for the unaligned residues in gap regions, equal to the logs of the ratio of their frequencies in gapped versus aligned regions. This might happen if it is expected that polar amino acids are more likely to occur in gaps in protein alignments than indicated by their average frequency in protein sequences, because the gaps are more likely to be in loops on the surface of the protein structure than in the buried core.

After having determined a certain scoring system, we need to have an algorithm for finding an optimal alignment for a pair of sequences. The alignment problem becomes very complicated when gaps are allowed: for sequences of length 30 there are 10^9 possibilities, and with length 60 we have 10^{18} possible alignments. But in terms of molecular biology sequences of this length are comparatively short, and often it is necessary to find the best alignment between sequences which have a length of some thousand amino acids or nucleotides (like in the case of virus genomes). It is of course not computationally feasible to enumerate all these.

So we need to find a way to obtain optimal alignments without testing and

valueing every possible solution. An algorithm for finding optimal alignments given an additive alignment score is called dynamic programming. It implies that we get the best possible alignment as a result of an optimal alignment till each current position. Using the introduced scoring scheme better alignments have higher scores. So what we have to do is to maximise the score to find the optimal alignment as opposed to other interpretations of scoring which search for minimal distances or costs. Both approaches have been used in the biological sequence comparison literature. Dynamic programming algorithms apply to either case. The differences are, simply said, just exchanges of 'min' for 'max' [82, 85, 86].

2.3 Pairwise Alignment Algorithms

The most important dynamic programming algorithm in biological sequence analysis for obtaining the optimal global alignment between two sequences, allowing gaps, is the Needleman-Wunsch algorithm [85, 6], introduced in 1970.

The idea behind all versions is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. A matrix F of the two sequences is constructed, indexed by i and j, one index for each sequence, where the value $F(i, j)$ is the score of the best alignment between the initial segment $x_{1...i}$ of x up to x_i and the initial segment $y_{1...j}$ of y up to y_j. The score value $F(i, j)$ is builded recursively and we start by initialising $F(0, 0) = 0$.

We have three possibilities of pairs of opposite symbols: (i) identity, (ii) substitution or 'mismatch' and (iii) insertion or deletion. So we proceed to fill the matrix from top left to bottom right, from the first letters of the sequences to their ends. Along the top horizontal row (where $j = 0$) and the first vertical column (where $i = 0$) we write the pairs of one sequence's letters with

a gap in the second sequence and get the scores of these gaps by multiplying the position of the letter by the gap penalty. So the values $F(i,0)$ represent alignments of a prefix of x to all gaps in y and we can define $F(i,0) = -id$. Likewise down the left column $F(0,j) = -jd$. If $F(i-1,j-1), F(i-1,j)$ and $F(i,j-1)$ are known, it is possible to calculate $F(i,j)$. There are three possible ways that the best score $F(i,j)$ of an alignment up to x_i, y_j could be obtained: x_i could be aligned to y_j, in which case $F(i,j) = F(i-1,j-1) + s(x_i,y_j)$, where $s(x_i,y_j)$ is the individual score for this pair of amino acids or nucleotides; or x_i is aligned to a gap, in which case $F(i,j) = F(i-1,j) - d$, where d is the gap penalty; or y_j is aligned to a gap, in which case $F(i,j) = F(i,j-1) - d$. The best score up to (i,j) is the largest of these three options. Therefore, we have

$$F(i,j) = \sup \begin{Bmatrix} F(i-1,j-1) + s(x_i,y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{Bmatrix} \tag{9}$$

This equation is applied repeatedly to fill in the matrix of $F(i,j)$ values, calculating the value in the bottom right-hand corner of each square of four cells from one of the other three values (above left, left, or above). And as we fill in the $F(i,j)$ values, we also keep a pointer in each cell back to the cell from which its $F(i,j)$ was derived. Finally the value in the bottom right cell of the matrix $F(n,m)$ is by definition the best score for an alignment of $x_{1\ldots n}$ to $y_{1\ldots m}$. To gain the alignment itself, we must find the path of choices which led to this final value. The procedure for doing this is called backtracking. We build the alignment in reverse, starting from the final cell, and following the pointers that we stored when building the matrix. At each step in the backtracking process we go back from the current cell (i,j) to the one of the cells $(i-1,j-1), (i-1,j)$ or $(i,j-1)$ from which the value $F(i,j)$ was derived. So with every step we get a pair of symbols and add it to the growing alignment: x_i and y_j if the step was to $(i-1,j-1)$, x_i and

the gap character '-' if the step was to $(i-1, j)$, or '-' and y_j if the step was to $(i, j-1)$. Finally we reach the starting point of the matrix, $i = j = 0$. The reason that the algorithm works is that the score is made of a sum of independent pieces, so the best score up to some point in the alignment is the best score up to the point one step before, plus the incremental score of the new step.

2.4 Alignments with Affine Gap Penalties

In fact, the common way of penalizing gaps is not the (more simple) linear score where the standard cost associated with a gap of length g is given by

$$\gamma(g) = -gd \tag{10}$$

where d is called the gap open penalty (see above). Mostly, it seems to be more legitimate to make a difference whether a gap is newly opened or an existing gap is just extended. The type of score used here is known as the affine gap score:

$$\gamma(g) = -d - (g-1)e \tag{11}$$

where e is called the gap extension penalty. Also for this form of gap cost there is once again an efficient implementation of dynamic programming. Doing affine gap alignments requires keeping track of multiple values for each pair of letters instead of the single value $F(i, j)$. Let us define $M(i, j)$ to be the best score up to (i, j) given that x_i is aligned to y_j. Let $I_x(i, j)$ be the best score given that x_i is aligned to a gap, and $I_y(i, j)$ be the best score given that y_j is aligned to a gap (this corresponds to the case that y_j is part of an insertion with respect to x, and vice versa).

The recurrence relations now become more complex:

$$M(i,j) = \sup \begin{Bmatrix} M(i-1,j-1) + s(x_i, y_j), \\ I_x(i-1,j-1) + s(x_i, y_j), \\ I_y(i-1,j-1) + s(x_i, y_j). \end{Bmatrix} \tag{12}$$

$$I_x(i,j) = \sup \begin{Bmatrix} M(i-1,j) - d, \\ I_x(i-1,j) - e, \\ I_y(i-1,j) - d. \end{Bmatrix} \tag{13}$$

$$I_y(i,j) = \sup \begin{Bmatrix} M(i,j-1) - d, \\ I_x(i,j-1) - d, \\ I_y(i,j-1) - e. \end{Bmatrix} \tag{14}$$

In these equations, we assume that it is possible for a deletion to be followed directly by an insertion. As previously, we find the alignment itself using the traceback procedure.

The scoring system defined here can be described by a type of diagram named finite state automaton. This shows a state for each of the three matrix values, with arrows between the states which symbolize the possible transitions. The transitions each carry a score increment, and the states each specify a $\Delta(i,j)$ pair, which is used to determine the change in indices i and j when the state is entered. The recurrence relations of all matrix values can be read directly from the finite state automaton. But it is important to note that this type of diagram does not reflect the alignment algorithm itself. In fact, it only summarizes the possible scorings, and the progress of the alignment can be represented as a walk through the diagram from one state to another via the transition paths: the new value for a state variable at (i,j) is the maximum of the scores corresponding to the transitions coming into the state. Each transition score is given by the value of the source state at the offsets specified by the $\Delta(i,j)$ pair of the target state, plus the

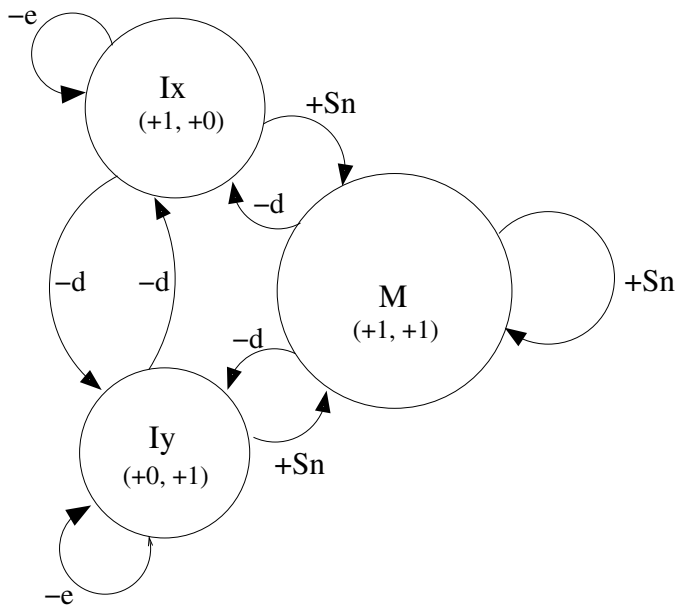


Figure 6: This finite state automaton is a diagram of the relationships and transitions between the three states used for affine gap alignment. The progress of the alignment can be represented as a walk through the diagram from one state to another via the transition paths. M means (mis)match, Ix and Iy are insertions and deletions, Sn is the increment of nucleic acid score, and d and e represent the gap open and extension penalties.

specified score increment. Letters from the underlying pair of sequences are transferred to the alignment referring to the $\Delta(i, j)$ values in the states.

Also the algorithm of the `code2aln` implementation makes use of the 'Gotoh-type' affine gap penalty algorithm, even though with some extensions [35].

2.5 Multiple Alignments

Full dynamic programming is used in order to align just two sequences. This guarantees a mathematically optimal alignment, given a table of scores for matches and mismatches between all amino acids or nucleotides and penal-

ties for insertions or deletions of different lengths. Attempts at generalising dynamic programming to multiple alignments, however, are limited to small numbers of short sequences [73]. For more than ten or so proteins of average length, the problem is infeasible given current computer power. Therefore, all of the methods capable of handling larger problems in practical timescales make use of heuristics. Nowadays, the most widely used approach is to exploit the fact that homologous sequences are evolutionary related. We can produce a multiple alignment progressively by a series of pairwise alignments, following the branching order in a phylogenetic tree [26]. We first align all possible pairs of sequences and derive a either a distance matrix or a resulting pairwise alignment score matrix in order to calculate the initial guide tree which is built up by the distances or similarities between the sequences. Then the most closely related sequences get aligned progressively according to the branching order in the guide tree, gradually adding in the more distant ones when we already have some information about the most basic mismatches or gaps. Some information which is derived from the first pairwise alignments of the most closely related sequences.

This approach is fast enough to allow alignments of virtually any size. Further, in most (simple) cases, the quality of alignments of this type is very good, as judged by the ability to correctly align corresponding domains from sequences of known secondary or tertiary structures [5]. So this approach also works well if the data sets consist of sequences of different degrees of divergence. By the time the most distantly related sequences are aligned, one already has a sample of aligned sequences which gives important information about the variability at each position. The placement of gaps in alignments between closely related sequences is much more accurate than between distantly related ones. Therefore, the positions of the gaps which were introduced during the early alignments of the closely related sequences are not changed as new sequences are added. One problem is that this ap-

proach becomes less reliable if all of the sequences are highly divergent. More specifically, any mistakes like misaligned regions made early in the alignment process cannot be corrected later as new information from other sequences is added. Thus, there is no guarantee that the global optimal solution has been found and the alignment is not caught in a local minimum. This risk increases with the divergence of the initially aligned sequences and is thought of as mainly resulting from an incorrect branching order in the initial tree. Initial trees are derived from a matrix of distances between the separately aligned pairs of sequences in the first steps of the multiple alignment process. Most relevant errors occur during these initial alignments.

Furthermore, the parameter choice problem is very important. One chooses a weight matrix and two gap penalties (one for opening a new gap and one for extension of an existing gap) and expects that these should work well over all parts of all the sequences in the data set. When the sequences are closely related this works in most cases. This problem also becomes worse with increasing sequence diversity. All residue weight matrices give most weight to identities. If identities dominate an alignment, almost any weight matrix will find approximately the correct solution. With very divergent sequences the scores given to non-identical residues will become critically important, because there are more mismatches than identities.

Another problem arises with the choice of the best gap penalties. The range of gap penalty values which will find the correct or best possible solution can be very broad for highly similar sequences, but as more and more divergent sequences are used, the exact values of the gap penalties become very important for success [113]. Further, in protein alignments, gaps do not occur randomly. They occur far more often between the major secondary structural elements like helices than within [90].

The first basic step of the multiple alignment algorithm described here is aligning separately all pairs of sequences in order to calculate a resulting final

pairwise score matrix giving the divergence of each pair of sequences. Accurate scores for constructing the best pairwise alignments are derived from full dynamic programming alignments using two gap penalties (for opening or extending gaps) and a full usage of the scoring function including an amino acid score matrix for information regarding the coding regions (see below).

The tree used to guide the final multiple alignment process is computed from the final pairwise score matrix derived in the first step. This method produces trees with branches and branching points that reflect the divergence relations calculated from the pairwise alignments.

Then the progressive profile alignments of already existing smaller alignments start. The basic procedure at this stage is to use the already available series of pairwise alignments to align larger and larger groups of sequences, following the branching order in the guide tree. First the most similar sequences or sequence groups at the tips of the tree get aligned. Then this alignment gets aligned with the third most similar sequence or group which is something more divergent from the first two sequences (groups) and so on till the last sequence (group) with the least homology gets aligned with an alignment consisting of all other sequences. See figure 7 for an example of a guide tree and the process of a multiple alignment.

At each stage a full dynamic programming algorithm [84] is used with a (mis)match score value, a score value for possible underlying coding regions in all frames, and penalties for opening and extending gaps. So each step consists of an alignment of two existing alignments or sequences. Gaps that are present in former alignments remain fixed, but new gaps that are introduced at each state of alignment get full opening and extending penalties even if they are introduced inside old gap positions in other sequences. The score between a position from one sequence or alignment and one from another is calculated as the average of all pairwise nucleic acid match scores and the appropriate amino acid weight (substitution) matrix scores in the two sets.

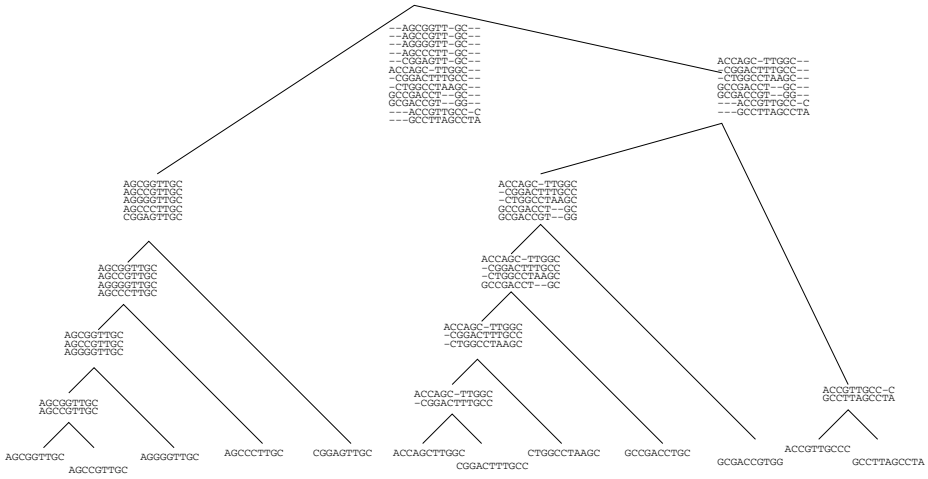


Figure 7: An example for a guide tree and a process of a multiple alignment. The guide tree tells nothing about the evolutionary distances between the sequences. It only produces an order of the profile alignments.

For example, aligning two alignments with 3 and 4 sequences, the final score for this position is the average of 12 (3×4) comparisons.

Two different gap penalties are used: a gap opening penalty, which gives the cost of opening a new gap of any length, and a gap extension penalty, which gives the cost of every item in the gap. We vary gap penalties and use weight (substitution) matrices of amino acid comparisons of amino acids which are encoded by the aligned nucleic acid sequences to improve the accuracy of the sequence alignments.

In principle, it is possible to offer two main series of weight matrices to the user: the `Dayhoff PAM` series [25] and the `BLOSUM` series [47]. In each case there is a choice of matrices ranging from strict ones, useful for comparing very closely related sequences, to less strict ones which are useful for aligning more divergent sequences. Depending on the distances between the two sequences or groups of sequences to be compared, `CLUSTAL W`, for instance, switches between 4 different matrices in each series. The distances

are measured directly from the guide tree [105]. But in our applications we prefer to reduce ourselves to one highly universal amino acid scoring matrix (`BLOSUM62`) [47] for the reason of reproducibility, because our algorithm of combining the information on nucleic acid level with that on amino acid level makes the scoring function more complex a priori.

2.6 Some Other Multiple Alignment Algorithms

In the following, a short overview about currently used alignment algorithms is given. All of them use heuristics, because the problem of exact dynamic programming for multiple alignments is, that the solution is computationally not accessible in most cases for sequences of typical biological lengths, given current computer power. None of the methods mentioned here makes use of genetic information about coding and amino acid scores within nucleic acid sequences to improve the resulting multiple alignment.

Calign

Genomic DNA and cDNA are compared to improve the understanding of coding regions consisting of exons and introns. The algorithm uses restricted affine gap penalties which penalize long gaps with a constant penalty. Several techniques developed for solving the approximate string-matching problem are employed for computing the optimal alignment with restricted affine gap penalties. The algorithms are derived based on the suffix automaton with failure transitions and on the diagonalwise monotonicity of the cost tables. The source code is freely available [18, 19].

FramePlus

An algorithm for DNA-Protein sequence alignment is used for automated annotation of Expressed Sequence Tags (ESTs). The approach is to align

gene fragments against well-documented databases of protein sequences. The SCOP database was used to develop a general framework for testing the sensitivity of such an alignment algorithm when searching large databases. In this framework, the performance of `FramePlus` was found to be better than other algorithms in the presence of moderate and high rates of frame shift errors. The source code for `FramePlus` is freely available [40, 41].

The dead-end elimination algorithm

A polynomial-time algorithm for rigorously solving the local multiple alignment problem for extracting functionally important regions shared by a family of protein sequences. The algorithm is based on the dead-end elimination procedure to avoid an exhaustive search: certain rejection criteria eliminate those sequence segments and segment pairs that can be mathematically shown to be inconsistent (dead-ending) with the globally optimal alignment. Iterative application of the elimination criteria reduce combinatorial possibilities without considering them explicitly. In contrast to the exhaustive search, whose computational complexity is combinatorial, the number of operations required to eliminate the dead-ending segments and segment pairs grows quadratically and cubically, respectively, with the total number of sequence elements. The source code is available from the authors [78].

OWEN

This is an interactive tool for aligning two long DNA sequences that represents similarity between them by a chain of collinear (non-conflicting) local similarities. `OWEN` uses several methods for constructing and editing local similarities and for resolving conflicts between them. Similarity between two long orthologous regions of genomes can be represented by a chain of *local* similarities. Within such a chain, pairs of successive similarities are collinear.

OWEN is supposed to find the true chain of these local similarities. Many conflicts between pairs are resolved by deleting a similarity by a stronger similarity. This constructs a chain of similarities faster than when a chain is sought optimally with some global criterion. The software is freely available [94, 95].

POA - Partial Order Alignments

One common problem of many progressive multiple sequence alignment algorithms is the fact that they depend on reducing an alignment to a linear profile for each alignment step. This leads to loss of information and gap scoring artefacts. POA represents an alignment as a graph that can itself be aligned directly by pairwise dynamic programming, eliminating the need to reduce the alignment to a profile. POA guarantees that the optimal alignment of each new sequence versus each sequence will be considered. Furthermore, a new edit operator (homologous recombination) is introduced to the algorithm, which is important for multidomain sequences. POA is significantly faster than other multiple sequence alignment algorithms [69], and is available at [70].

Handel

Handel is a software implementation of a synthesis of probabilistic sequence alignment, profiling and phylogeny: A multiple alignment algorithm for Bayesian inference in the links model proposed in [107]. The program samples from and/or maximizes the posterior distribution over multiple alignments for any number of DNA or protein sequences, conditioned on a phylogenetic tree. No more computational resources than for pairwise alignment are required. Handel is freely distributed [57, 58].

Divide-and-Conquer Alignment Algorithm

Also this method allows to quickly compute heuristic multiple sequence alignments. The sequences are cut at certain positions near to their center. This divides the problem of aligning K long sequences into the two problems of aligning the shorter K prefix and K suffix sequences. Assuming that it is possible to compute optimal alignments of these two sets of shorter sequences, an alignment of the complete sequences is obtained by conctatenating the prefix alignment and the suffix alignment. If one of those is still too long to be aligned optimally, the procedure is repeated until the sequences are of a length short enough to be tractable for the exact alignment procedure. The choice of the cut positions is critical and determined using heuristic approaches [92, 104, 108]. The program is available at [109].

ComAlign

This is a heuristic method that extracts qualitatively good sub-alignments from a set of multiple alignments and combines these into a new, often improved alignment. The algorithm is implemented as a variant of the traditional dynamic programming technique. `ComAlign` actually does combine parts from different alignments and not just select the best of them. By this method it is guaranteed that the results always lie within a certain distance of an optimal solution (given a measure of quality) [14]. The source code of `ComAlign` is free and available on [13].

ProtEST

This method constructs multiple protein sequence alignments from protein sequences and translated expressed sequence tags (ESTs). `ProtEST` is more effective than a simple `TBLASTN` search of the query against the EST database, as the sequences are automatically clustered, assembled, made non-redund-

ant, checked for sequence errors, translated into protein and then aligned and displayed. The ProtEST method [23] is available as an internet (WWW) service at [22].

DIALIGN

In the segment-by-segment approach to sequence alignment, as implemented in DIALIGN, pairwise and multiple alignments are generated by comparing gap-free segments of the sequences under study. The input can be seen as a set of non-gapped segment pairs (diagonals). Given a weight function that assigns a weight score to every possible diagonal, the goal is to choose a consistent set of diagonals of maximum weight. In this approach, the score of an alignment is defined as the sum of all weights of these segment pairs. A later modification of the weight function used in the original version of the alignment program DIALIGN, published recently, has two further important characteristics: it can be applied to both globally and locally related sequence sets, and the running time of the program is considerably improved. The program is available online at the Bielefeld University Bioinformatics Server (BiBiServ) [71, 72, 83].

2.7 RNA Secondary Structure Prediction

RNA polymers are macromolecules, consisting of a linear arrangement of building blocks, the monomers [42, 118]. RNA polymers have the ability to fold back on themselves, due to interactions between individual base pairs. For biopolymers like proteins or RNA these interactions are specific, and can lead to the adoption of a unique compact conformation called 'native state'. During the structure formation process both, RNA and proteins, try to minimize the solvent exposure of hydrophobic residues by burying these residues in the interior of the structure [48]. But it is self-evident from

the different chemical nature of RNA and proteins that the ways how these macromolecules achieve their compact conformation is different. For proteins the driving force of the collapse into compact conformations is the formation of a hydrophobic core. For RNA the formation of compact conformation is promoted by the tendency to maximize the stacking interaction between base pairs [49]. And it is essential for living cells that this formation of the correct and functional conformation is achieved in biologically relevant sufficiently short time [44, 68, 114, 115, 116].

From a theoretical point of view, the problem of how biopolymers achieve their native state splits up into two aspects. The first aspect is the structure prediction problem. The second aspect deals with the dynamics of the folding process itself [15, 17, 46, 63].

Since the sequence of a biopolymer specifies its three-dimensional structure, it should be possible, at least in principle, to predict its native structure solely from the knowledge of the sequence [56]. And in fact, as is becoming increasingly clear, biopolymers like proteins or RNA are flexible and rapidly fluctuating molecules whose structural mobilities have functional significance [20, 80].

The native states of RNA consist of a large ensemble of closely related and rapidly inter-converting conformational sub-states of nearly equal stabilities. Theoretical methods for structure prediction require extensive computation. The secondary structure of RNA is defined as the pattern of base pairs, which is formed by hydrogen bonds between atoms of the four bases [89, 88].

For RNA folding, powerful algorithms [87, 119] based on the method of dynamic programming [6] and experimentally measured energy parameters have been developed [31, 45, 60, 110].

`RNAfold` as part of the `Vienna RNA package` calculates the minimum free energy structure from RNA sequences, the partition function and base pairing probability matrix [52, 81]. It returns the minimum free energy structure,

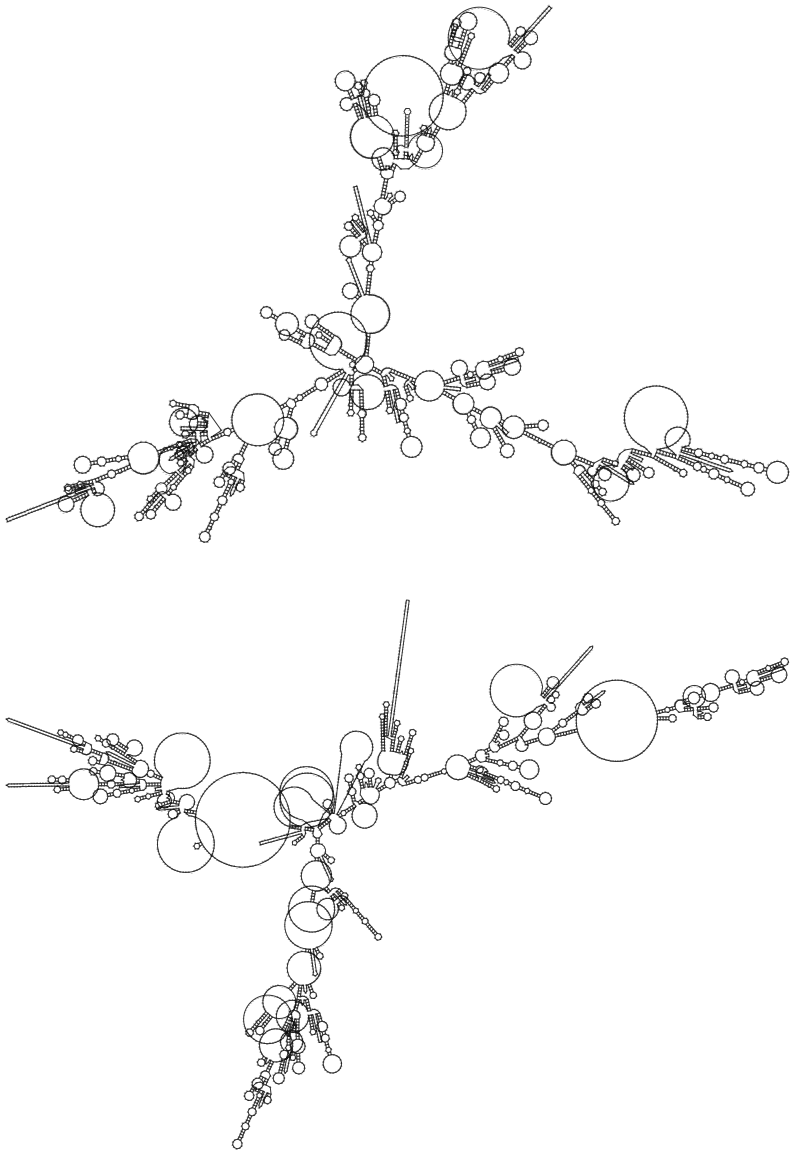


Figure 8: Two predicted secondary structures of the complete RNA genomes of Levivirus genus. These plots were generated using `RNAalifold` which is part of the `Vienna RNA package` [52]. The only difference between the plots is the alignment procedure (`clustalw` and `code2aln`) that was used for producing the input.

its energy, the free energy of the thermodynamic ensemble and the frequency of the minimum free energy structure in the ensemble. It also produces PostScript files with plots of the resulting secondary structure graph and a dot plot of the base pairing matrix. The dot plot shows a matrix of squares with area proportional to the pairing probability in the upper half, and one square for each pair in the minimum free energy structure in the lower half.

`RNAalifold`, also part of the `Vienna RNA package`, does the same, in principle, but processes alignments. See figure 8.

The results that were produced in this work using `RNAfold` are used as an input for `alidot` [54, 55, 51].

2.8 Inherent Difficulties of Nucleic Acid Alignments

Alignments of nucleic acid sequences bear one main problem: the higher sequence heterogenity on the level of nucleic acid, as compared to proteins, makes good alignments often impossible. The resulting alignments contain too many gaps and poorly aligned regions although the sequences should be very similar regarding their high degree of relationship. While protein sequences can still show substantial homology, the corresponding nucleic acid sequences have already evolutionary diverged, thus they are essentially randomized. This is caused by the inherent redundancy of the genetic code: most amino acids have more than one codon on the level of nucleic acid. As a result it is possible that two different nucleic acid sequences code for the same protein sequence. In a protein alignment these amino acids would match each other while the differences on the level of nucleic acids can produce gaps in a nucleic acid alignment. This specific problem leads to various gaps and badly aligned segments within coding regions where they are not really necessary, because the biologically important part of the system is protein at this region of the genome.

```
SARGLSSTVSLGQFEHWSPR
+AR+LS+TVSL+QF+H SPR
NARNLSDTVSLSQFDHPSPR

AGTGCAAGAGGATTAAGTAGTACAGTAAGTTTAGGACAATTTGAACATTGGAGTCCAAGA
   GC  G G   T       AC G      T    CA TT GA CA       CC  G
GACGCCCGCGACCTCTCCGACACCGCTTCCCTCTCCCAGTTCGACCACCCCTCCCCCCGC
```

Figure 9: Example for the problem of higher sequence heterogenity on the level of nucleic acids. It shows an hypothetical amino acid alignment on top which represents a high degree of similarity between both protein sequences allowing for an unambiguous alignment. Below the same sequences are aligned on the level of nucleic acids. It is clearly visible that the sequences are much more heterogenous: the pairwise identity is only 33%. This is only slightly above the 25% identity expected for two random nucleic acid sequences.

Therefore, on the level of protein, many of this alignment problems of higher sequence divergency of nucleic acids as compared to the underlying protein sequences could have been avoided. And, in most cases, it is possible to obtain better alignments: The scores (the per cent homologies) are higher and the number of gaps within the protein sequences is not as high as it would be in the case of nucleic acids. Reducing the gaps within an alignment improves the resulting alignment which may be used as input into other sequence data processing programs like those for secondary structure prediction.

So, what we need is a procedure that utilizes the information contained in the amino acid sequences to construct an improved multiple alignment of the underlying nucleic acid sequences.

3 A First Attempt: The ralign Project

3.1 The Idea behind ralign

The idea behind the combined amino acid and nucleic acid based alignments (`ralign`) is that coding regions on the level of protein vary less than on the level of nucleic acid, because most amino acids are encoded by more than one codon (base triplet) and some different nucleic acid sequences can produce the same protein sequence after translation. Thus, this first approach was to improve the quality of sequence alignments of RNA viruses (especially the pregenomic RNA intermediate of Hepatitis B virus) by creating and implementing an environment for a combined alignment algorithm.

The algorithm proposed formerly has the following features: first the program detects all possible coding regions of a minimum size without user intervention. The next step is translation of the detected coding regions into amino acid sequences.

Corresponding open reading frames are identified and then aligned. While the program can operate fully automatically, manual alteration of the list of corresponding open reading frames by the user is possible and often recommended.

The protein alignment results are then reverse translated back into nucleic acids and the non coding regions between the open reading frames are aligned as nucleic acids using the alignments of the open reading frames as constraints.

The last step consists of joining all reverse translated protein sequence alignments with the nucleic acid alignments of the non coding regions.

In many cases, the sequences under consideration code for proteins. It is well known that amino acid sequences can be aligned much more reliably than their underlying genomic DNA or RNA sequences.

One could argue that the quality of sequence alignments could be im-

proved simply by translating the entire nucleic acid sequence into protein and processing on the level of proteins. But a very important factor is that the viral genomic sequence could consist of more than just one open reading frame (various coding regions in different frames) as well as some non-coding regions. These non-coding regions should, of course, be processed as nucleic acids, and every open reading frame should be processed in the correct frame.

3.2 The ralign Algorithm

The combined amino acid and nucleic acid based alignment procedure is made available in the program called `ralign`. The source code of the package is written in the C programming language and will run on all computers with an ANSI C compiler.

`Ralign` processes nucleic acid sequences from various sequence file formats. Besides, it is possible to define one or more than one codon tables for each sequence or a group of sequences. 18 codon tables are available and cover most needs (see Appendix A). Every input file can be processed using its own codon table. The standard codon table is the universal genetic codon table which fits most cases. These user-defined codon tables are then used by the program for translation and, of course, for detecting the correct start- and stop codons in the nucleic acid sequences. Then the program finds all possible open reading frames which have a previously defined minimal length.

`GenBank` files may contain information about the exact positions of start- and stop codons, the genomic structure of exons and introns or the protein sequence after translation. If some information like this (e.g. regarding exons and introns) is presented in the `GenBank` file, it can be obtained and used as preferred information.

The detected coding regions are translated, using the correct codon table, and the resulting proteins are compared to the protein sequences in the

`GenBank` file, if available. An output file is created which contains all data about the detected open reading frames, either derived by reading the data in the `GenBank` file or as a result of the automatic search done by the program. From this file the user can get information about all open reading frames, about their length, their start and stop, and the lengths of their proteins after translation. Also a second file is created: a `PostScript` output file which gives a graphical representation of the found open reading frames either in one of the three frames or, beyond these, as derived from the `GenBank` file input with all exons and introns.

Overlapping open reading frames are quite frequent in virus genomes. If a certain part of the sequence is coding for two or even three proteins, a decision has to be made which open reading frame is used for the protein alignment. `Ralign` constructs a hierarchy which considers the lengths of the open reading frames. The largest selected coding regions from every sequence get aligned first as a protein alignment.

The program makes a first decision, which coding regions are maintained through the alignments as protein sequences.. The proposed assignment is presented to the user in a file listing the open reading frames chosen for protein alignment. The user now has a possibility to alter this choice.

Then `ralign` computes alignments of the homologous sequence parts. In this older implementation the widely known `clustalw` algorithm is used for this purpose [106, 105]. In contrast, `code2aln` features its own alignment algorithms.

Normally, an alignment like this produces end gaps. End gaps are not penalised by the `clustalw` algorithm, so they occur very often. Here we are aligning only a piece of the genomic sequence; thus this treatment of end gaps is not desireable. Since `clustalw` is used as a ‘black box’ via a system call, we have to resort to a trick: `Ralign` cuts off the end gaps (the frail ends) such that the remaining ’central alignment block’ has no gaps both at the

first and the last position. The sequence pieces that have been cut off are joined to the neighbouring sequence parts before and after the now aligned protein parts of the sequence. In the case of overlapping coding regions these cut off parts are again handled on the level of the proteins that these regions code for (at least the parts that are not covered by the prior alignment). On the other hand, if the neighbouring sequences are non-coding, the cut off sequence pieces are handled directly as nucleic acids.

Then the second protein alignment of the second largest open reading frames (with second priority) is started. The alignment is treated the same way concerning the end gaps.

After having aligned all protein subsequences (all chosen open reading frames), after having removed all end gap containing regions, and after having linked them to the neighbouring parts of the sequences, the alignments of the non-coding regions start. Again the ends of the aligned sequence parts are forced to lie one above the other, if these ends are adjacent to formerly aligned protein parts. That way all parts can be joined smoothly together.

The protein alignments are then reverse translated. At every position where the protein alignments contain a gap of length n, a gap of length $3n$ is inserted into the corresponding nucleic acid sequence at the corresponding site.

Finally, all alignments, either on the level of proteins or nucleic acids, get combined and a resulting alignment output file is created which contains the complete nucleic acid sequence alignment.

In some rare cases `clustalw` will not properly align the tag regions added to suppress end gaps. Gaps inserted into the tags can lead to imperfect removal of the tags and thereby corruption of the sequences.

In a last step the final alignment is checked for such errors. Unfortunately, regarding `ralign` the only recourse is to remove the offending sequence from the alignment.

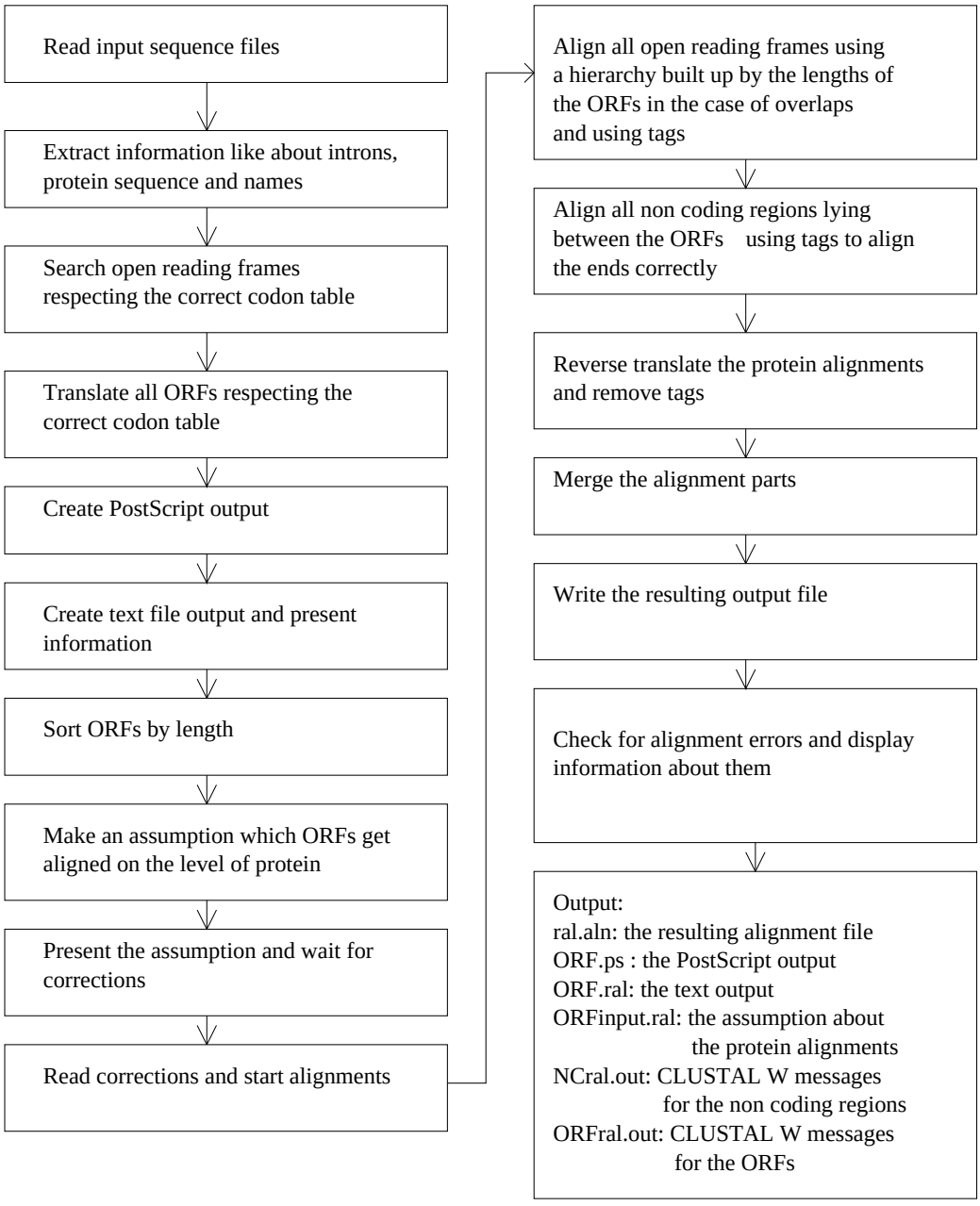


Figure 10: This flow chart shows the main steps of `ralign`.

3.3 Results and Conclusions on ralign

In principle, the `ralign` approach is indeed feasible and leads to significant improvements. The number of small gaps is reduced, and insertions and deletions at the nucleic acid level within a coding region match insertions and deletions at the level of protein sequences.

It was the aim of the implementation of `ralign` to use the information contained in the amino acid sequences translated from the coding regions of a (virus) genome as a means of improving the quality of the alignment of the genomic DNA or RNA sequences. The program `ralign` was the first implementation of this approach.

`Ralign` was written in `ANSI C` and hence easily portable to different operating systems. It was developed on PC's running `Linux` and works well with different `Unix` dialects. The standard multiple alignment package `clustalw` is invoked by means of a system call.

So far, the results showed that `ralign` yields significantly improved alignments, compared to the output of the simple `clustalw` alignments of the same nucleic acid sequences. As examples for the significant reduction of the number of short gaps we have calculated alignments of complete HIV-1 RNA genomes. While the length of the entire alignment was almost the same, we found the number of gaps was reduced by 44% up to 70 % in certain parts of the multiple alignment. The `ralign` approach guarantees that insertions and deletions within coding regions correspond exactly to insertions and deletions at the protein level. A number of very short gaps is therefore often collected into a single longer one.

As a first application for the combination of amino acid and nucleic acid based alignments, the output of `ralign` was used as a basis of the search for conserved RNA secondary structure elements using the program `alidot`. Although the results are surprisingly robust with respect to minor alignment problems, we know that alignment inaccuracies are a substantial problem

```
ADI-MAL         AU-AGUACAUGGCAGAAUAAUGGUGC----AAGA------CU-AA--GU----AAUAGCACAGAGUC---AACUGGUAGUAUCACACUCCCAUG
AE-90CF402      AU-AGUACUUGGAUA---------------AAUGGAACCAUGCAGGAGGUU--AAUGGCACAAACUC---A---GGCAAUAUCACACUUCCAUG
AE-CM240        AU-AAUACUUGCCUAG-----------GAAAUGAAACCAUGGCGGGGUGU--AAUGACAC-----------------UAUCACACUUCCAUG
B-896           AU-AGUACUUGGAAU-------G-------U-UA------CUGGAGGGACA--AAUGGCACUGAAGG---AAAUGACAUAAUCACACUCCAAUG
B-ACH320A       AU-AGUACUUGG------AAUGAUACUGGGAAUGUUA---CUGAAAGGUCA--AAUAACAAUGA------AAAU------AUCACACUCCCAUG
B-BCSG3         AU-AGUACUUGGGCUGGGAAUAAUACUUGGAAUAGUAGUGCUGAAAGGUCA--GAUGACACUGGAGG---AAAU------AUCACACUCCCAUG
B-CAM1          AU-ACUACUUGGCUGUUUAAUGGUACUUGGAAUGAUA---CUGAAGGGUUA--AAUAACACUGAAAG---AAAU------AUUACACUUCCAUG
B-D31           AU-AGUACUUGGAAU-----------------GAUA---CUAAAGAGUCA--AAUAACACAAAU---------GGAACUAUCACACUCCCAUG
B-HIV1AD8       AU-AGUACUUGGAAUUUUAAUGGUACUUGGAAUUUAA---CACAAUCG-----AAUGGUACUGAAGG---AAAUGACACUAUCACACUCCCAUG
B-HXB2          AU-AGUACUUGGUUU---AAUAGUACUUGGAGUA------CUGAAGGGUCA--AAUAACACUGAAGG---AAGUGACACAAUCACCCUCCCAUG
B-JRCSF         AU-AGUACUUGGAAU-------G-------A-UA------CUGAAAAGUCA--AGUGGCACUGAAGG---AAAUGACACCAUCAUACUCCCAUG
B-LAI           AU-AGUACUUGGUUU---AAUAGUACUUGGAGUA------CUGAAGGGUCA--AAUAACACUGAAGG---AAGUGACACAAUCACACUCCCAUG
B-MANC          AU-AGUACUUGGAAUACUGGG---------AAUGAUA---CUAGAGAGUCA--AAUGACACAAAUAA---UACUGGAAAUAUCACACUCCCAUG
B-OYI           AU-AGUACUUGGAAU-----------------GAUA---CUACAAGGGCA--AAUAGCACUGAA---------GUAACUAUCACACUCCCAUG
B-SF2           AU-AAUACAUGGAGGUUAAAU-----------------CACACU-G-----AA-GGAACUAAAGG---AAAUGACACAAUCAUACUCCCAUG
B-WEAU          AU-AGUACUUGGCAUGCUAAUGGUACUUGGAAGAAUA---CUGAAGGGGCA--GAUAACAAU-----------------AUCACACUCCCAUG
B-YU2           -------CUUGG------AAUGAUACUAGAAA--------------GUUA--AAUAACACUGGAAG---AAAU------AUCACACUCCCAUG
B-pNL43         AU-AGUACUUGGUUU---AAUAGUACUUGGAGUA------CUGAAGGGUCA--AAUAACACUGAAGG---AAGUGACACAAUCACACUCCCAUG
D-ELI           AU-AGUACAUGGAAUAU---UAGUGCAUGGAAUAAUAU---UACAGAGUCA--AAUAAUAGCACAAA---CAC---AAACAUCACACUCCAAUG
D-NDK           AU-AGUACAUGGAAU----------CA--GACUAAUAG---UACAGGGUUC--AAUAAUGGCACAG--------------UCACACUCCCAUG
O-ANT70         AUUA-UACCUUU-UCA----------UGUAACGGAACCACCUGUAGUGUUAGUAAUGUUAGUCAAGG------UAACAAUGGCACUCUACCUUG
O-MVP5180       ACUA-UACUUUUAUCAA--------CUGUACAAAGUCCGGAUGCCAGGAGAUCAAAGGGAGCAAUGAGACCAAUAAAAAUGGUACUAUACCUUG
SIVCPZGAB       CUGACAACAUUA-------------------------------------CA--AAUGGCAUU-----------------AUAAUACUGCCAUG
```

Figure 11: This figure shows part of a nucleic acid alignment of HIV-1 sequences (produced by `clustalw`, located in the `env`-gene). The alignment is highly disrupted. We see a lot of gaps, some of them very short, some a little bit longer. An alignment like this is not usable for detection of conserved secondary structure elements where the quality of the input alignments is of great importance. 116 gaps, 39 can not be divided by 3.

```
ADI-MAL         AUAGUACAUGGCAGAAUAAUGGU---GCAAGACUAAGUAAUAGCACAGAGUCAACUGGU----------------------AGUAUCACACUCCCAUG
AE-90CF402      AUAGUACUUGGAUA-----------AAUGGAACCAUGCAGGAGGUUAAUGGCACAAACUCA----------------GGCAAUAUCACACUUCCAUG
AE-CM240        AUAAUACUUGCCUAGGA---------AAUGAAACCAUGGCGGGGUGUAAUGACACU--------------------------AUCACACUUCCAUG
B-896           AUAGUACUUGGAAU-----------------GUUACUGGAGGGACAAAUGGCACUGAAGGA--------------AAUGACAUAAUCACACUCCAAUG
B-ACH320A       AUAGUACUUGGAAUGAUACUGGG------AAUGUUACUGAAAGGUCAAAUAACAAUGAA----------------------AAUAUCACACUCCCAUG
B-BCSG3         AUAGUACUUGGGCUGGG----------------AAUAAUACUUGGAAUAGUAGUGCUGAAAGGUCAGAUGACACUGGAGGAAAUAUCACACUCCCAUG
B-CAM1          AUACUACUUGGCUG-----------------UUUAAUGGUACUUGGAAUGAUACUGAAGGG---UUAAAUAACACUGAAAGAAAUAUUACACUUCCAUG
B-D31           AUAGUACUUGGAAU-----------------GAUACUAAAGAGUCAAAUAACACAAAU--------------------GGAACUAUCACACUCCCAUG
B-HIV1AD8       AUAGUACUUGGAAU-----------------UUUAAUGGUACUUGGAAUUAACACAAUCGAAUGGUACUGAAGGAAAUGACACUAUCACACUCCCAUG
B-HXB2          AUAGUACUUGGUUUAAUAGUACU------UGGAGUACUGAAGGGUCAAAUAACACUGAAGGA--------------AGUGACACAAUCACCCUCCCAUG
B-JRCSF         AUAGUACUUGGAAU-----------------GAUACUGAAAAGUCAAGUGGCACUGAAGGA--------------AAUGACACCAUCAUACUCCCAUG
B-LAI           AUAGUACUUGGUUUAAUAGUACU------UGGAGUACUGAAGGGUCAAAUAACACUGAAGGA--------------AGUGACACAAUCACACUCCCAUG
B-MANC          AUAGUACUUGGAAUACU---GGG------AAUGAUACUAGAGAGUCAAAUGACACAAAUAAU--------------ACUGGAAAUAUCACACUCCCAUG
B-OYI           AUAGUACUUGGAAU-----------------GAUACUACAAGGGCAAAUAGCACUGAAGUA--------------------ACUAUCACACUCCCAUG
B-SF2           AUAAUACAUGGAGG-----------------UUAAAUCACACUGAAGGAACUAAAGGAAAU--------------GAC---ACAAUCAUACUCCCAUG
B-WEAU          AUAGUACUUGGCAU-----------------GCUAAUGGUACUUGGAAGAAUACUGAAGGG------------GCAGAUAACAAUAUCACACUCCCAUG
B-YU2           -----ACUUGGAAU-----------------GAUACUAGAAAGUUAAAUAACACUGGAAGA--------------------AAUAUCACACUCCCAUG
B-pNL43         AUAGUACUUGGUUUAAUAGUACU------UGGAGUACUGAAGGGUCAAAUAACACUGAAGGA--------------AGUGACACAAUCACACUCCCAUG
D-ELI           AUAGUACAUGGAAUAUUAGUGCAUGGAAUAAUAUUACAGAGUCAAAUAAUAGCACAAACACA--------------------AACAUCACACUCCAAUG
D-NDK           AUAGUACAUGGAAUCAGACU---------AAUAGUACAGGGUUCAAUAAUGGCACA----------------------------GUCACACUCCCAUG
O-ANT70         AUUAUACCUUUUCA------UGUAACGGAACCACCUGUAGUGUUAGUAAUGUUAGUCAA-----------------GGUAACAAUGGCACUCUACCUUG
O-MVP5180       ACUAUACUUUUAUCAAC---UGUACAAAGUCCGGAUGCCAGGAGAUCAAAGGGAGCAAUGAG------------ACCAAUAAAAAUGGUACUAUACCUUG
SIVCPZGAB       ---------------------------------------ACUGACAACAUUACAAAUGGC--------------------AUUAUAAUACUGCCAUG
```

Figure 12: This figure shows the same region and demonstrates the advanced capability of the `ralign` alignment algorithm environment to obtain better alignments. The number of gaps could has been reduced dramatically as a result of the combined amino acid and nucleic acid based alignment technique. Most lines still have only two remaining gaps. 46 gaps (60% less gaps).

when dealing with more diverse data sets.

Hence, the advantages of `ralign` have been demonstrated formerly in the example of viral sequences: A number of probably significant secondary structure elements were predicted from a sample of 14 sequences of pregenomic RNA of different isolates of the Human Hepatitis B virus.

The most important structure among these, the well-known ϵ-structure, was correctly predicted by `alidot` from the `ralign` alignment of human hepatitis B virus genomes. Also a number of secondary structure elements have been detected by this method that have not been described in the literature so far.

But `ralign` has some serious disadvantages too: First, to create a hierarchy of open reading frames just based on length is highly arbitrary. In many cases this may emphasize the wrong point. And we have to keep in mind that this is very likely to produce artefacts. Furthermore, `ralign` uses only one frame in case of overlapping genes. But overlapping open reading frames are quite frequent in virus genomes. If a certain part of the sequence is coding for two or even three proteins, a decision has to be made which open reading frame is used for the protein alignment. Also this is arbitrary.

In many cases we can see significant differences in the genetic structure regarding the number and order of various open reading frames even between very closely related sequences. This makes it difficult to decide which coding regions correspond to each other in the various sequences.

Two of the major problems are as follows: First, the junctions between coding and non-coding regions and the way `ralign` organizes them surely produce more or less artefacts. And, second, `ralign` is not sufficiently stable against handling errors.

The `ralign` procedure of combining amino acid and nucleic acid based partial alignments has a number of shortcoming that were the reason for choosing a different approach in the code2aln project.

The alignment might use alternative hierachies of open reading frames. In particular, it might be advantageous to use the more conserved (instead of the longer) ORFs with higher priority. In general, the current mechanism for detecting homologous reading frames is not very robust. The procedure could probably be improved significantly by explicitly comparing the pairwise sequence homologies of all ORFs.

Even in the case of overlapping reading frames, it might be useful to consider the alignments of each reading frame and to combine them, for instance, using local optimzation rules, instead of selecting a single reading frame.

In some cases, `clustalw` introduces gaps into the 'tagging sequences' that are used to force sequence ends to lie one above the other when combining the alignments of the individual open reading frames. At present, `ralign` returns an error message if this problem occurs. While this a rare problem, a more reliable procedure would be desireable.

`Ralign` invokes `clustalw` using a system call. One might want to use different multiple alignment procedures instead, if only for the sake of comparison. On the other hand, system calls cause a substantial loss in performance, so that it would be desirable to have the multiple alignment procedure available as a run-time library.

Finally, `ralign` at present does not produce auxiliary information, such as guide trees, that can be used to trace the process of the multiple alignment.

4 The code2aln Project

4.1 Code2aln in Short

For usage of alignments by processes like `alidot` the best possible quality of the input alignment is of crucial importance. In this thesis it is shown that the alignments are indeed improved in various means either by integration of genetic information into the scoring function, or by combination of several protein and nucleic acid alignments to one final alignment. The number of gaps is reduced significantly (at least in the case of the combined amino acid and nucleic acid based alignments as implemented in the older program `ralign`) [101, 102].

`Code2aln` processes complete nucleic acid sequences without cutting sequences, produces 'real' multiple nucleic acid alignments and uses the information on coding and non-coding regions as part of the scoring function, in order to prevent the problem of higher sequence divergency on the level of nucleic acid as compared to the alignments of the underlying protein sequences in the case of coding at a certain region of the input nucleic acid sequences.

`Code2aln` reduces the number of gaps not as effectively as when `ralign` is used. But `code2aln` produces better results in terms of avoiding artefacts like those which occur at the junctions between coding and non-coding regions when the chosen alignment procedure is `ralign`, because forcing the sequence ends to lie one above the other, as implemented in `ralign`, is highly arbitrary; also creating a hierarchy of open reading frames just based on length is often just at random. In many cases this emphasizes the wrong point.

Furthermore, `code2aln` uses not only one frame in case of overlapping genes like `ralign`. In the case of `ralign` we loose information that could be used for further improvement of the alignment. This is a great advantage of `code2aln`, especially in the applications to viruses, because overlapping

open reading frames are quite frequent in virus genomes. If a certain part of the sequence is coding for two or even three proteins, a decision has to be made, in the case of `ralign`, which open reading frame is used for the protein alignment. Also this is as arbitrary as the fact, that we often also have to evaluate significant differences in the genetic structure regarding the number and order of various open reading frames even between very closely related sequences. This makes it difficult to decide which coding regions correspond to each other in the various sequences. `Code2aln` bypasses this problem effectively.

4.2 More Complex Scoring Systems

Describing dynamic programming and its scoring by usage of the finite state automaton model has one important advantage: it is easy to see how to generate new types of algorithms and scoring. The idea behind this representation is that there may be more information about the sequences aligned than only the type of nucleic acid. This information can be used for further improvement of the alignment. The progress of the alignment can be represented as a walk through the diagram from one state to another via the transition paths.

For example, there may be high fidelity regions without gaps, corresponding to a certain match state A, while lower fidelity regions with gaps can correspond to another match state B. Each state can bear its own substitution scores which should be chosen to reflect the expected degrees of similarity in the different regions.

Another possibility constitutes the algorithm proposed here: it is an example for a type of more complex scoring system without introducing a new match state (and a new scoring matrix and a new recurrence relation) which has a high cost of computational resources, especially for long sequences. In

the following, this finite state automaton gives a diagram of the relationships and transitions between the states and scoring terms used for the `code2aln` algorithm. The progress of the alignment can be understood as a course through the finite state automaton model from one state to another via the transition paths. We see states for matches and mismatches on the level of nucleic acid, states for insertions and deletions, gap open- and extension penalties and the score increments for matches and mismatches on nucleic acid level. But, additionally, three score factors (increments) are possible for the amino acid scores in three different frames to add a certain value to the nucleic acid score in case that the nucleotide is part of one or more codons in one or more reading frames. Hence, nucleic acid sequence positions that are part of codons are weighted significantly higher than positions that are not in open reading frames (or exons).

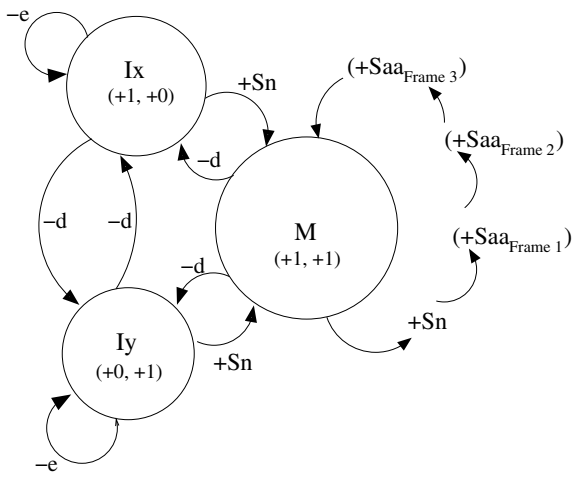


Figure 13: This finite state automaton gives a diagram of the relationships and transitions between the states and scoring terms used for the `code2aln` algorithm. M means (mis)match, Ix and Iy are insertions and deletions, Sn is the increment of nucleic acid score, and d and e represent the gap open and extension penalties. The Saa terms represent the amino acid scores in three different frames, they add a certain value to the nucleic acid score in case that the nucleotide is part of one or more codons.

In principle, `code2aln` represents a classical Gotoh-type version of the Needleman-Wunsch algorithm with affine gap penalties and three scoring matrices, that means, three recurrence relations. The special point is the scoring function which is more complex regarding the extensions for amino acid sequence information.

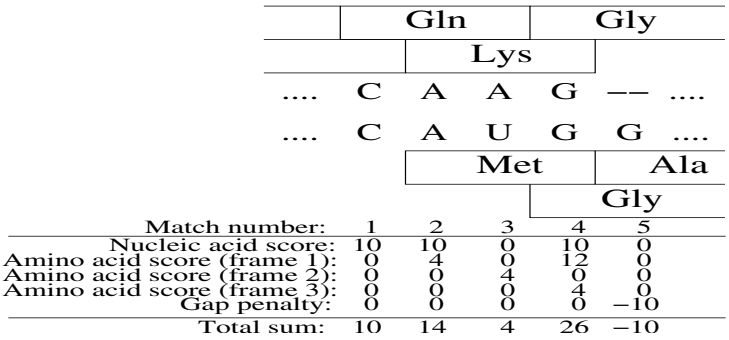
The scoring function of the `code2aln` algorithm as defined by the finite state automaton is:

$$S = S_N + \sum_{i=1}^{3} S_{A_i} \tag{15}$$

where S is the total score of the considered pair of nucleotides, S_N is the nucleic acid part of the scoring, and the sum of the three amino acid scores S_{A_i} reflects the coding of the pair of nucleic acids as the first, second, and/or third position in a base triplet.This type of scoring takes frameshifts at least partially into account, because an amino acid score is only added if the two nucleic acid letters that are compared lie in the same codon positions, see Figure 14.

In our applications to viral sequences we often see the fact that the sequences under consideration code for proteins in one or more than one frame(s). So, we can use this information as an heuristic approach to specially adapt the scoring to this case, if possible. We do this by adding the amino acid scores for all amino acids that are encoded by the considered nucleotide in the way that the nucleotide is part of the codon of the amino acid. Three codons are possible, thus three amino acids and three additional scores. This means that the scoring in general is additive. The higher the degree of coding is for a considered nucleotide, the higher is the possible maximum score. This is biologically meaningful, in that, for example, a position that is double encoded, also counts more. In fact, positions which code for one or more amino acids as part of a codon, are supposed to be higher conserved throughout evolution. We can see this in the lower sequence heterogenity on

the level of proteins as compared to the underlying nucleic acid sequences (see before).



Match number:	1	2	3	4	5
Nucleic acid score:	10	10	0	10	0
Amino acid score (frame 1):	0	4	0	12	0
Amino acid score (frame 2):	0	0	4	0	0
Amino acid score (frame 3):	0	0	0	4	0
Gap penalty:	0	0	0	0	−10
Total sum:	10	14	4	26	−10

Figure 14: This figure gives an example of the local scoring of an alignment of two short nucleic acid fragments that code in none, one or two frames. The local scores of the 5 matches of positions are calculated as follows:
Match 1: A CC match, one of the nucleotides is part of two codons (above), the other is not coding (below). The score for this match is only the nucleotide match score 10.
Match 2: An AA match, the nucleotide above is coding for Lys as the first position in the codon, and for Gln as the second position. The nucleotide below codes for Met as the first codon position. Thus, the total score for this match is 14: The nucleotide match score 10 + the amino acid mismatch score for Met-Lys which is 4.
Mismatch 3: An AU mismatch, the A codes for Gln as position 3 in the codon, and for Lys as position 2. The U codes for Met as codon position 2. The total score is 0 (for the nucleotide mismatch) + 4 (for the Met-Lys mismatch) = 4.
Match 4: A GG match, the nucleotide above codes for Gly as position 1 and for Lys as position 3 in the codons, the nucleotide below is also position 1 in Gly, and position 3 in Met. Thus, the score for this match is: 10 (nucleotide match) + 12 (Gly-match) + 4 (Lys-Met-mismatch) = 26.
Gap 5: G matches to a gap. The score is the gap open penalty -10.

Our scoring function also considers frame shifting as a side effect: an amino acid score is only added to the total score if we look at the same position within a codon triplet. Thus, if frame shifting alters the codon and destroys the open reading frame, this is respected inherently, simply because such comparisons of shifted positions are not allowed.

But the major problem of more complex scoring functions is as follows: In the case of an alignment with a simple parameter set, it is possible, at least theoretically, to optimize the parameter set by using a training set of alignments and to adapt the parameters. This is a process of learning. The resulting (small) set of parameters and usage of them in alignments should then be quite robust and universal, even for test sets that are very different from the training set of alignments. In the case of larger parameter sets, this optimization is not possible, computationally not feasible. Hence, we use theoretical approaches what might be useful for the special current alignment problem: for example, we add protein sequence information when we know that protein sequences are of dominant importance in our alignment of nucleic acids. But the problem is that we, first, have to avoid the effect of overlearning, and that we, second, have no chance to overview the complete space of parameter combinations and weighting, and their resulting alignments, because our training set is too small in any case for our larger parameter set. A larger parameter set needs much more training. So, we do not know, whether our parameter set is useful for *all* or *most* sequences, or only for a very small subset of sequences and their according alignments.

Overlearning is another insight to the problem. It is known from work with neural nets that, if we do too much learning with a special (in this case too small) training set, we specialize ourselves too much to our training set, and the parameters fail when used with a diverse test set, that is different from all solutions (alignments) in our training set.

In principle, every approach of using parameter sets that can not be

optimized widely absolutely, is heuristic and arbitrary. This means for this work that it is not possible to build too complex scoring functions (we did testing) and we have to restrict ourselves to more 'simple' extensions of the scoring. The addition of amino acid scores for all three possible frames for each nucleotide, was the most that is feasible to get better alignment results that are not partially or totally at random, because of this problem mentioned above.

4.3 The code2aln Algorithm

The improved multiple alignment procedure using genetic information about coding and non-coding regions in the input nucleic acid sequences as a function of the scoring system, is made available in the program called `code2aln`. The source code of the package is written in the C programming language and will run on computers with an ANSI C compiler.

The idea behind the usage of information about open reading frames as it is implemented in `code2aln` is that sequences vary less on the level of protein than on the level of nucleic acid, because most amino acids are encoded by more than one codon (base triplet) and some different nucleic acid sequences can produce the same protein sequence after translation. This coding information is part of the scoring function and leads to better alignments.

`Code2aln` technically processes the input nucleic acid files as described in Appendix B. There are various possible input sequence file formats. `Code-2aln` automatically detects the types of the various input sequence files and handles them accordingly. Also files that contain only the sequence in one or more lines can be handled. Besides, it is possible to define one or more than one codon tables for each sequence or groups of sequences. Default is the universal genetic code which fits most cases. The codon tables are important, because they ensure that fine adjustment of detection and translation of

coding regions during the process of aligning is possible for various organism groups or their fitting mitochondrial sequences.

`GenBank` files contain information about the exact positions of coding regions, the genomic structure of exons and introns, or the protein sequence after translation. If some information like this (e.g. regarding exons and introns) is present in the `GenBank` file, it is used for supporting the alignment algorithm, especially the scoring. In any case, either if the `GenBank` file contains information about coding parts, or if there are no supporting data (e.g. in FASTA format files), `code2aln` searches open reading frames respecting the correct codon table. These data may then be compared one to each other for critical analysis of the input.

`Code2aln` also produces some output files which contain the data about sequences and coding and provide a graphical representation of the genomic structures of the input nucleic acid files. The open reading frames are shown either within the three frames or, beyond these, as derived from the `GenBank` file respecting all introns and exons.

Especially in the case of viruses, one has often to handle circular genomes which sometimes have coding regions that get torn into two pieces when the genome is cut for sequencing and treatment as data base content. In these cases `code2aln` automatically rejoins the pieces to get the full length of the open reading frame for further processing. Also if exons and introns exist as part of the input sequences, `code2aln` simulates splicing and produces the correct protein sequences after translation.

Then the pairwise nucleic acid alignments of all sequences get started. All knowledge about coding regions is incorporated. `Code2aln` uses an affine gap scoring, which differentiates between gap open- and gap extension penalties. The (mis)match scoring system is a combination of nucleic acid score and amino acid score (in all three frames) in the case of coding regions (see above). As an amino acid scoring matrix a highly universal matrix is used; it fits the

```
File name: Test2/NewMAUS1.gbf
Codon Table: univ
Sequence name: > MUSMHQAMB   [2875bp]   Number: 1
GenBank file information derived exons, if available:
  START - STOP:
  200 - 272
  502 - 768
  1028 - 1303
  1871 - 2155
  2255 - 2388
GenBank file derived CDS marked open reading frames:
  START - STOP:
  200 - 272
  502 - 768
  1028 - 1303
  1871 - 2155
  2255 - 2388
Open reading frames derived by code2aln automatic search:
  START - STOP:
  1066 - 1356
  1807 - 2163
```

Figure 15: An example for the text file output of `code2aln`. Various data about the input sequences are shown: the names of files and sequences, the codon table used, the start and stop codons of found and Genbank derived open reading frames, and, if available, the exon data.

most protein types and organism groups.

The next step after the initial alignments is producing a guide tree which controls the subsequent (multiple) profile alignments. Again a file output is created that presents the clustering of sequences in order to build up the guide tree. It is important to note that the guide tree in this context tells nothing about the evolutionary distances between the sequences. It is only usable to get an order of the following profile alignments, to define which smaller clusters are merged to a larger one.

The process of the profile alignments runs according to the guide tree, and genetic information, affine gap penalties, and the suited nucleic acid and amino acid scorings are used throughout all alignments. Finally, the resulting alignment is checked for errors and the output file is written.

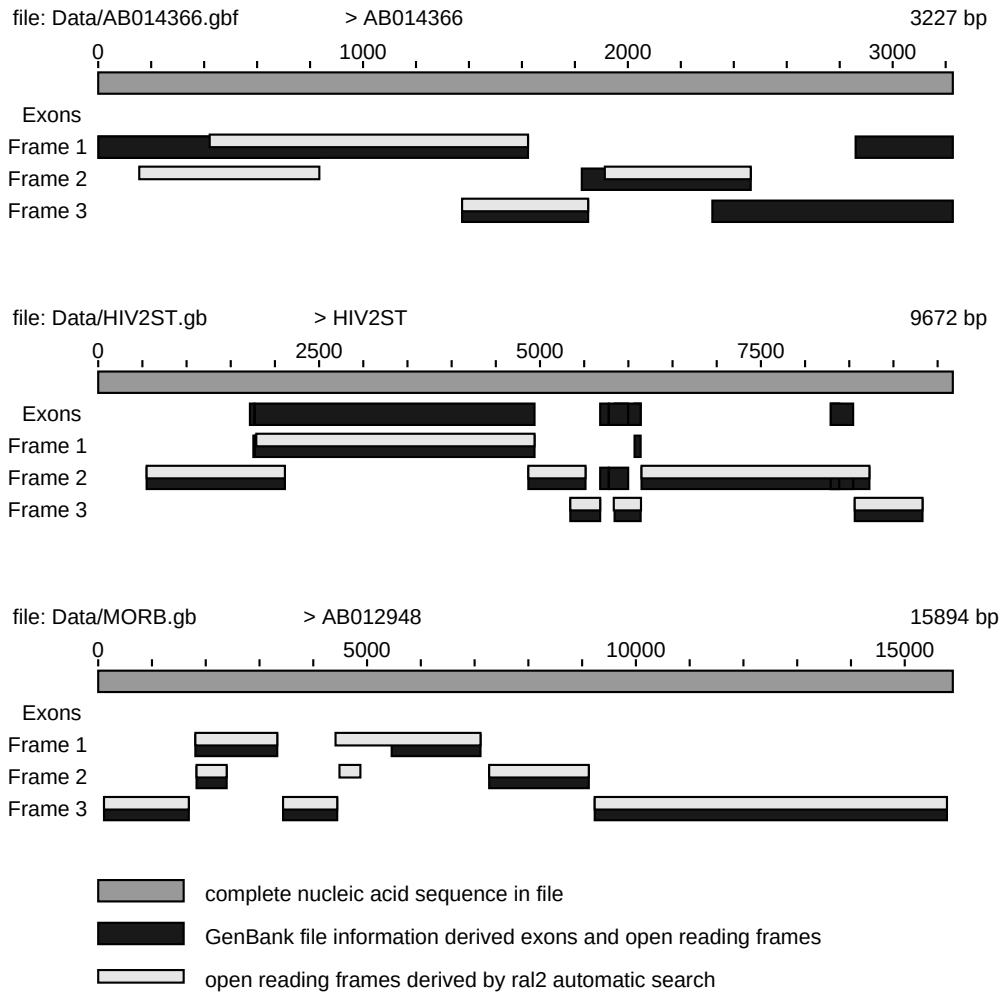


Figure 16: An example for the `PostScript` output of `code2aln`. The figure shows a graphical representation of the found open reading frames of three unrelated sequences. The genomes of hepatitis B virus, HIV1 and the measles virus.

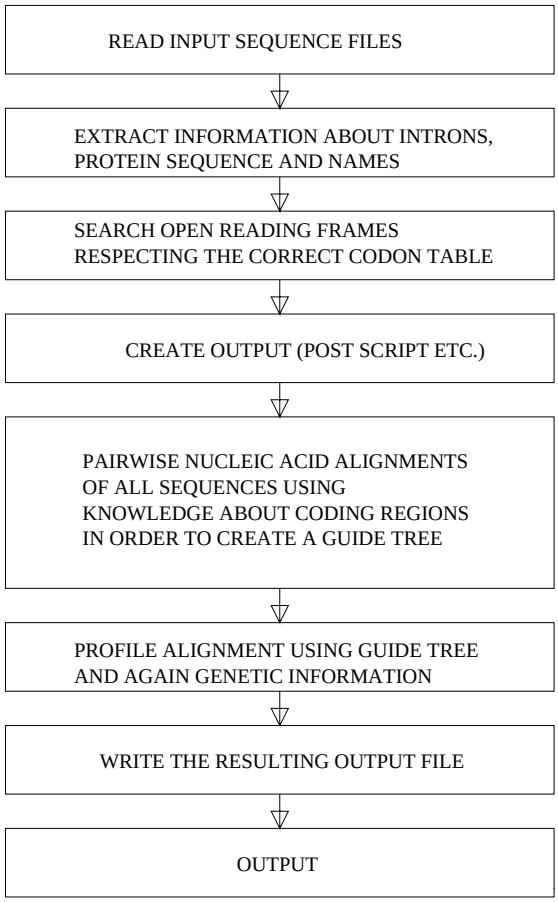


Figure 17: This flow chart gives a graphical representation of the main steps of `code2aln`. It produces a priori only nucleic acid alignments; the scoring function uses information about coding and non-coding regions and the type of amino acid.

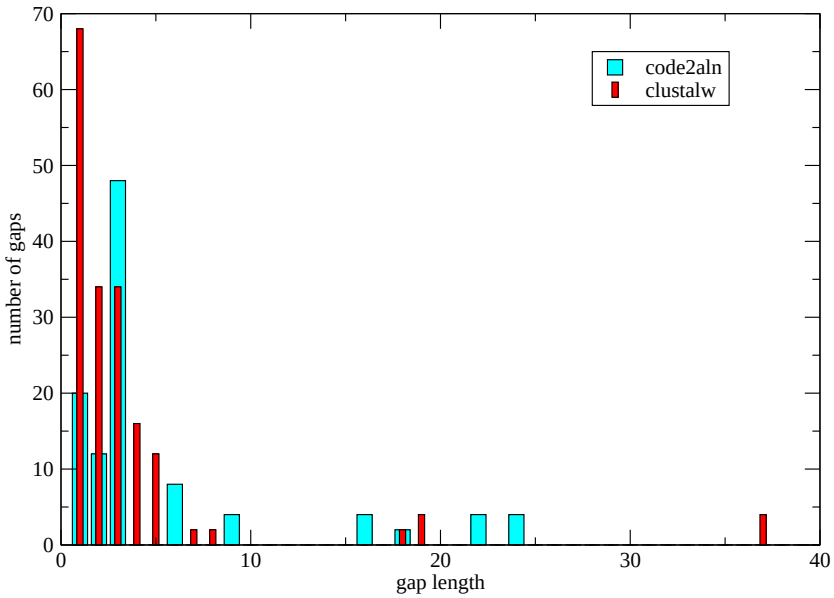


Figure 18: This chart shows the distribution of the types of various gap lengths after alignment of Levivirus genomes using `clustalw` and `code2aln` (see below). `Code2aln` produces a higher fraction of gaps with a length that can be divided by 3 (`Clustalw`: 182 gaps, 40 can be divided by 3; `code2aln`: 110 gaps, 66 can be divided by 3). This shows the strong tendency of `code2aln` not to disrupt codons.

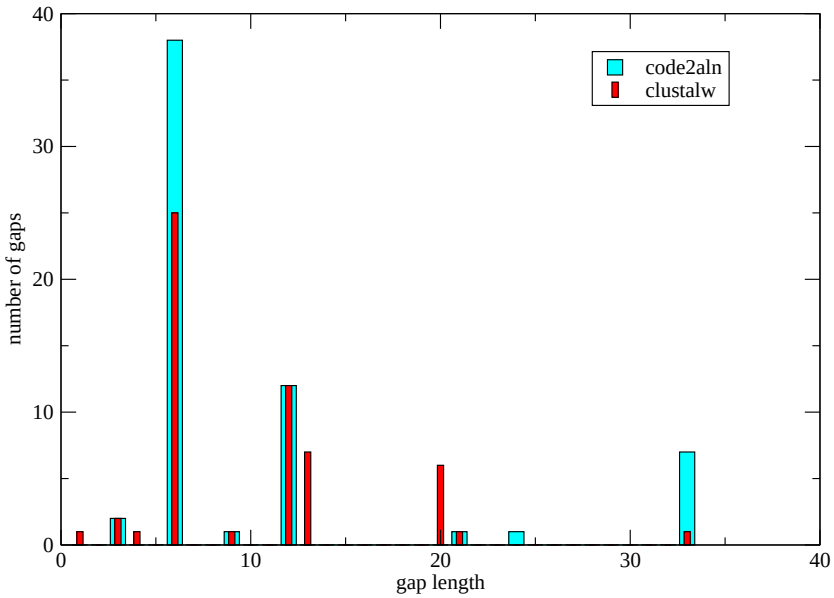


Figure 19: This chart shows the distribution of the types of various gap lengths after alignment of hepatitis B virus pregenomes using `clustalw` and `code2aln` (see below). `Code2aln` produces without exception gaps with a length that can be divided by 3 (`Clustalw`: 57 gaps, 42 can be divided by 3; `code2aln`: 62 gaps, all can be divided by 3). This shows the strong tendency of `code2aln` not to disrupt codons.

4.4 An Example Program Run

INPUT FILES: 12 sequences of mouse (Mus musculus), manually edited (random mutations, insertions and deletions were added) for reasons of demonstration and testing.

PROGRAM CALL: `code2aln seq*`

OUTPUT (STDOUT):

```
Encoding.....Done
Pairwise alignments.....
Alignment: 1   Sequ. (1 : 2)   Score: 71716
Alignment: 2   Sequ. (1 : 3)   Score: 71047
Alignment: 3   Sequ. (1 : 4)   Score: 70957

               ( .... )

Alignment: 65   Sequ. (10 : 12)   Score: 74646
Alignment: 66   Sequ. (11 : 12)   Score: 74716
Profile alignments.....
Alignment: 67   Cluster: 12   Score: 74933
Alignment: 68   Cluster: 13   Score: 74716

               ( .... )

Alignment: 76   Cluster: 21   Score: 35763
Alignment: 77   Cluster: 22   Score: 33775
See 'cluster.txt' and 'info.txt' for information about profiles.
```

OUTPUT FILES: `cluster.txt, info.txt, ORF.ps, aln.aln`

cluster.txt:

```
Cluster: 0   Sequence(s): 1
Cluster: 1   Sequence(s): 2
Cluster: 2   Sequence(s): 3
Cluster: 3   Sequence(s): 4
Cluster: 4   Sequence(s): 5
Cluster: 5   Sequence(s): 6
Cluster: 6   Sequence(s): 7
Cluster: 7   Sequence(s): 8
Cluster: 8   Sequence(s): 9
Cluster: 9   Sequence(s): 10
Cluster: 10   Sequence(s): 11
Cluster: 11   Sequence(s): 12
Cluster: 12   Sequence(s): 3, 4
Cluster: 13   Sequence(s): 10, 11
Cluster: 14   Sequence(s): 12, 10, 11
Cluster: 15   Sequence(s): 9, 12, 10, 11
Cluster: 16   Sequence(s): 7, 8
Cluster: 17   Sequence(s): 5, 6
Cluster: 18   Sequence(s): 9, 12, 10, 11, 5, 6
Cluster: 19   Sequence(s): 2, 3, 4
Cluster: 20   Sequence(s): 9, 12, 10, 11, 5, 6, 2, 3, 4
Cluster: 21   Sequence(s): 7, 8, 9, 12, 10, 11, 5, 6, 2, 3, 4
Cluster: 22   Sequence(s): 1, 7, 8, 9, 12, 10, 11, 5, 6, 2, 3, 4
```

info.txt:

```
File name: seqA.seq
Codon Table: univ
Sequence name: > MUSB1   [3221bp]   Number: 1
GenBank file information derived exons, if available:
  no exons
GenBank file derived CDS marked open reading frames:
  Could not read GenBank file information.
Open reading frames derived by code2aln automatic search:
  START - STOP:
  4 - 309
  986 - 1243
  1247 - 1483
  2190 - 2564
  2480 - 2740
  ( .... )
```

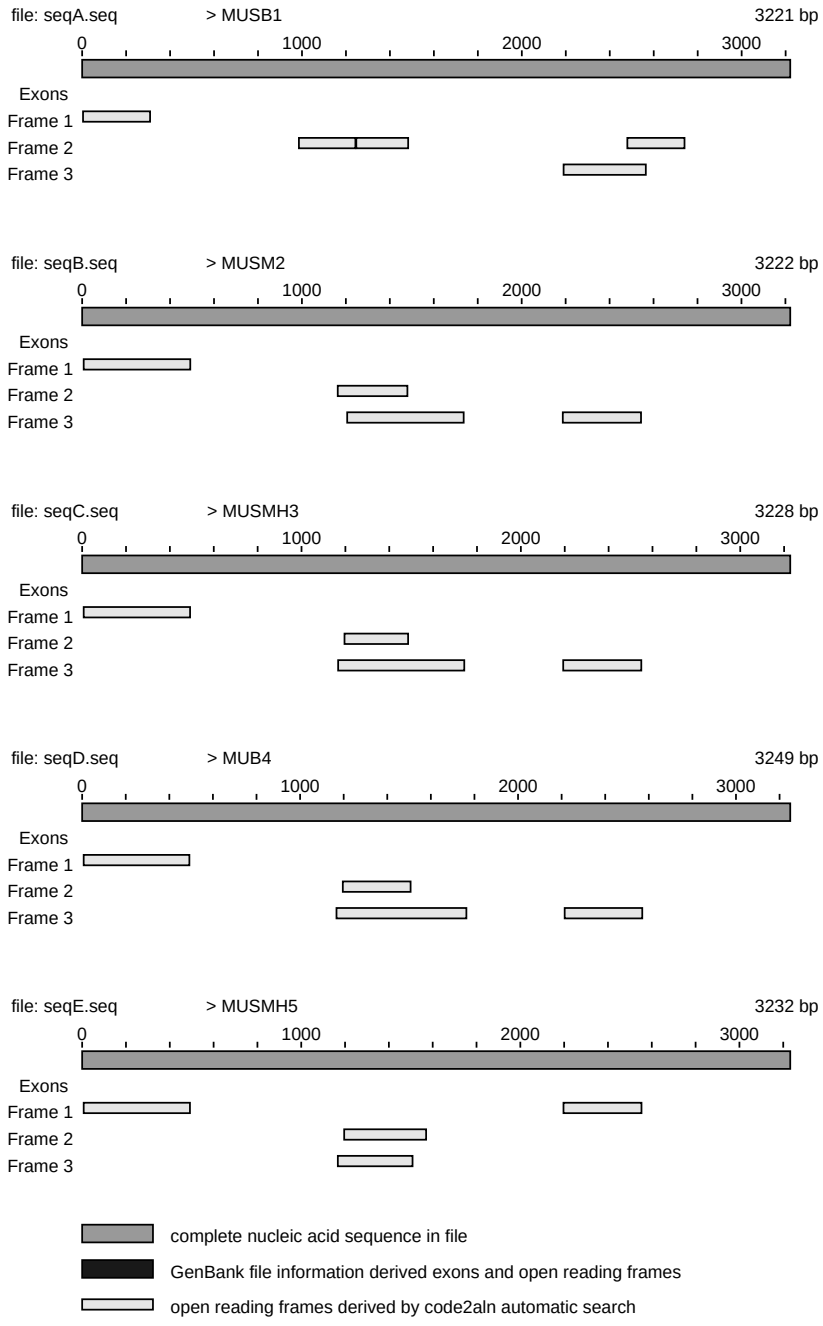


Figure 20: `ORF.ps`, the `PostScript` output of `code2aln`, page 1.

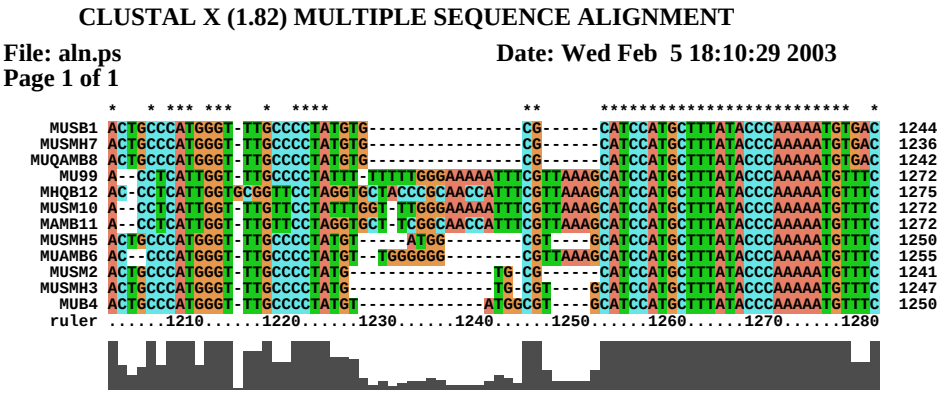


Figure 21: aln.aln: the resulting alignment. This short cutout of the resulting alignment gives an impression about the function of `code2aln`. One partially divergent region is shown, as represented by `clustalx`. We see 6 gaps with lengths that can be divided by 3. In contrast, the alignment below (derived by `clustalw`) has only one gap of this type. Thus, `code2aln` has produced the sixfold amount of those gaps. This result is representative and shows the much stronger tendency of `code2aln` *not* to disrupt codons in coding regions.

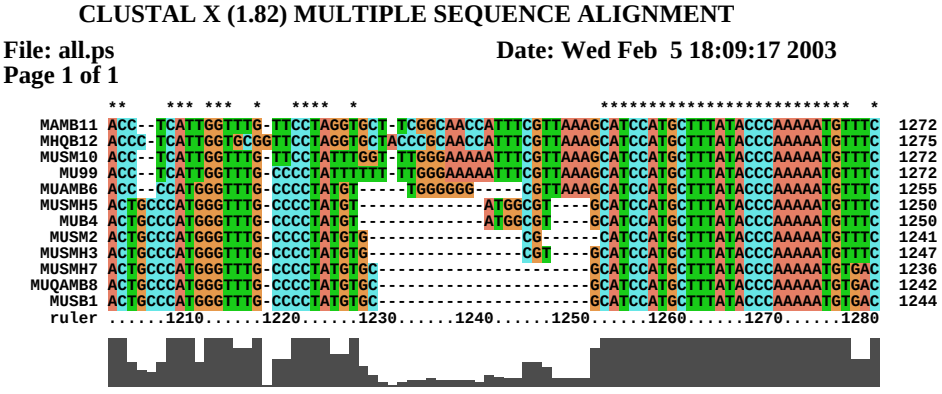


Figure 22: For comparison, the same part of the resulting alignment, after alignment with `clustalw`, as represented by `clustalx`. One gap length can be divided by 3.

5 An Example for an Application

Much research is performed to find methods for detecting and identifying significant common sequence and structure motifs in a set of nucleic acid sequences [34, 36, 43]. Various implementations of algorithmic approaches exist that are supposed to find solutions to this problem [76, 77].

The program `alidot` detects conserved RNA secondary structure elements in relatively small sets of RNAs by combining multiple sequence alignments and secondary structure predictions. Both a (good) sequence alignment and secondary structure predictions for each sequence in the alignment must be provided as inputs [51, 53].

The starting point of the analysis of conserved secondary structure elements is a list of all predicted base pairs. This list will in general not be a valid secondary structure, because it is possible that one certain nucleotide takes part in more than one base pairs and it is possible that base pairs cross.

The quality of the input sequence alignment is of crucial importance. So the approach of using genetic information in the combined amino acid and nucleic acid based alignments, or as part of the scoring function, should provide one with better sequence alignments [101].

The basic idea behind both `alidot` is to sort the individual base pairs by their *credibility* and to reduce the number of entries in the list by subsequent filtering steps until only those secondary structure elements are left that are consistently predicted. Of course the sorting procedure is very important. For each predicted base pair the nucleotides occurring in the corresponding positions in the sequence alignment are stored. A sequence is *non-compatible* with a base pair *(i.j)* if the two nucleotides at positions i and j would form a non-standard base pair such as **GA** or **UU**. A sequence is *compatible* with base pair *(i.j)* if the two nucleotides form either one of the following six combinations: **GC**, **CG**, **AU**, **UA**, **GU**, **UG**.

When different standard combinations are found for a particular base pair *(i.j)* we may speak of *consistent* mutations. If we find combinations where both positions are mutated at once we have *compensatory* mutations. The occurrance of consistent and, in particular, compensatory mutations strongly supports a predicted base pair, at least in the absence of non-consistent mutations.

We call a base pair *(i.j) symmetric* if j is the most frequently predicted pairing partner of i and if i is the most frequently predicted pairing partner of j. Foe each sequence position i there is at most one symmetric base pair involving i.

In the first step of `alidot`, a list of 'believable base pairs' is extracted from the set of all pairs which are contained in the input. In this first processing step, all but the most frequent pair *(i.j)* for each base i is removed. The remaining list is then sorted according to some hierarchical criteria like: (i) the more sequences are non-compatible with *(i.j)*, the less credible is the base pair, (ii) symmetric base pairs are more credible than other base pairs, (iii) a base pair with more consistent mutations is more credible, (iv) base pairs are more credible with smaller values of a certain pseudo-entropy which is derived from the frequencies f_{ij} with which *(i.j)* is predicted in the sample of sequences and which is a measure for the reliability.

As the next step the sorted list is reduced by running through it and removing all base pairs that cross with higher ranking ones and hence would not yield a valid secondary structure.

The resulting secondary structure will, in general, still contain ill-supported base pairs. These are removed by three subsequent filtering steps. First all pairs are removed that have more than two non-compatible sequences, as well as pairs with two non-compatible sequences adjacent to a pair that also has non-compatible sequences.

Next all isolated base pairs are omitted. The remaining pairs are col-

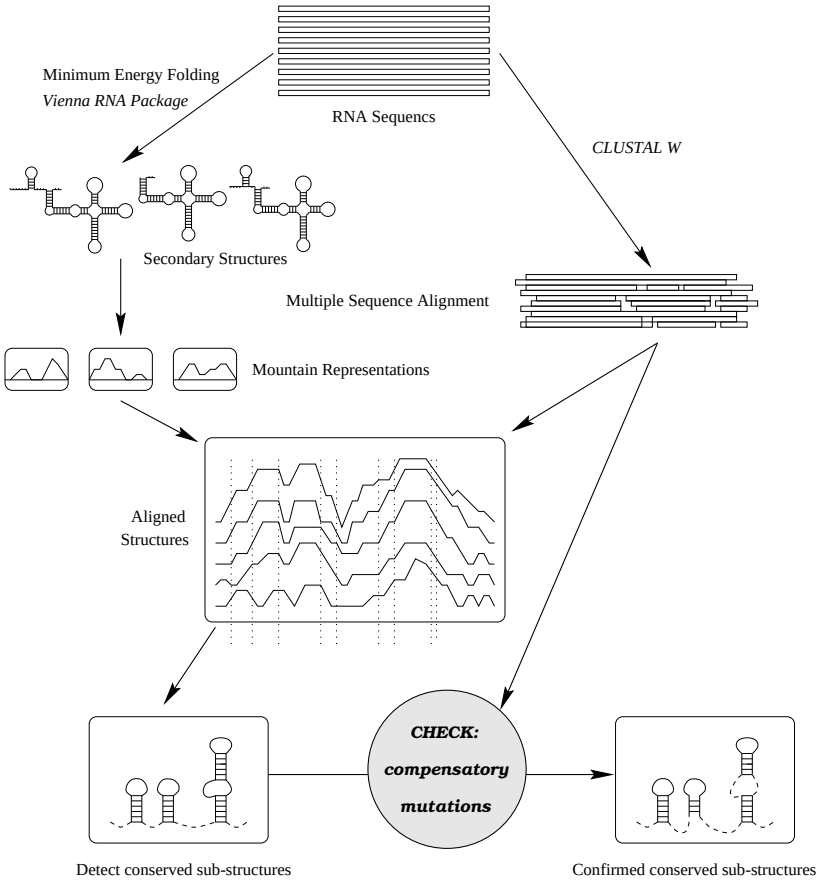


Figure 23: Scheme of the secondary structure analysis of viral genomes. Sequences are aligned using a standard multiple alignment procedure. Secondary structures for each sequence are predicted and gaps are inserted based on the sequence alignment. The resulting aligned structures can be represented as aligned mountain plots. From the aligned structures consistently predicted base pairs are identified. The alignment is used to identify compensatory mutations that support base pairs and inconsistent mutants that contradict pairs. This information is used to rank proposed base pairs by their credibility and to filter the original list of predicted pairs.

lected into helices. Only those helices are retained that satisfy the following conditions: (i) the highest ranking base pair must not have non-compatible sequences, (ii) for the highest ranking base pair the product of the mean probability and the number of different pairing combinations must be greater than 0.3, (iii) if the helix has length 2, it must not have more non-compatible sequences than consistent mutations.

The search algorithms for detecting conserved RNA structure elements are based on both a multiple sequence alignment and predicted secondary structures. The multiple sequence alignment is contained in a single file in `clustalw` format. The predicted secondary structures are contained in individual files (e.g minimum free energy files or base pairing probabilities produced by RNAfold as part of the `Vienna RNA package`).

Both text and `PostScript` output is produced. The text output contains some statistics, all base pairing data and the conserved structure in bracket notation. The `PostScript` output are dot plots, a simple graphical representation of structures, where each base pair corresponds to a small square in a matrix of size sequence length $\times$ sequence length. The size and color of this 'dot' are used to encode additional information like the frequency of prediction and the number of different consistent base pairs. The dot plots finally were used to construct the secondary structure graphs and the Hogeweg mountain representations. Before producing the secondary structure graphs the consensus sequences were calculated.

In principle, the development and implementation of computational methods capable of reliably predicting functional structural elements on the basis of sequence information is necessary to provide advanced benefits in terms of our understanding of the relationship between sequence and structure [21]. Thus, much research is performed to find conserved secondary structure elements [79, 91], e.g. as part of the pregenome of hepatitis B viruses or the genomes of Leviviridae.

For such a procedure a high quality alignment of nucleic acid sequences is of major importance and methods that improve the input sequence alignments will become more and more essential.

Virus genomes sometimes contain various open reading frames within their genomes. The lengths of small virus genomes can vary from some 3500 bp as in hepatitis B up to about 20000 bp as in the case of Ebola. The typical genome size is about 10000 bp. The genomes can consist of single-stranded or double-stranded DNA or single- or double-stranded RNA. Also some viruses with relatively large double-stranded genomes exist, such as the pox viruses. Retrotranscribing viruses are the retroviruses (e.g. HIV), the hepatitis B viruses as well as caulimoviruses which have a DNA genome but use RNA as an intermediate during their replication. RNA viruses have enormously high mutation rates of up to 10^{-3} per position and replication. This makes them a reasonable candidate for the process of `alidot`. The number of the viral open reading frames depends on the type of virus considered. In addition, the organization of virus genomes is extremely variable. Overlapping open reading frames are possible and, in fact, frequent, hence one part of the nucleic acid sequence codes for more than one protein in different frames. Theoretically, three open reading frames can be covered by the same nucleic acid sequence in all three possible reading frames. This possibility is actually realized neither in the hepatitis B virus nor in Levivirus (see Figures 25, 41, and 42). Both groups contain overlapping open reading frames in 'only' two frames. In addition, various non coding regions can exist in a certain virus genome [12].

6 Hepatitis B Virus

6.1 The Morphology of the Hepatitis B Virus

The Hepatitis B virus infects mammals and members the family of the Hepadnaviridae. Their morphology is spherical, occasionally pleomorphic, 40-48 nm in diameter (40-42nm in the case of HBV, 46-48nm in the case of duck HBV) but with no evident surface projections. The outer 7 nm thick, detergent-sensitive envelope contains the surface antigens and surrounds an icosahedral, 27-35 nm diameter (HBV: 27nm, DHBV: 35nm) nucleocapsid core with 180 capsomers. The core is composed of one major protein species, the core antigen, and encloses the viral genome (DNA) and some associated minor proteins. Some viruses occur with filamentous forms of variable length and 22 nm diameter and others with spherical 16-25 nm structures that lack cores [12].

The genome consists of a single molecule of non-covalently closed, circular DNA that is partially double-stranded and partially single-stranded (see Figure 25). The G+C content is about 48 %. One strand in negative sense (complementary to the viral mRNAs) is full-length (3100 -3400 nt), the other is shorter and varies in size. In the hepatitis B virus genome, which is a member of the Orthohepadnavirus genus, the full-length negative sense DNA strand has a nick at a unique site corresponding to a position 242 nt downstream from the 5' end of the positive sense strand. The ssDNA may represent up to 60 % of the circle. In the second genus (Avihepadnavirus, e.g. duck hepatitis B) of the family Hepadnaviridae the nick in the negative sense DNA is about 50 nt from the end and genomes may be fully double-stranded. The uniquely located 5' ends of the two strands overlap by about 240 nt so that the circular configuration is maintained by base pairing of cohesive ends. The 5' end of the negative sense DNA has a covalently attached terminal protein. The 5' end of the positive sense DNA has a covalently

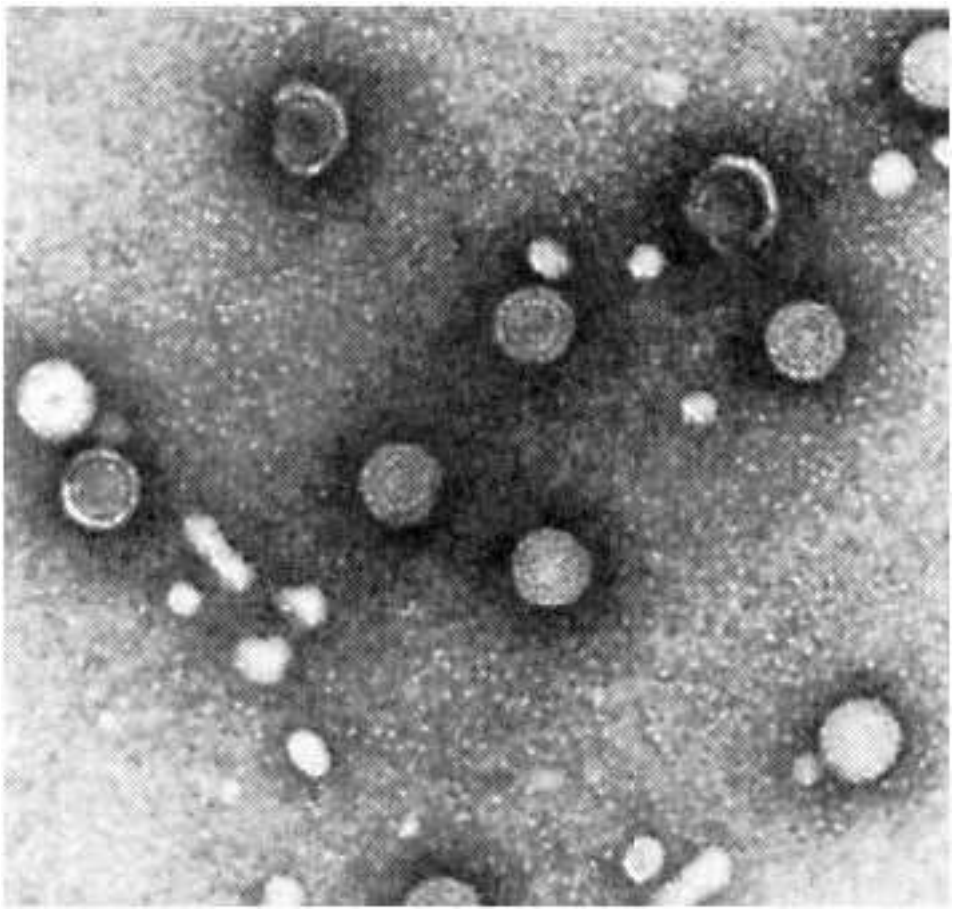


Figure 24: Electron micrograph of hepatitis B virus particles (virion cores produced by detergent treatment of virions) [12].

attached 19 nt, 5' capped oligoribonucleotide primer [66].

In orthohepadnaviruses the envelope (surface antigen) proteins of virions consist of three groups of antigenically complex proteins: S-proteins (p24, gp27), M-proteins (p33, gp36) and L-proteins (p39, gp42). All three have common carboxy termini and differ in amino termini (due to different sites of translation initiation) and in the presence and form of glycosylation. In hepatitis B virus the S-proteins appear to have the same amino acid composition and, beside this, gp27 has a single glycosylation site that is shared by the M-proteins p33 and gp36 which are composed of p24 with additional 55 amino acids and a further glycosylation site. The L-proteins contain about further 120 amino acids and their N-termini are myristylated.

The virion core is composed principally of the 22 kDa core antigen, a phosphoprotein. Enzymes associated with virions include a protein kinase and reverse transcriptase with RNA- and DNA- dependent DNA-polymerase and RNase H activities (P-gene products). Other functional components include the terminal protein covalently attached to the 5' end of the full length DNA strand. This terminal protein has been shown to be a component of the about 90 kDa P gene product.

The envelope lipids of virions are derived from the host cell membranes.

6.2 The Genomic Organization of Hepadnaviruses

The hepatitis B virus genome has four partially overlapping genes (S, C, P, X), all oriented in the same direction. The duck hepatitis B virus (genus Avihepadnavirus) consists of three genes (S, C, P). There appear to be no intervening sequences. The S-gene ORF codes for the surface antigens. In the S-gene the p24 protein is preceded by pre-S2 which, in turn, is preceded by pre-S1. Each has an in-frame ATG start codon for the initiation of protein synthesis of all surface antigens (S-, M- and L-proteins). The C-gene ORF

specifies the core protein. This is preceded by a short pre-C region that varies in size between different viruses and is larger in avihepadnaviruses than in mammalians.

The P-gene covers 80 % of the entire genome and overlaps the other three ORFs. It codes for the reverse transcriptase, with DNA polymerase and RNase H activities, and the genome-linked terminal protein. Finally, the X-gene specifies a protein with a probable transactivation function. It varies in size among the HBV serotypes.

After having entered the hepatocytes, the single-stranded regions are made full length double-stranded DNA. The terminal protein of the negative strand is removed, also the terminally redundant region and the oligoribonucleotide of the positive strand, and the DNA is converted into a covalently closed circular DNA by ligation.

After location of the double-stranded DNA into the nucleus of infected cells, transcription of viral mRNAs starts enhanced by the X-protein. Transcription yields various species of mRNAs with various lengths which code for the viral proteins. Following transcription, translation of the gene products ensues in the cytoplasm of the infected cells. The 3.4 kb pre-genome is transcribed which is greater in size than the genome, because of terminally redundant sequence parts. The pre-genome is initiated near the start of the pre-C ORF and terminates about 100 nucleotides downstream of the pre-C initiation site after making a complete copy of the genome (240 nucleotides downstream in the case of avian hepatitis B virus). The mRNAs are unspliced and are made from distinct promotors. Two regions of the HBV genome have transcription enhancer activities, another is similar to glucocorticoid responsive elements.

Integration of viral DNA into the host genome is possible, but singly integrated forms cannot serve as templates for the synthesis of the 3.4 kb pre-genome (which requires circularized or concatenated copies of integrated

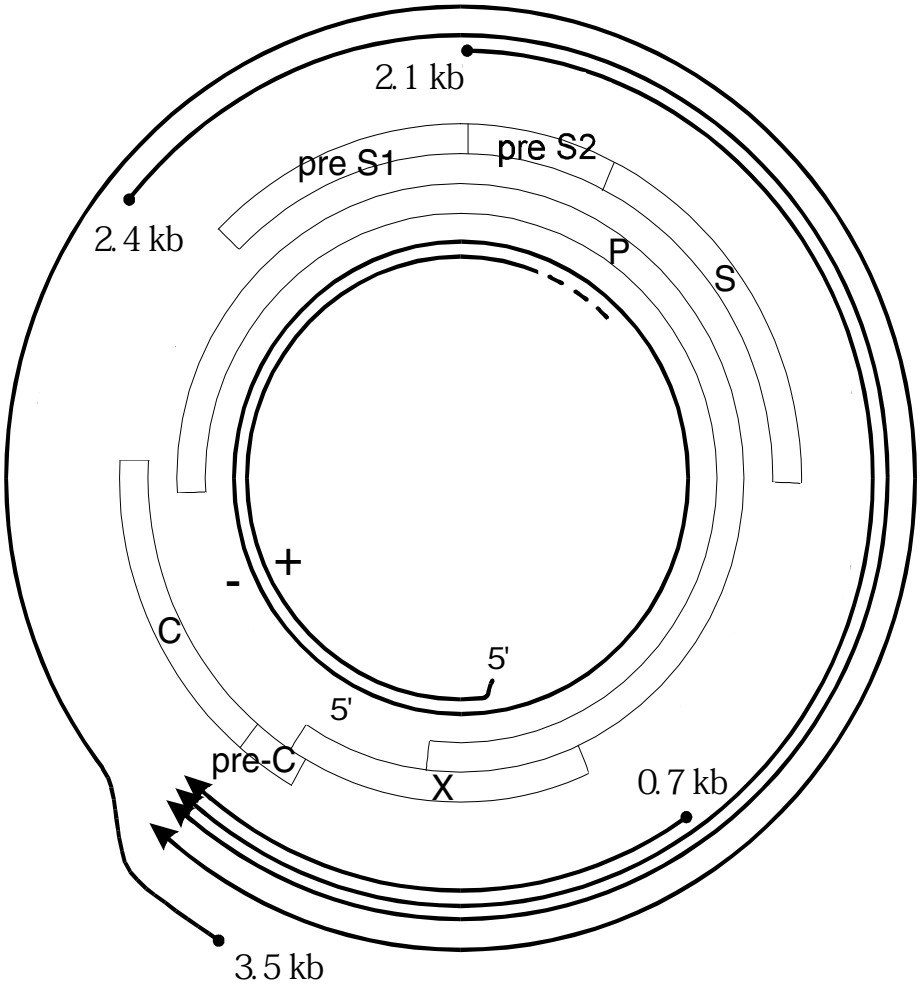


Figure 25: Diagram of the genome organization of hepatitis B virus indicating the DNA arrangement, the positions of the four open reading frames (C, P, S, X) and the mRNA transcripts.

DNA).

Current evidence indicates that following the generation of a covalently closed circular DNA and synthesis of the 3.4 kb pre-genome, this RNA associates with viral core particles where it serves as a template for synthesis of minus-strand DNA by reverse transcription using a protein primer. Then the minus-strand DNA serves as a template for plus-strand DNA synthesis. The plus-strand DNA strand is incomplete in most core particles at the time of virion assembly and release from infected cells. And finally the carboxy-terminal domain of C-protein probably is required for packaging the DNA.

6.3 The ε-Structure: a proximal Encapsidation Signal

The ε-structure is a 5'-proximal encapsidation signal which mediates the specific packaging of the transcript into viral capsids after reverse transcription of the terminally redundant RNA pregenome by interaction with the reverse transcriptase (P-protein) [65, 93]. Because of the redundancy of the RNA pre-genome a second copy of the ε-structure occurs at the 3'-end. See figures 28 and 31. ϵ-function is correlated with the formation of a hairpin containing bulges and a loop. This interaction is not only central to pregenome packing but also to capsid assembly. Furthermore, it is essential for initiation of reverse transcription. There is striking assymetry in the importance of primary sequence in the 5'- and the 3'-part of the ϵ-structure. The motif CU is important in the proximal bulge position. It has been proposed that these nucleotides are in close contact with P protein. Deletion of the bulge prevents incapsidation [93].

6.4 The HPRE regulatory element

Posttranscriptional regulatory processes may be an important means of enhancing transfection efficiency. Although the exact mechanism is unknown,

posttranscriptional regulatory systems may involve facilitating RNA processing and/or RNA export to the cytoplasm. The post-transcriptional regulatory element (HPRE) facilitates the cytoplasmic localization of intronless transcripts and is composed of at least two independent sub-elements (α and $\beta 1$) which are necessary for full HPRE function and conserved throughout the mammalian Hepadnaviruses. Two regions of these sub-elements (SLα and SL$\beta 1$) form stem-loop structures [100], Figure 27.

6.5 Results for Hepatitis B Virus

The Hogeweg-mountain representation is often the best suited representation of RNA for the techniques of finding conserved secondary structure elements. In a mountain plot every left-handed ascent and its corresponding right-handed descent indicate base pairs, plateaus are loops.

We have investigated 14 pregenomic RNA sequences and all known subgenomic RNAs of hepatitis B virus. The processed sequences are members of 6 different subgroups which contain some different sequence parts.

Table 1: With 14 pregenomic RNA sequences of hepatitis B virus the alignments using various algorithms were done:

AF046996	HBV131567	HBVAYWE	AB014366
HHVBFFOU	HBV131571	HBV18858	HBD50521
HBVADW4A	HBV131574	HBV18857	HBVAYWMCG
HBV131573	HVHEPB		

In the case of hepatitis B virus, the redundant terminal part of the pregenomes was attached to the sequence end. Using `code2aln` and also `clustalw` led to alignments (of hepatitis B virus and Levivirus) which were suitable for further processing by `alidot` and the technique of detecting conserved

secondary structure elements in complete virus genomes.

In principle, the two different alignment algorithms yield results that look quite different, but we can see various structures which are confirmed by both processes in the same way (see the next pages).

In the case of `clustalw` we find the four most prominent conserved elements: the ε-element, a proximal encapsidation signal which is necessary for the specific packaging of the transcript into viral capsids, and its copy at the 5' and 3' end of the pregenome. And the two stem loop structures α and $\beta 1$ in the HPRE region (the post-transcriptional regulatory element) that facilitates the cytoplasmic localization of intronless transcripts. Using `code2aln`, the $\beta 1$-element could not be detected, but we see some novel structures that are not detected the same way using `clustalw`.

In the C-transcript we found one stem-loop structure, Figure 29, that does not appear to be conserved in the pregenomic RNA because interactions with a region outside the C-gene are thermodynamically favorable. The fact that we find a conserved structure in C and distinctly different folding in other context makes this feature a candidate for a regulatory function.

In principle, secondary structures of ssRNA viruses or (as in the case of hepatitis B virus) pregenomic RNA intermediates, or even mRNAs, are known to play an important role in the regulation of the viral life cycle.

When comparing the alignments themselves, we can see one tendency regarding `code2aln`: `code2aln` tends *not* to disrupt codons within coding regions. The reason is that `code2aln` respects the fact that amino acids are the biologically relevant part of the system in coding regions, and amino acids, respectively the codons, should not be disrupted, and gaps should not be inserted into codons, if possible, given the used scoring function.

6.5.1 Using clustalw

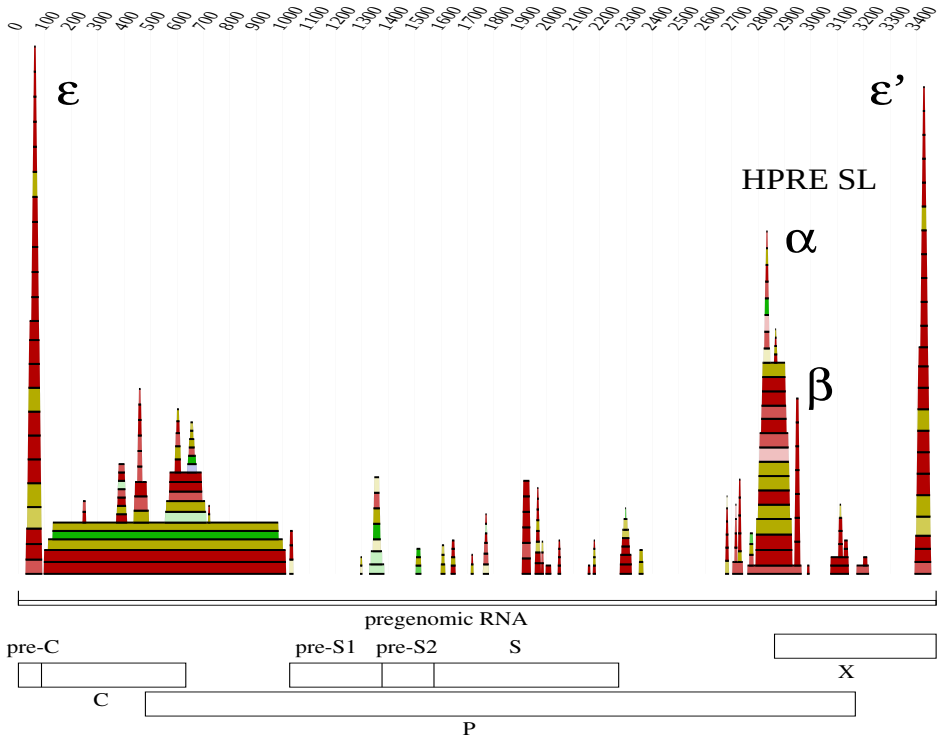


Figure 26: Using `clustalw`: The four most prominent conserved elements are the ε-element, a proximal encapsidation signal which is necessary for the specific packaging of the transcript into viral capsids, and its copy at the 5' and 3' end of the pregenome. And the two stem loop structures α and $\beta 1$ in the HPRE region (the post-transcriptional regulatory element) that facilitates the cytoplasmic localization of intronless transcripts. Colors indicate compensatory mutations: red = 1, ocre = 2, green = 3, turquoise = 4, blue = 5 and violet = 6 types of base pairs.

Various colours are used to indicate compensatory mutations. Pale colours tell us that we have some inconsistent mutations, which means, variable sequence positions where not all sequences do base pairing. The colour codes of the Hogeweg mountain plots are as follows:

`red`	all sequences have the same two nucleotides
`ocre`	two types of base pairs occur
`green`	three types of base pairs occur
`turquoise`	four types of base pairs occur
`blue`	five types of base pairs occur
`violet`	all six possible types of base pairs occur

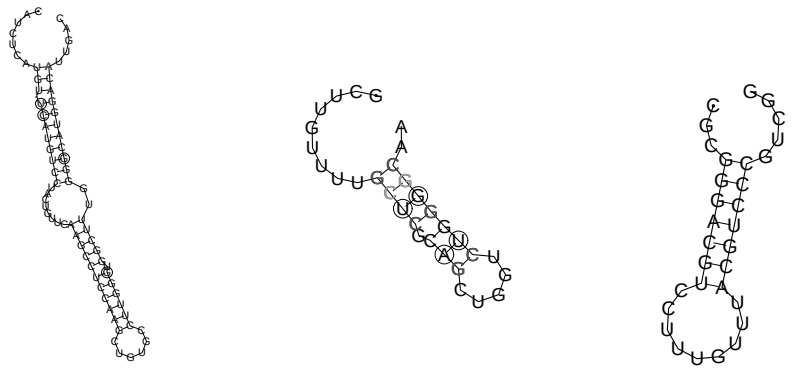


Figure 27: Predicted secondary structure of ε, α, and $\beta 1$ element. Circles indicate compensatory mutations. The post-transcriptional regulatory element (HPRE) facilitates the cytoplasmic localization of intronless transcripts and is composed of at least two independent sub-elements (α and $\beta 1$) which are necessary for full HPRE function and conserved throughout the mammalian Hepadnaviruses.

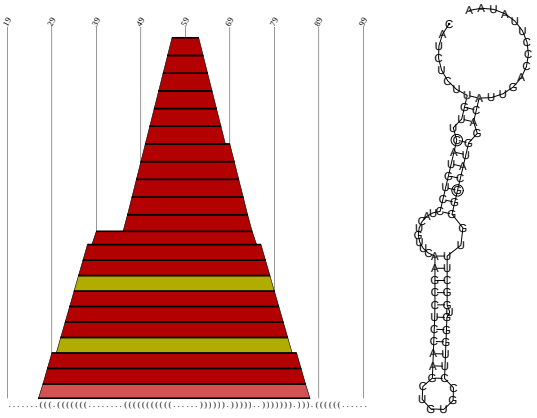


Figure 28: The ε-element in Human Hepatitis B Virus RNA Pregenomes. This structure is conserved among all Mammalian Hepatitis B Viruses. Because of the redundancy of the RNA pre-genome a second copy of the ε-structure occurs at the 3'-end. This is a highly conserved sequence part, only two compensatory mutations confirm the conserved structure.

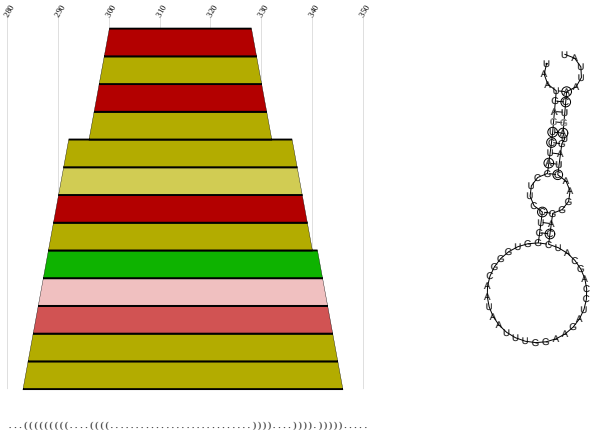


Figure 29: A secondary structure element with unknown function that appears to be conserved in the C-mRNA but does not appear as conserved structure in complete pregenomic RNA. In this case interactions with a region outside the C-gene are thermodynamically favorable. The fact that we find a conserved structure in C and distinctly different folding in other context makes this feature a candidate for a regulatory function.

6.5.2 Using code2aln

The Hogeweg mountain plot after usage of `code2aln` looks quite different from the one after usage of `clustalw`, although most of the important elements are also detected.

The usage of genetic information about coding and non-coding sequence parts as part of the scoring function like implemented in `code2aln`, or the combined amino acid and nucleic acid based alignment algorithm as implemented in `ralign` formerly, is especially helpful when used with rather closely related sequences. In these cases the reduction of gaps (especially with `ralign`) and the incorporation of amino acid scores (in `code2aln`) bears quite good alignment results.

Therefore, as an example for an application, human hepatitis B virus genomes were used as well as the genome sequences of various species of Levivirus genus (in the following). It is possible to detect various conserved secondary structure elements in hepatitis B; the most important of them are also detected after alignment with `clustalw`, but there are also some differences: the β1-element could not be detected here, and we see some novel structures that are not detected the same way using `clustalw`.

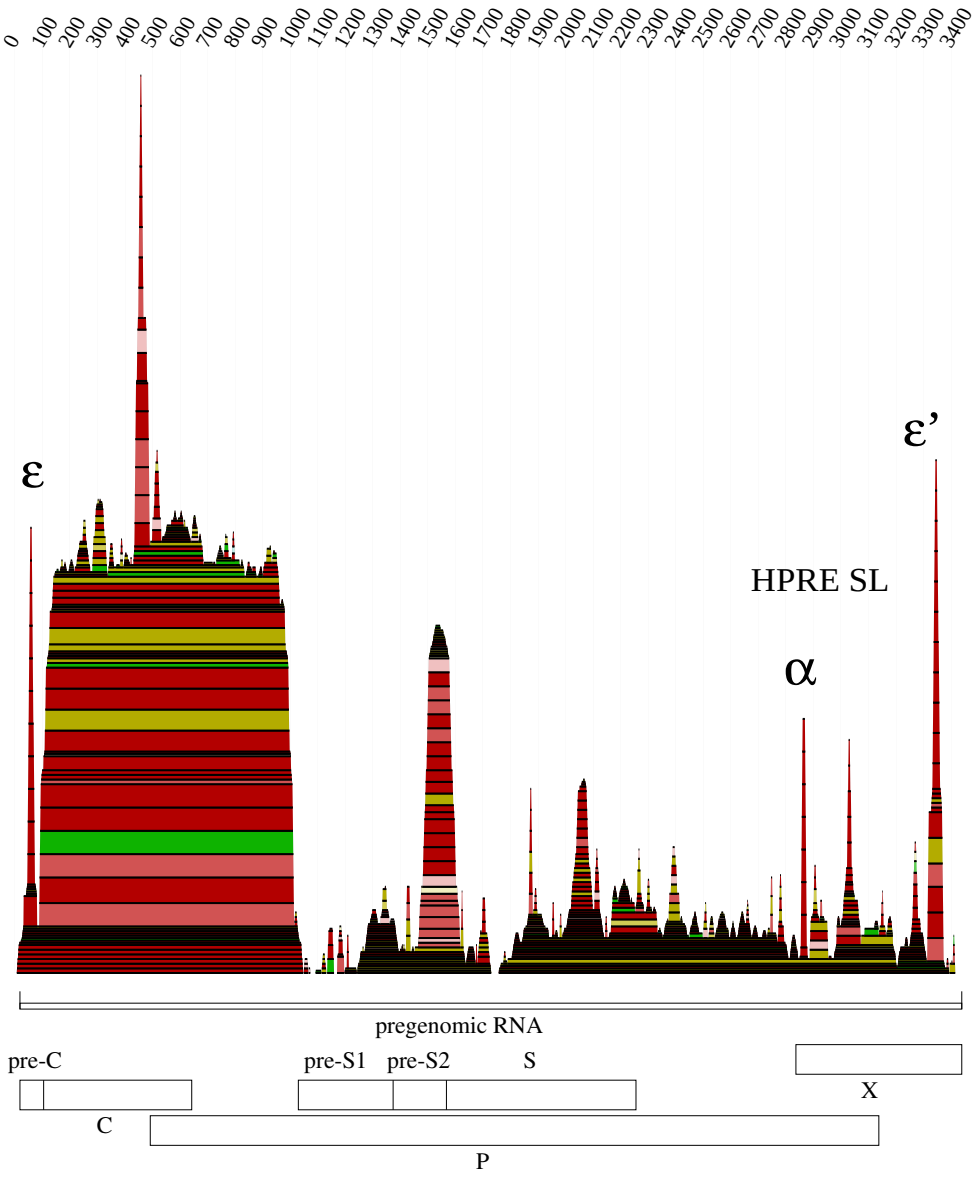


Figure 30: Using `code2aln`: Again three of the four most prominent conserved elements: the ε-element and its copy at the 5' and 3' end of the pregenome and the stem loop structure α in the HPRE region. The β1-element could not be detected here. Instead of it we see an other, but noticeable, element beside the α element which is maybe also part of the HPRE region.

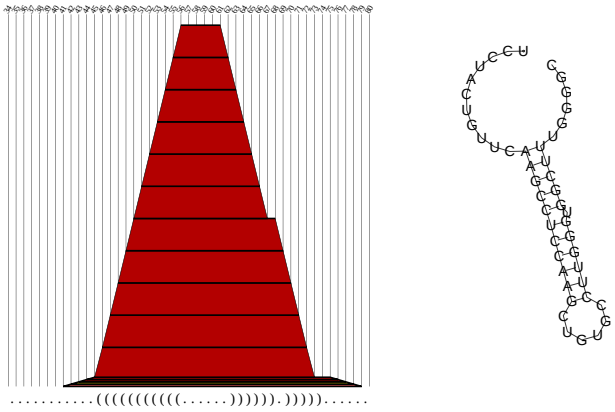


Figure 31: The ε-element in human hepatitis B virus RNA pregenomes as detected after `code2aln` alignment. A 5'-proximal encapsidation signal which mediates the specific packaging of the transcript into viral capsids. This structure is conserved among all Mammalian Hepatitis B Viruses.

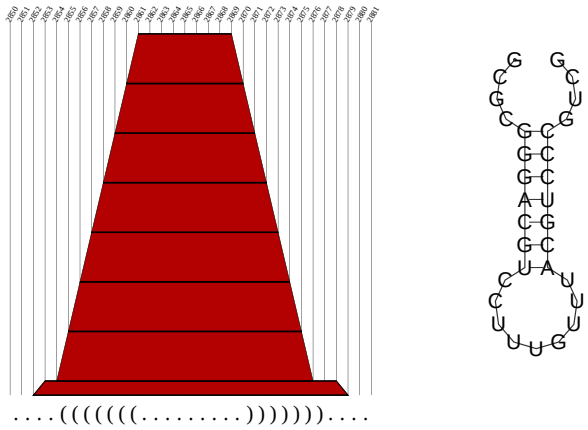


Figure 32: The α-element in human hepatitis B virus RNA pregenomes as detected after `code2aln` alignment. This structural element is conserved as part of the HPRE region. HPRE facilitates the cytoplasmic localization of intronless transcripts and is composed of at least two independent sub-elements (α and $\beta 1$) which are necessary for full HPRE function and conserved throughout the mammalian Hepadnaviruses. Using `code2aln` as the input alignment we could not detect the $\beta 1$ structure.

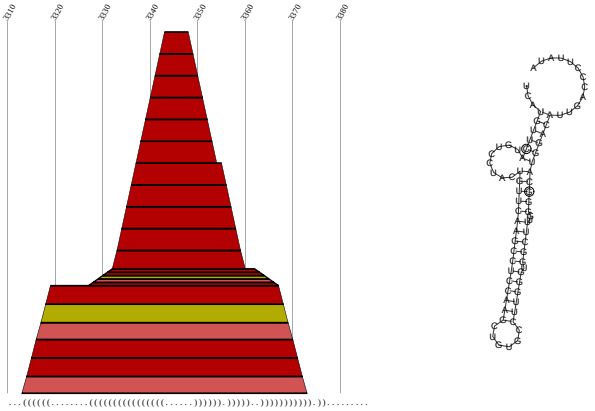


Figure 33: The ε-element at the end of human hepatitis B virus RNA pregenomes as detected after `code2aln` alignment. This structure is conserved among all mammalian hepatitis B viruses, but `code2aln` alignment led to a longer structural element at the redundant end than at the beginning.

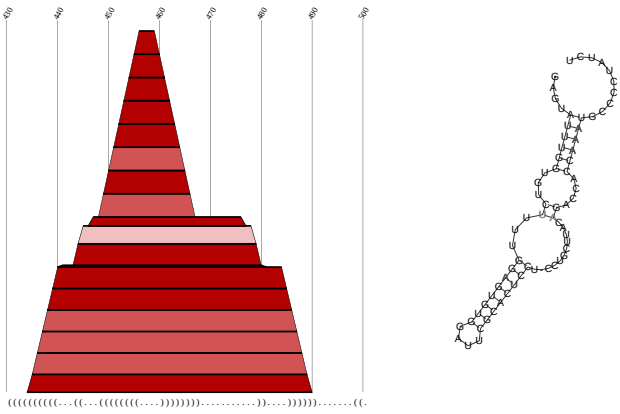


Figure 34: Positions 430 - 500: A conserved structural element as detected after `code2aln` alignment. This structural element is obviously unknown, there are no informations what the function might be; but it seems to be conserved, at least as a primary structure, on sequence level.

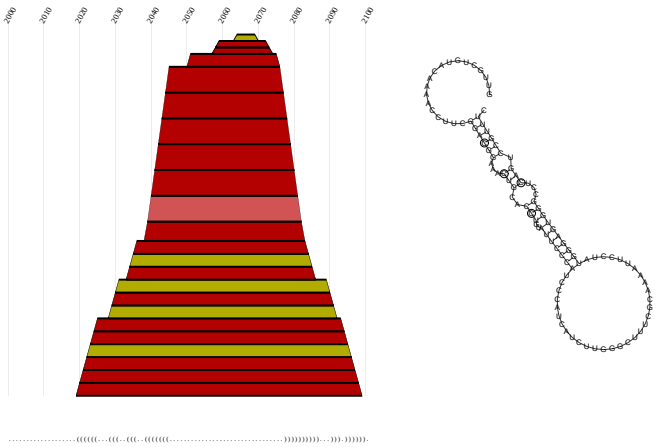


Figure 35: Positions 2000 - 2100: A long hairpin with two interior loops and one short bulge as a conserved structural element as detected after `code2aln` alignment. The structure is highly conserved and confirmed by various compensatory mutations. There are no informations what the function of this element might be.

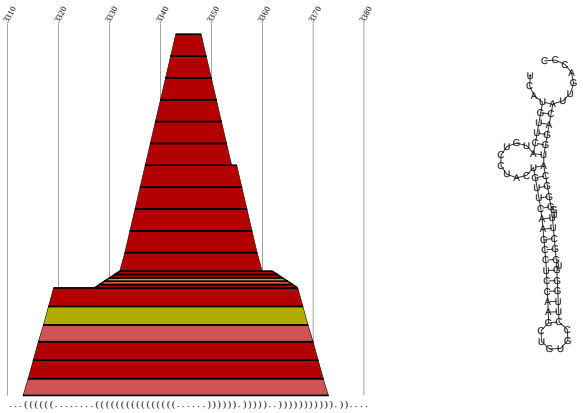


Figure 36: Positions 3310 - 3380: A long hairpin with two short bulges and one long bulge. Two compensatory mutations. We don't know what the function of this element might be.

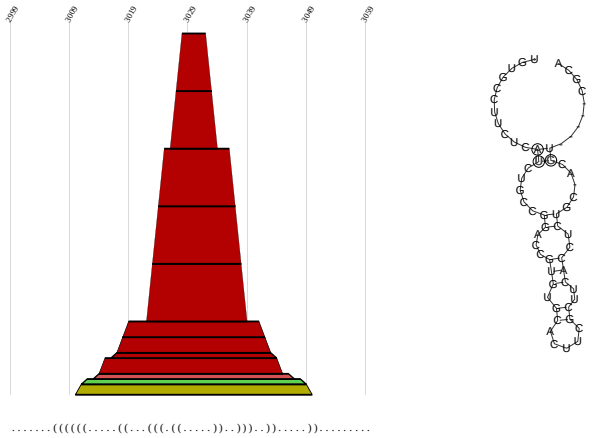


Figure 37: Positions 3000 - 3060: Also this conserved structural element as detected after `code2aln` alignment is obviously unknown regarding its function. We see some interior loops between highly conserved segments. Highly conserved at least on the level of the primary structure, the sequence. We see no sequence variation; this is a fact that is not supporting the prediction of a conserved structure. Maybe, it is also some part of the HPRE region, but it is obviously not the β1-element.

Table 2: Summary of the positions of predicted secondary structure elements in the RNA pregenome of Human Hepatitis B Virus after two different types of alignment (`clustalw` and `code2aln`).

`clustalw`	Figures	`code2aln`	Figures	Function
1-100	26, 27, 28	40-75	30, 31	ε-element
2800-2850	26, 27	2850-2870	30, 32	α-element (HPRE)
2930-2970	26, 27	?	none	β1-element (HPRE)
3360-3440	26, 27	3300-3380	30, 33	repeat of ε-element
C-mRNA	29	?	none	?
?	none	430-500	34	?
?	none	2000-2100	35	?
?	none	3310-3380	36	?
?	none	3000-3060	37	?

7 Leviviridae

7.1 The Morphology of the Levivirus Genus

The members of the genus Levivirus infect eubacteria. The enterobacteria phages MS2, KU1, GA, and fr are species of the genus Levivirus which belongs to the family of Leviviridae.The virions of these ssRNA Phages are neither enveloped nor tailed. The nucleocapsids are isometric, 24-26 nm in diameter. Their type of symmetry is icosahedral (this means twenty edges). Every virus particle consists of 32 capsomers per nucleocapsid. Virions contain 31 % nucleic acid, that is one molecule of linear positive-sense single stranded RNA. The molecular mass of one virus capsule of members of the family Leviviridae is about $3.6 - 4.2 \times 10^6$ (depending on the genus). The virions are sensitive to various detergents and they contain no lipids [2, 27].

7.2 The Genomic Organization of Levivirus

All members of the family Leviviridae are ssRNA positive-strand viruses.The replication cycle includes no DNA stage. The total genome length is 3466 up to 4276 nucleotides depending on type of strain. Base ratio between the purine bases is 50% guanine and 50% adenine; Also the pyrimidine bases share 50% cytosine and 50% uracil.

Most Levivirus species have four (partly) overlapping genes, some exceptions exist which contain only three open reading frames. Currently, two structural virion proteins have been found and identified. Every capsid contains one copy of the so called A protein. A typical protein size of this type is about 35000-44000 Da. It is required for the maturation of the virion and for the pilus attachment during infection of the host [12].

The protein size of the 2nd largest protein found so far is about 14000 Da. It functions as a coat protein and builds up the capsid. The capsid contains

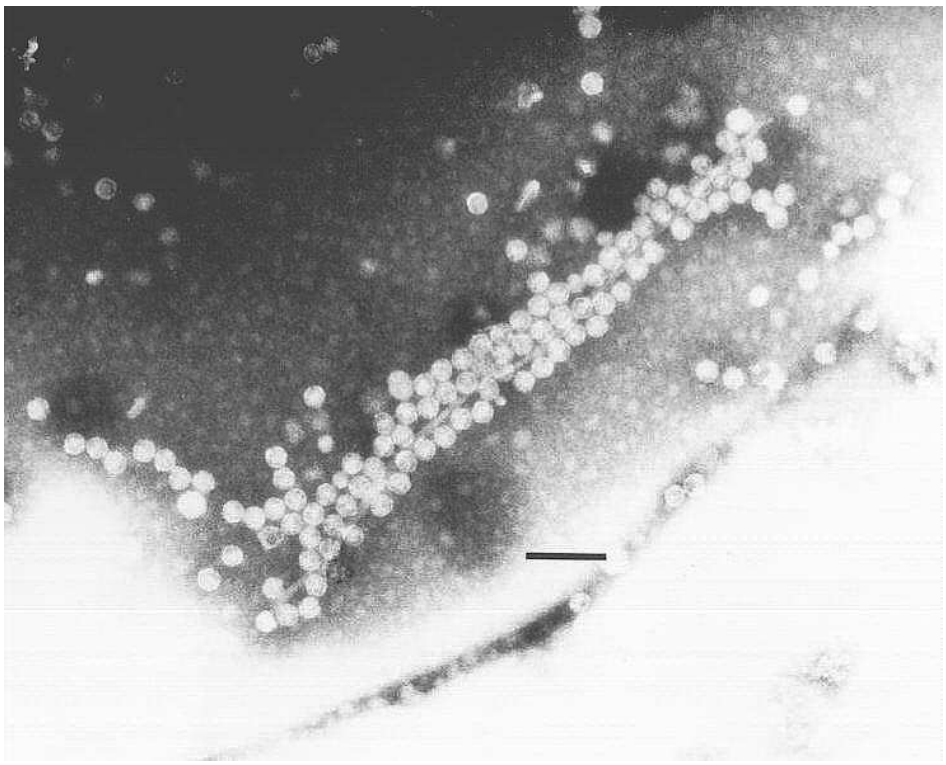


Figure 38: Electron micrograph of Levivirus virion cores. Levivirus contains no spheres, no filaments and no tails (like many other bacteriophages), from [12].

180 copies of the coat protein, arranged in 60 identical triangular units.

The infection is mediated and performed mainly by the genomic nucleic acid sequence. All newly produced virions are found in the cytoplasm. The genome replicates in the cytoplasm. The host of the members of the Levivirus genus belongs to the domain Eubacteria (Procarya) [111, 112].

7.3 Results for Levivirus

We have investigated 8 sequences of the Levivirus genus. Currently, there are 5 types of strains represented in the `GenBank` data base as complete genomes: The Enterobacteria phages `MS2`, `KU1`, `GA`, `fr`, and `AP205`.

The last one was eliminated from the data set analyzed here because of large changes in genomic structure. Changes, which cause a massive divergency from all other types and lead to the impossibility of making useable alignments for further processing by `alidot`.

The genus Allolevivirus (especially the well analyzed Qbeta) is also object of intensive research, but the sequence of Qbeta which was also in our data set first, is highly divergent from the Levivirus sequences. Neither using `clustalw` nor with `code2aln` it is possible to obtain alignments that are suited input for `alidot`. The alignments are completely disrupted and it is clearly visible that the incompatibility between Levivirus and Allolevivirus is the reason. Thus, we restrict ourselves to Levivirus in this work (see Figure 39).

At the 5'-terminal end of the Levivirus sequences we detect a short GC-rich hairpin (tetraloop) which follows to an unpaired GGG element, see Figures 40 and 51. This is probably the analogon to the recognition signal site for the RNA replicase in Alloleviviruses. This stem-loop-structure is well known and defined in Qbeta. The Qbeta replicase amplifies RNA templates autocatalytically with high efficiency. This recognition element, consisting of a hairpin and a very short unpaired region, at the 5'-terminus, is essential for function [7, 8].

The results regarding Levivirus are quite complex for analysis: the sequences are much more divergent than in the case of hepatitis B virus. Nevertheless, some conserved structure elements which are also well supported by compensatory mutations, are visible.

The two different alignment processes produce significantly different re-

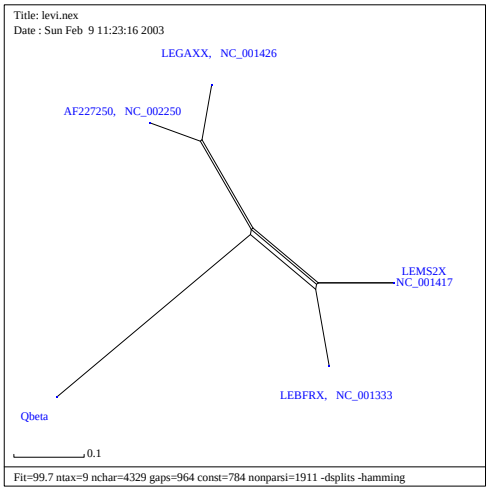


Figure 39: The phylogenetic relationships among the Levivirus sequences analyzed here and the Allolevivirus Qbeta. The evolutionary distance between the four groups of Levivirus and Allolevivirus, which destructs the alignments, is visible. The tree was produced using `SplitsTree3.1`.

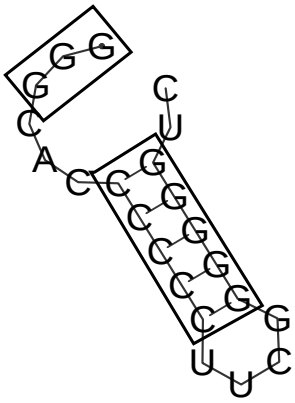


Figure 40: The 5'-terminal RNA replicase recognition site in the Allolevivirus Qbeta, which is highly functional for recognition of the template and amplification of it. Important characteristics are marked (see also Figure 51).

Table 3: The GenBank data files with the 8 input sequences of complete Levivirus RNA genomes.

Enterobacteria phage MS2	NC_001417, LEMS2X
Enterobacteria phage KU1	NC_002250, AF227250
Enterobacteria phage GA	NC_001426, LEGAXX
Enterobacterio phage fr	NC_001333, LEBFRX

sults. Especially interesting is the fact that code2aln and alidot detect an obvious type of conserved long range interactions, clearly visible at the basis of the mountain plot (see figure 50).

Using clustalw these long range effects are not detected, but they seem to be important, because they are apparently supported by compensatory mutations.

When comparing the alignments themselves, we can again see that code2-aln tends *not* to disrupt codons within coding regions. We count 36 gaps (out of 182) with a length that can be divided by 3 in the clustalw alignment, but 40 gaps (out of 166) of this type in the code2aln alignment. This is a indication that code2aln respects coding regions, and that amino acids, respectively the codons, should not be disrupted, and gaps should not be inserted into codons, if possible, given the used scoring function.

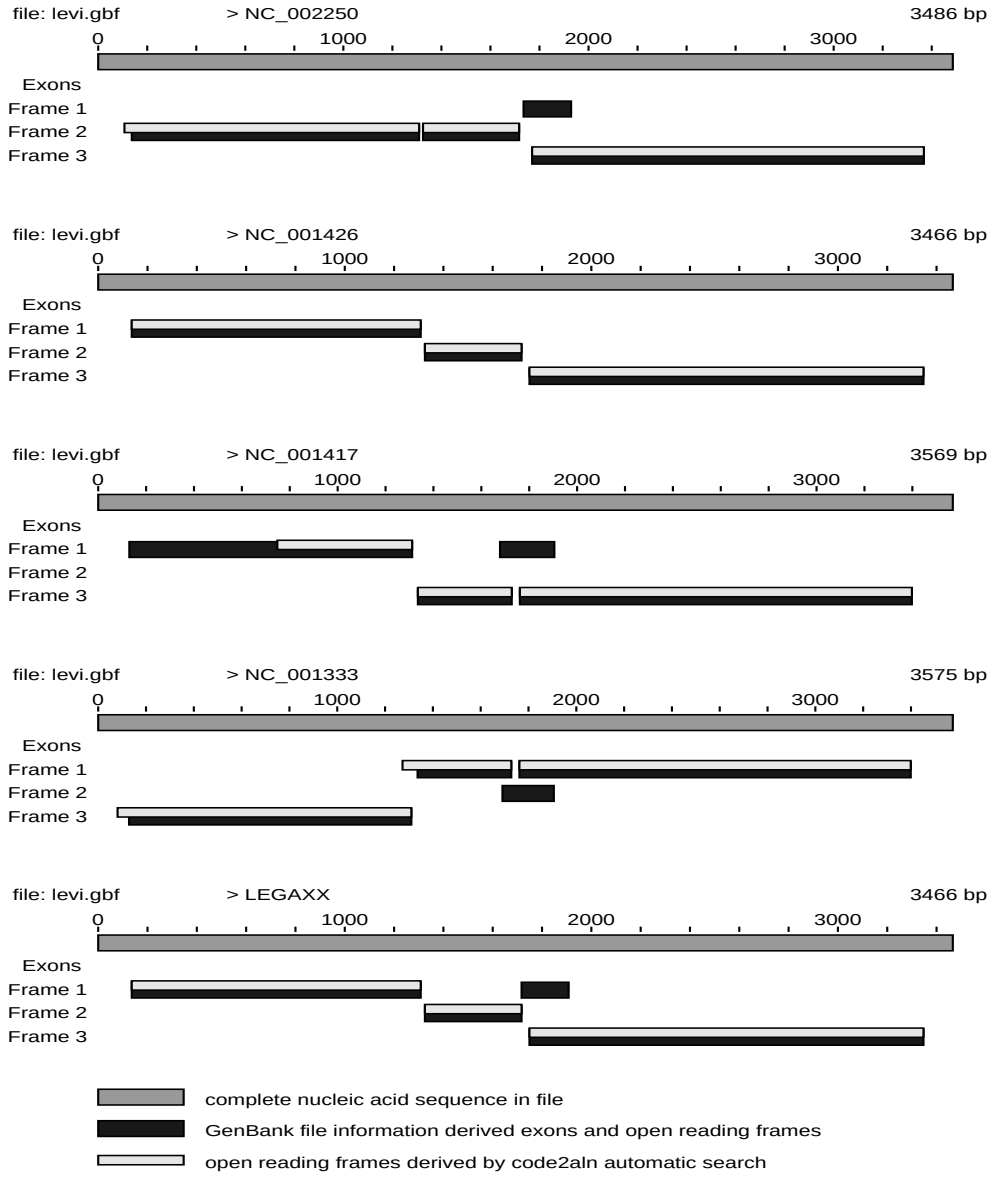


Figure 41: This `PostScript` output of `code2aln` shows the graphical representation of the Levivirus open reading frames derived from the `GenBank` file and detected by the `code2aln` automatic search for coding regions. All sequences represented here were used as an input into the process of detecting conserved secondary structure elements by `alidot`.

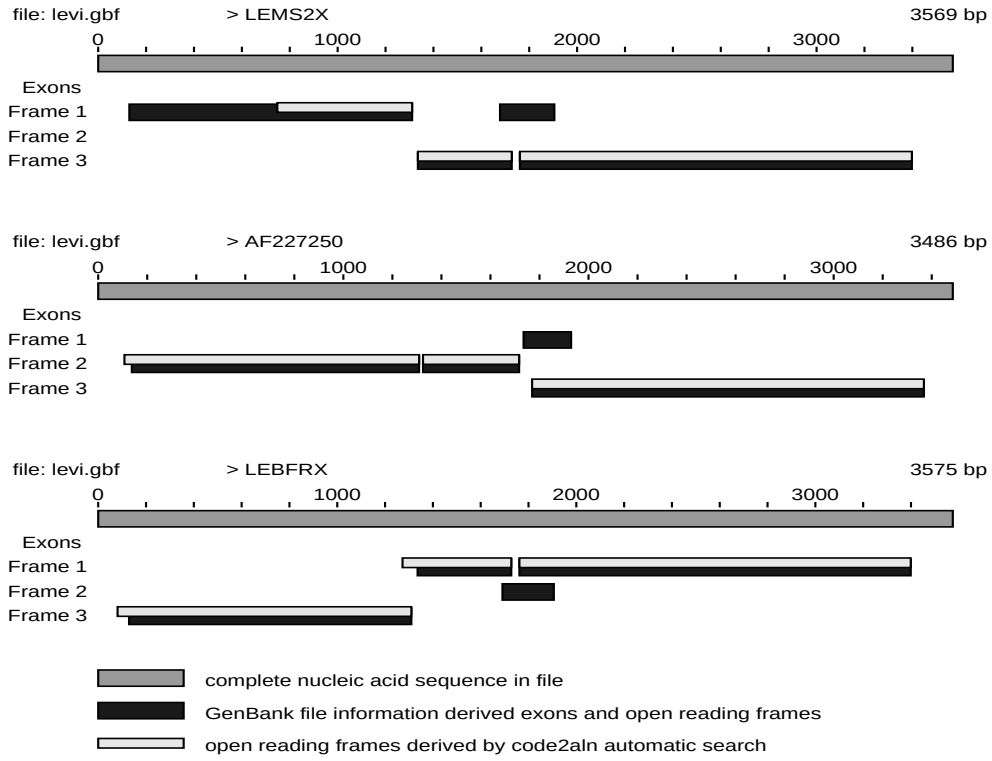


Figure 42: This `PostScript` output of `code2aln` shows the graphical representation of the Levivirus open reading frames derived from the `GenBank` file and detected by the `code2aln` automatic search for coding regions. All sequences represented here were used as an input into the process of detecting conserved secondary structure elements by `alidot`.

7.3.1 Using clustalw

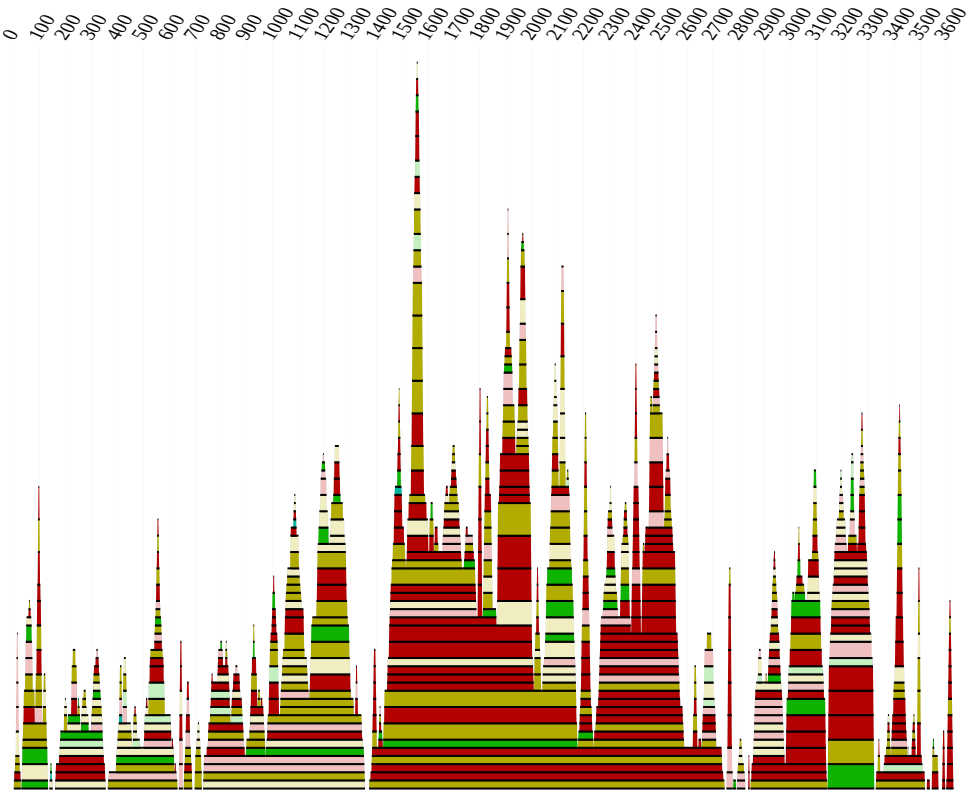


Figure 43: Using `clustalw`: This Hogeweg mountain plot shows an overview about the complete RNA genome of Levivirus genus. It looks quite different from the version that was produced using `code2aln` as an input alignment, see below. But one can see at a first glance that the sequences are very divergent, although it seems that we have quite a lot of compensatory mutations.

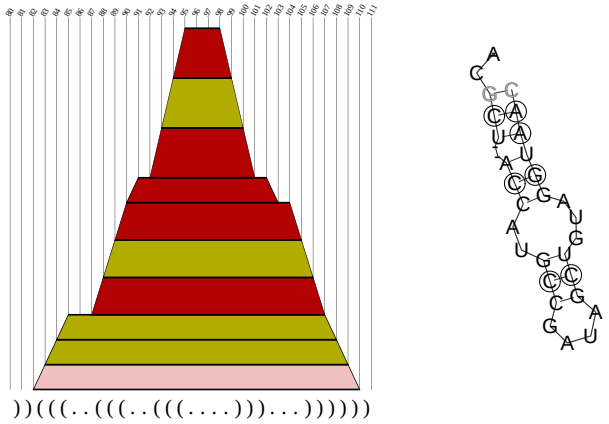


Figure 44: Position 80 - 110: A highly conserved structure, a hairpin with a tetraloop and an interior loop. Lots of compensatory mutations strongly confirm this element. At this point, we do not know what the function of this element is.

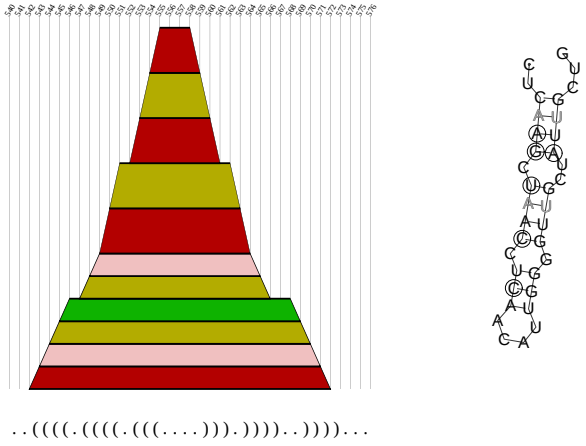
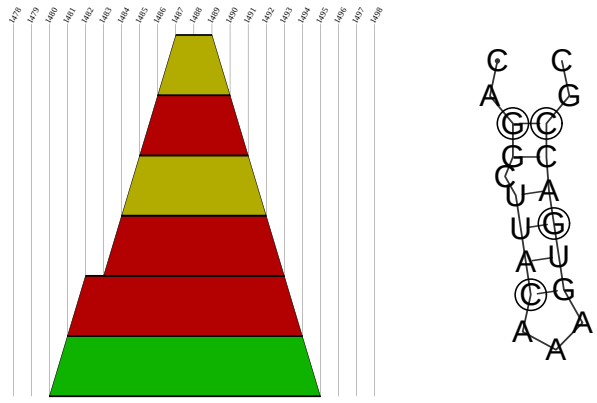


Figure 45: Position 540 - 575: A conserved tetraloop tip followed by a variable position and again a very conserved secondary structure part. The green colour indicates that we have three types of base pairs at this position. We have no information what the function might be.



. . ((. ((((. . .)))))) . .

Figure 46: Position 1480 - 1495: A short but highly conserved hairpin structure. Two ocre and one green trapezoids indicate compensatory mutations that strongly support the structure. No information about the function.

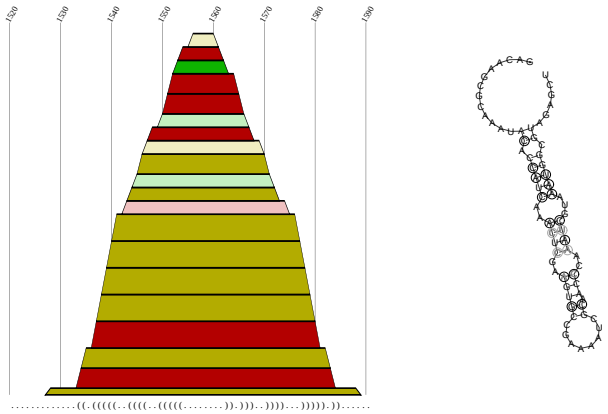


Figure 47: Position 1520 - 1590: Lots of compensatory mutations with two and (once) three types of base pairs. A highly conserved structure with unknown function.

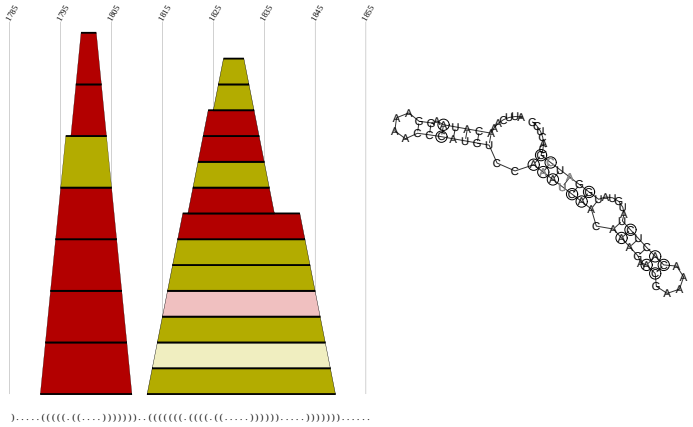


Figure 48: Position 1785 - 1855: Two hairpins, one features a tetraloop and a short bulge, the other has an interior loop and also a short bulge near the terminal loop. Both are conserved and thus obviously important. Various compensatory mutations.

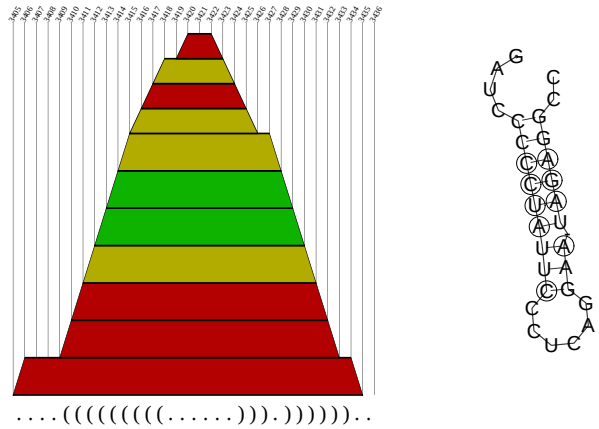


Figure 49: Position 3405 - 3435: A very highly conserved structure, strongly supported by lots of compensatory mutations. One short bulge, two green trapezoids in the mountain plot show three types of base pairs.

7.3.2 Using code2aln

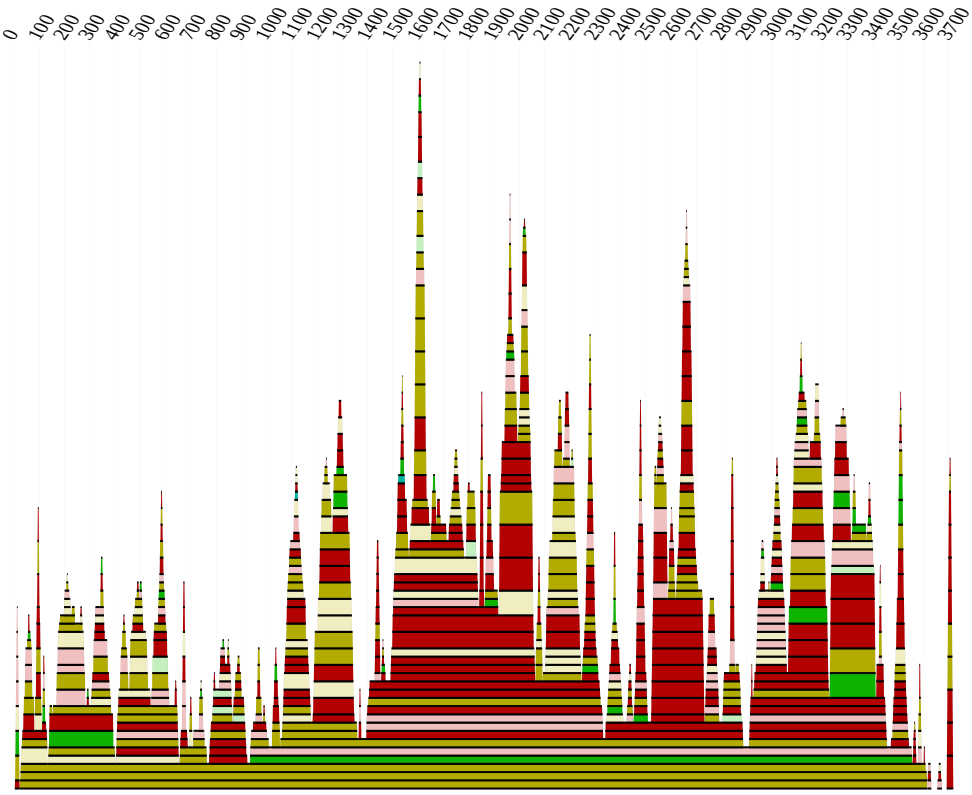


Figure 50: Using `code2aln`: This Hogeweg mountain plot shows an overview about the complete RNA genome of Levivirus genus after alignment by `code2aln`. It is very different from the version above that was produced using `clustalw`. Very interesting is this obvious type of conserved long range interactions, clearly visible at the basis of the plot. Using `clustalw` these long range effects are not detected, but they seem to be important because they are apparently supported by compensatory mutations.

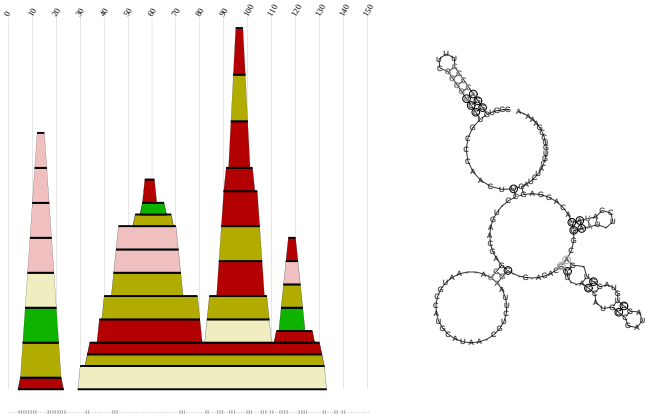


Figure 51: Position 1 - 150: A multiloop structure containing three tetraloops and some interior loops. We see various compensatory mutations of second and third order, but also some variable positions that show higher divergence. The 5'-terminal hairpin is probably the analogon to the recognition signal site for the RNA replicase in Alloleviviruses which is well analyzed in Qbeta. In Qbeta the replicase amplifies RNA templates autocatalytically with high efficiency. This recognition element in Levivirus is supposed to be essential for a similar function

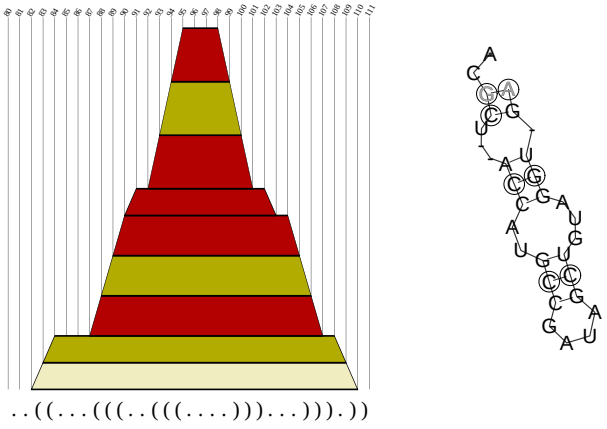


Figure 52: Position 80 - 110: This structure contains alternating conserved stems and interior loops and a terminal tetraloop. Three positions with compensatory mutations.

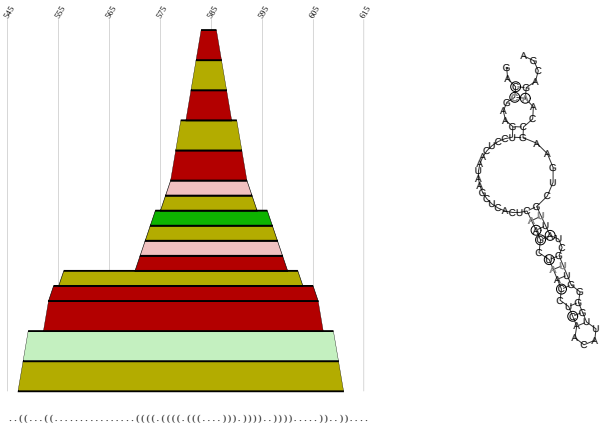


Figure 53: Position 545 - 615: Also here we see a tetraloop structure and various interior loops of different size. Some compensatory mutations; no information what the function of the element might be.

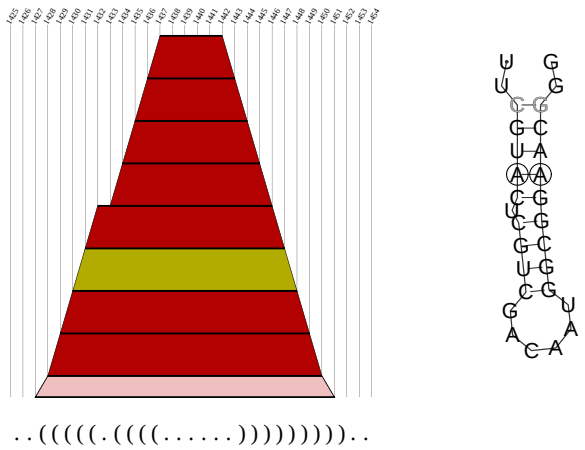


Figure 54: Positions 1427 - 1451: A highly conserved sequence, one compensatory mutation supports the structure, we see a short one-letter bulge.

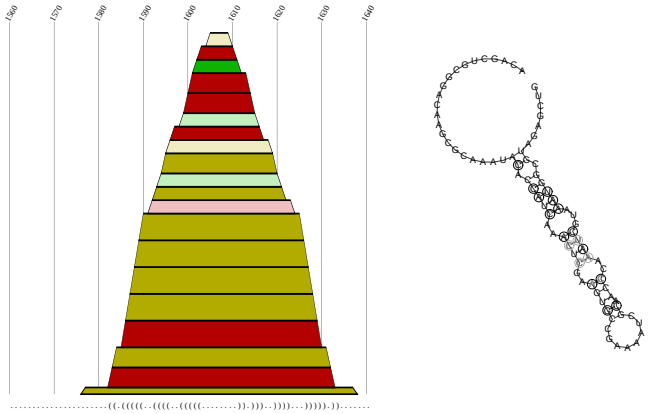


Figure 55: Positions 1560 - 1640: A hairpin structure with three interior loops and one bulge near the terminal loop. Lots of compensatory mutations strongly support the structure on one hand, but on the other hand we a see some variable positions (pale colours in the mountain plot).

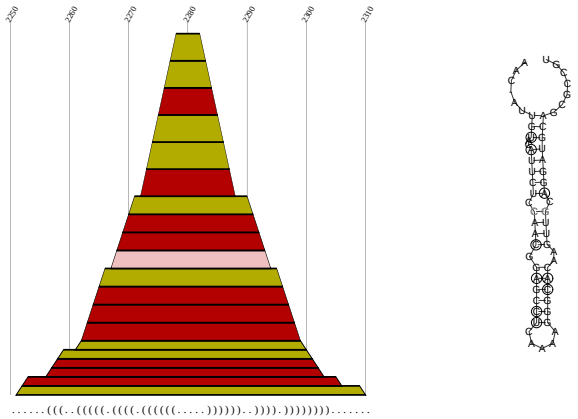


Figure 56: Positions 2250 - 2310: A long conserved hairpin structure containing two internal loops and one short bulge. We cannot tell what the function of this element might be.

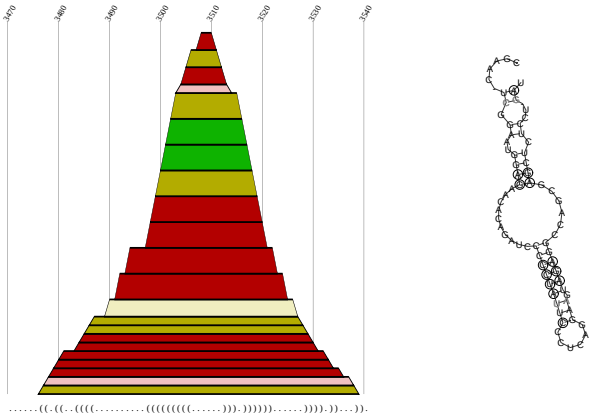


Figure 57: Positions 3470 - 3540: At the top of the mountain plot we see a highly supported stem with green trapezoids (three types of base pairs), then a large interior loop, then again two conserved pairs, confirmed by compensatory mutations.

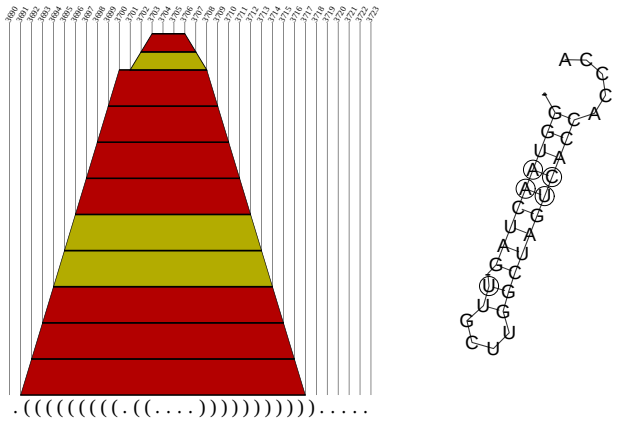


Figure 58: Positions 3690 - 3717: A longer hairpin with tetraloop and one short bulge near the tetraloop; three compensatory mutated base pairs support the structure. Also in this case we have no information what the function of the element might be.

Table 4: Summary of the positions of predicted secondary structure elements in the genomic RNA of Leviviridae after two different types of alignment using `clustalw` and `code2aln`.

`clustalw`	Figures	`code2aln`	Figures	Function
?	none	1-150	51	replicase recognition
80-110	44	80-110	52	?
540-575	45	545-615	53	?
1480-1495	46	1427-1451	54	?
1520-1590	47	1560-1640	55	?
1785-1855	48	?	none	?
?	none	2250-2310	56	?
3405-3435	49	3470-3540	57	?
?	none	3690-3717	58	?

8 Conclusions and Outlook

It was the aim of this work to construct and implement an alignment algorithm that produces multiple nucleic acid alignments using information on coding and non-coding regions as part of the scoring function. Information as contained in the amino acid sequences translated from the coding regions of a partial or complete (virus) genome after identification of these open reading frames or exons. Overlapping open reading frames, as they occur in viruses, are respected and information about them is used in order to improve the resulting alignment. This algorithm is implemented in the program `code2aln`.

`Code2aln` is written in `ANSI C` and hence easily portable to different operating systems. It was developed on PCs running `Linux` and works well with different `Unix` dialects.

The program was utilized for the generation of part of the input for the procedure of detecting conserved RNA secondary structure elements in hepatitis B virus and Leviviridae (pre)genomes.

In this thesis it is demonstrated that the alignments are indeed improved by integration of genetic information into the scoring function and that the `code2aln` approach is indeed expedient, in that the alignments, used as input into the process of `alidot`, produce various predictions of secondary structure elements in both investigated virus types that are not detected when using conventional alignment algorithms like that implemented in `clustalw`.

The program `alidot` is used to scan a group of RNA sequences for conserved secondary structure elements and to predict their consensus structure. A good multiple alignment and structure predictions for each of the sequences under consideration are required as input for this procedure. The performance of the `alidot` algorithm depends crucially on the quality of the multiple nucleic acid sequence alignment. Although the results are quite

robust to minor alignment inaccuracies, those always become a substantial problem when dealing with more diverse data sets. Hence, the advantages of `code2aln` lie especially in the fact that `code2aln` makes implicit use of all information about the genetical relevance of the input data reflecting the coding and non-coding of sequence parts which are biologically meaningful as protein sequences. Nucleic acid sequence positions that are part of one or even more codons are weighted significantly higher than positions that are not in open reading frames (or exons).

In principle, when calculating `alidot`-predictions, the occurrance of consistent and, in particular, compensatory mutations strongly supports a predicted base pair.

When comparing the alignments themselves, we can see one strong tendency regarding `code2aln`: `code2aln` significantly tends *not* to disrupt codons within coding regions.

In the case of hepatitis B virus, we count 54 gaps with a length that can be divided by 3 in the `clustalw` alignment, but 173 gaps of this type in the `code2aln` alignment. This is a factor of more than 3.2.

In the case of Leviviridae, we can again see the same effect: we count 36 gaps with a length that can be divided by 3 in the `clustalw` alignment, but 276 gaps of this type in the `code2aln` alignment. This is a factor of almost 7.7 and a very strong indication that `code2aln` works very well.

The reason is that `code2aln` respects the fact that amino acids are the biologically relevant part of the system in coding regions, and amino acids, respectively the codons, should not be disrupted, and gaps should not be inserted into codons, if possible, given the used scoring function.

When using the `code2aln` approach we can assure that insertions and deletions within coding regions highly tend to correspond to insertions and deletions at the protein level.

Besides, it is possible to define one or more than one codon tables for

each sequence or groups of sequences. The codon tables are important, because they ensure that fine adjustment of detection and translation of coding regions during the process of aligning is possible for various organism groups or their fitting mitochondrial sequences.

We use a theoretical approach what is supposed to be useful for the special alignment problem of coding sequences: we add protein sequence information when we know that the translated protein sequences are of dominant importance in our alignment of nucleic acids.

We find a number of highly significant secondary structure elements, not being described in the literature so far, and some well known elements such as the ε-element and two elements of the HPRE region in hepatitis B virus. Also the results of the Levivirus group are of particular interest: We detect various secondary structure elements that are strongly confirmed by compensatory mutations and obtain novel insight into the structural organization of Levivirus genomes that guides the life cycle of Levivirus.

The ε-structure in human hepatitis B virus was correctly predicted after both `clustalw` and `code2aln` alignment. Also the stem-loop structure α in the post-transcriptional regulatory element HPRE. Using `code2aln` we could not find the β1-element, but some other noticeable elements that, at least in one case, could also be part of HPRE.

In principle, every approach of using parameter sets that can not be optimized absolutely, is heuristic and arbitrary. This means for this work that it is not possible to build too complex scoring functions and we have severe restrictions to moderate extensions of the scoring function. The addition of amino acid scores for all three possible frames for each nucleotide, is the optimal introduction of more complexity to the scoring to get better alignment results that are not partially at risk to be at random, because our training set is too small in any case for a much larger parameter set that is computationally (even theoretically) not feasible to be optimized by learning. Our set

of parameters and usage of them in alignments is quite robust and universal, and is useful for all types of nucleic acid sequences.

It could be shown that the usage of genetic information about coding and non-coding sequence parts as part of the scoring function, like implemented in `code2aln`, is especially helpful when used with rather diverse data sets. In these cases we get relatively good alignment results. Using in combination with `alidot` we can detect a lot of known and unknown conserved RNA secondary structure elements.

In principle, using this method we cannot tell what the function of the conserved structure elements might be.

Nevertheless, knowledge about their location is very helpful and important to guide further studies.

For further implementations, it would be desirable to investigate if it is possible, at least with forthcoming computer resources, to extend the scoring function and to use a wider parameter set, and, in spite of that, to calculate a sufficiently broad overview (with a sufficiently large training set) of the space of possible parameter combinations for test cases.

The scoring function could be extended by an explicit term for frame shifting, in that way:

$$S = S_N + \sum_{i=1}^{3} S_{A_i} + \sum_{i=1}^{3} S_{FShift_i} \tag{16}$$

where S is the total score of the considered pair of nucleotides, S_N is the nucleic acid part of the scoring, and the sum of the three amino acid scores S_{A_i} is the coding of the pair of nucleic acids as the first, second, and/or third position in a base triplet. Also a sum of three scores might be considered that reflects frame shifting in three frames (S_{FShift_i}).

Instead of just looking for open reading frames one could use, for instance, hidden Markov Models to search for *likely* coding regions. These methods

could improve the discriminating between open reading frames that do not code for a protein and actual coding sequences, see e.g. [9].

The user interface is terse in the present implementation. A graphical user interface, following the example of `clustalx` would be possible.

It would certainly be very interesting to see if mutations that affect the secondary structure but are neutral at the level of the proteins, have an effect on the viability of the viruses. Unfortunately, only experimental evidence will eventually decide whether the RNA structure in some region is of importance for the viability of the virus.

Finally, the `code2aln` guide tree at present does not reflect auxiliary phylogenetic information. It tells nothing about the evolutionary distances between the sequences, it is only usable to get an order of the profile alignments.

9 Appendix A - The Codon Tables

This table shows the codon tables which ensure that fine adjustment of detection and translation of coding regions during the process of aligning is possible for various organism groups or their fitting mitochondrial sequences. The following codon tables are available:

`univ`	universal genetic code (default)
`acet`	Acetabularia
`ccyl`	Candida cylindrica
`tepa`	Tetrahymena, Paramecium, Oxytrichia, Stylonychia, Glaucoma
`eupl`	Euplotes
`mlut`	Micrococcus luteus
`mysp`	Mycoplasma, Spiroplasma
`mitocan`	canonical mitochondrial code
`mitovrt`	Vertebrates - mitochondrial code
`mitoart`	Arthropods - mitochondrial code
`mitoech`	Echinoderms - mitochondrial code
`mitomol`	Molluscs - mitochondrial code
`mitoasc`	Ascidians - mitochondrial code
`mitonem`	Nematodes - mitochondrial code
`mitopla`	Plathelminths - mitochondrial code
`mitoyea`	Yeasts - mitochondrial code
`mitoeua`	Euascomycetes - mitochondrial code
`mitopro`	Protozoans - mitochondrial code

10 Appendix B - The Program Description

10.1 The Structure Variables

```
struct FILEdata {
  char FN[256], CTN[8];
};
```

This structure contains the file name of one input file, and the name of the codon table that is used for this input file. The codon table is important for correct finding of coding regions and translation of them.

```
struct CDSdata {
  long start, stop;
  char *AAseq, *joinedSeq;
  short join;
};
```

In this structure all information is contained regarding start and stop codons of open reading frames and start and stop positions of exons. Further, in the case of exons and open reading frames that have to be joined, we have a short integer that indicates whether this coding part has to be joined to another element downstream, upstream, or if it is located between others. The last element of an array of exons (ORFs) that have to be joined, contains also the joined sequence and its translated protein sequence.

```
struct LETTERdata {
  char nucl;
  char aa[3];
};
```

This structure represents one letter in a nucleic acid sequence from the input. It contains the nucleotide and amino acid information: if the nucleotide is within one coding region or more than one, in the case of overlapping coding regions in three possible frames, then a character which indicates the according amino acid is written into the fitting position (up to three) of the character array. The encoding is as follows: If the nucleotide is the first position in a codon, then the first character in the array is the appropriate amino acid, if the nucleotide is the second position in the codon, then the second character in the array is written, and so on.

```
struct SEQdata {
  struct CDSdata *exon;
  struct CDSdata *ORF;
  struct CDSdata *found;
  struct LETTERdata *letter;
  int no;   /* starts with 1! */
  char *seq, *seqname, FN[256], CTN[8];
  int len;
};
```

This structure contains three arrays of structures of the type `CDSdata` for exons and open reading frames read from the `GenBank` file, and for ORFs which have been detected by the program. It contains also an array of structures of the type `LETTERdata` that simply represents the nucleic acid sequence after encoding and addition of coding and amino acid information. Further, an

integer which counts the sequences, and one for the length of the sequence, and a character pointer for each, the sequence, the sequence name, the file name, and the codon table name.

```
struct ALNdata {
  char ***TRmatrix;
  int ***SCmatrix;
  char **seqname;
  struct LETTERdata **alignment;
  long score;
  int no[200]; /* starts with 1! */
};
```

Each element in an array of structures of this type represents an alignment. It holds a three dimensional array of integers and one of character variables (in both cases three two-dimensional matrices). These, in summation, three dimensional matrices contain the score matrix for all Gotoh-type states and a backtracking matrix which guides the backtracking process after the scoring matrix is filled. A two dimensional character pointer contains all sequence names in the alignment, another two dimensional array of the type `LETTERdata` contains the sequences with gaps, if any. An integer array holds all numbers of the involved sequences, and a long integer gives the score.

```
struct CLUSTERdata {
  int no[200];
  int melted1[200];
  int melted2[200];
  short done;
};
```

An array of this structure encodes the clustering of the initial pairwise alignments regarding to their scores: it builds up the guide tree. The clusters are the branches of the guide tree, and they are encoded as elements of this type. The structure contains the numbers of all concerned sequences and how the cluster was made (which smaller clusters were merged to the larger one). We have also a short integer as an indicator for the program if the considered cluster has already been integrated into the tree.

10.2 The Routines

```
void start(int argc, char **argv);
```

This routine takes over the integer value `argc` which represents the number of arguments, the program was started with, and the two dimensional character array `**argv` which lists all arguments. These arguments are on one hand the paths and the names of the input sequence files and on the other hand the synonyms for the codon tables that should be used for finding the open reading frames in the sequences and for translating them. Every desired codon table follows the argument '`-c`' and is used then by the program for all the input sequence files that follow this codon table name in the command line till the next '`-c`' with an codon table name.

The standard codon table is the universal genetic code. This will be used if no other codon table name is specified and it will work well in most cases.

The sequence input files can contain one ore more sequences. They can be `GenBank` files or files in Pearson's format (FASTA) or only sequences without any format. If the sequences have distinct names within the files, these names are used. If the sequence input file contains just the sequence without any name, the name and path of the file become the sequence name.

This routine also fills an array of structures of the type `FILEdata`, one structure for each input file. It writes the desired codon table and the file name into this structure. Furthermore, it calls the routine

`short decideFN(char FN[256], char CTN[8]);`

for processing and handling `GenBank` files and files in Pearson's format. This subroutine is called once for each input file.

```
void help(short ret);
void use_me_this_way(short ret);
```

These routines both supply the user with some helping information about correct starting of the program, about the available codon tables and their shortcuts, about the synopsis and usage.

```
short decideFN(char FN[256], char CTN[8]);
```

This routine opens the input sequence files and decides whether to use the routines for processing of `GenBank` files or those for the files in other formats; and it calls these subroutines. The return value is a short integer variable that gives the number of the input sequences for reasons of internal control.

```
char *get_seq(FILE *fp, char *line, long len, short is_fp);
```

The routine is called during reading of `GenBank` files, it extracts the nucleic acid sequence of the input file. It takes over a file pointer which points at the line of the input file where the sequence data start, a character pointer with

the current read line, a long integer (the sequence length) and a short integer that is an indicator whether the input came from `stdin` or as arguments. It returns a character pointer with the read sequence.

```
short readPears(char FN[256], char CTN[8], char *line,
FILE *fp, short is_fp);
```

This function is called after the decision has been made that the current input file should be processed as a file in FASTA format or as a file without any format. The routine takes over character arrays containing the file name and the codon table name, a character pointer with the current line of the file, the file pointer and the short integer which is an indicator if the input came as arguments or from `stdin`. The file might contain one or more sequence entries with according names. The nucleic acid sequences and names are extracted from the file after the necessary memory space has been allocated. Finally, the file name, the codon table name, the sequence name, the sequence, the sequence length, and two initialized, but empty arrays for read open reading frames and exons are all together given to the following subroutine. The return value is the number of the input sequences processed so far.

```
short write2SEQ1(char FN[256], char CTN[8], char seqname[256],
char *seq, long len, struct CDSdata *ORF,
struct CDSdata *exon);
```

The file name, the codon table name, the sequence name, the sequence, the sequence length, and two initialized, but empty arrays for read open reading frames and exons are all together received by this function. And all these data are written into one element in the array of structure variables of the type `SEQdata`. The routine returns the number of the currently processed input sequence for the reason of program internal control.

```
void findORF(void);
```

This routine detects and processes open reading frames in nucleic acid sequences. First, memory space is allocated for an array of structure variables of the type `CDSdata`. These structures contain relevant information about open reading frames (see below). Elements of the structures get initialized, then the search for possible coding regions is started. The function searches start codons and, depending on them, the fitting stop codons. The used codon table is respected for appropriate recognition of codons. Then the sequences of the detected possibly coding regions get translated. Finally, all data about the found possible open reading frames (including the protein sequences) are written into the elements of the array.

```
void infoFile(void);
```

Here a file is opened for writing (`info.txt`) and many data about the input files, sequences and the read or found open reading frames and exons are written to this file for further information of the user during the program run.

```
void makePS(void);
```

A file is opened for writing and the source code for the `PostScript` output of the program is written to this file (`ORF.ps`).

```
short readGBF(char FN[256], char CTN[8], char *line,
FILE *fp, short is_fp);
```

This function reads `GenBank` files and extracts all necessary information. It takes over character arrays containing the input file names and codon table names, a character pointer with the current line of the read `GenBank` file, the file pointer, and a short integer which is an indicator whether the input

file came as an argument or from `stdin`. It calls subroutines for reading the nucleic acid and protein sequences and memory space is allocated for the arrays of the structure type `CDSdata` that then get filled with all extracted information about open reading frames and exons (after initialization with zero values). The structures also contain informations about the joining of exons to complete open reading frames or, in the case of complete ORFs, if two parts of those have to be connected beyond the ends of the nucleic acid sequence, if the sequence should be circularly closed. Finally, the input file names and codon table names, and the extracted sequence names, sequences, sequence lengths, and arrays of the structure type `CDSdata` are written into elements of the array of the type `SEQdata`, one element for each input sequence. The return value is the number of the input sequences processed so far.

```
char *readAAseq(char *line, FILE *fp, short is_fp);
```

This routine is called during the reading of the input `GenBank` file when a line is reached that contains an amino acid sequence and it extracts this sequence and returns it.

```
void mergeJOIN();
```

In the case of existence of exons or divided open reading frames that have to be joined to establish the complete coding sequence, it has to be checked whether the single parts of the coding frame are the first part or the last or one in between. These parts get joined during this routine after the allocation or reallocation of appropriate memory space. The structure variable of the type `CDSdata` contain a short integer that is an indicator if the exon or part of an open reading frame that we look at is at this first, middle, or last position. Finally, the joined sequence parts are written into the fitting element of the array of the structure variable type `SEQdata`. Also `CDSdata` is part of `SEQdata`.

```
char *translateORF(char *orf, char *CodonTableName);
```

This routine simply translates incoming nucleic acid sequences; it takes over two character pointers, one containing the nucleic acid sequence that is to be translated, the second is the currently used codon table name that is necessary for suited and correct translation. It gives back the character pointer with the protein sequence.

```
void pretranslate();
```

Here the translation of complete open reading frames and joined parts of them is coordinated, if there was no possibility to read the protein sequence from a `GenBank` file. Coding regions, joined ORFs and exons, are translated after the allocation of the desired memory space. All resulting protein sequences are written into the fitting element of the structure variable type `SEQdata`, where the nucleic acid sequences came from.

```
void encode();
```

An array of the structure variable type `LETTERdata` (see below) is established and memory is allocated. `LETTERdata` is part of `SEQdata`. Each nucleotide is written into its own element; if the nucleotide is within one coding region or more than one, in the case of overlapping coding regions in three possible frames, then a character which indicates the according amino acid is written into the fitting position (up to three) of a character array. The encoding is as follows: If the nucleotide is the first position in a codon, then the first character in the array is the appropriate amino acid, if the nucleotide is the second position in the codon, then the second character in the array is written, and so on.

```
int align();
```

This routine organizes the alignment process. First, an array of the structure variable type `ALNdata` (see below) is established and memory is allocated. Part of this structure is a two dimensional array of the type `LETTERdata` that contains an alignment. One alignment per structure element. The alignment is initialized, the routine which constructs the matching matrix is called, also the traceback routine and the function that constructs the guide tree after all pairwise alignments are done. Some output is written to `stdout` that gives information about the advance of the complete multiple alignment procedure. Finally, the multiple alignments are started. The return value is the number of the pairwise or profile alignments processed so far.

```
short matrix(int a, int b, int c);
```

The scoring and traceback matrices for the initial pairwise sequence alignments are built by this routine. Three integers which indicate the current sequences and the number of the alignment are input. A three dimensional array of integers is allocated and one of character variables (in both cases three two-dimensional matrices). These, in summation, three dimensional matrices contain the score matrix for all Gotoh-type states and a backtracking matrix which guides the backtracking process after the scoring matrix is filled. For every pair of positions of both sequences three routines are called that calculate and return the scores for each Gotoh type of states (the match- and the two gap states). Finally, it is decided which of the three two-dimensional backtracking matrices serves as the starting point for the traceback process and an indicator for that decision is returned.

```
void trace(int a, int b, int c, short TRstart);
```

The backtracking for the initial pairwise sequence alignments is performed by this routine. Three integers which indicate the current sequences and

the number of the alignment, and a short integer that indicates which backtracking matrix serves as the starting point for the traceback process, are input. One element of the structure variable of the type `LETTERdata` is initialized and defined as empty for usage as gap element. Then, step by step, the backtracking through the three two-dimensional backtracking matrices is performed and the new alignment is written.

```
void bluntEnd(int c);
```

After the backtracking each raw alignment needs to be processed by this routine due to technical reasons of the program. In principle, overhanging useless parts at the end of an alignment are cut away, if necessary.

```
struct CLUSTERdata *cluster(int c);
```

This function performs the clustering of the initial pairwise alignments regarding to their scores: it builds up the guide tree. The clusters are the branches of the guide tree, and they are encoded as elements in an array of the structure variable type `CLUSTERdata` (see below) which contain information about the associated sequences and how the cluster was made (which smaller clusters were merged to this larger one). Finally, a file is opened for writing and all information about the assembly of the guide tree is written to this file (`cluster.txt`). The return variable is one element of the array of the structure variable type `CLUSTERdata` that represents the currently processed cluster.

```
int multiple(struct CLUSTERdata *clust, int c, int a);
```

This routine takes over the array of structures of the type `CLUSTERdata` and initializes the process of the multiple alignment. It has just organizatorial and, for the program run, logistic functions. For example, it writes short

information about the progress of the complete multiple alignment to `stdout`. The return value is the number of the pairwise or profile alignments processed so far.

```
void findPartn(int c, struct CLUSTERdata clust);
```

The function takes over one element of the array of the structure variable type `CLUSTERdata` and, according to this, prepares two elements of the structure variable type `ALNdata` (see below) for alignment. Then the routine is called that constructs the matching matrices and the traceback matrices. And, finally, the backtracking is started.

```
short multMx(struct ALNdata tempaln[2], int c);
```

The scoring and traceback matrices for the profile alignment are built by this routine. It takes over two elements of the structure variable type `ALNdata` (see below) and starts their alignment; and an integer which counts the numbers of the alignments, is also accepted. A three dimensional array of integers is allocated and one of character variables (in both cases three two-dimensional matrices). These, in summation, three dimensional matrices contain the score matrix for all Gotoh-type states and a backtracking matrix which guides the backtracking process after the scoring matrix is filled. For every pair of positions of both sequences or alignments three routines are called that calculate and return the scores for each Gotoh type of states (the match- and the two gap states). Finally, it is decided which of the three two-dimensional backtracking matrices serves as the starting point for the traceback process and an indicator for that decision is returned.

```
void multTrace(struct ALNdata tempaln[2],
int c, short TRstart);
```

The backtracking for the alignments of profiles is performed by this routine. It takes over two elements of the structure variable type `ALNdata` (see below), an integer which counts the numbers of the alignments, and an indicator which backtracking matrix serves as the starting point for the traceback process. One element of the structure variable of the type `LETTERdata` is initialized and defined as empty for usage as gap element. Then, step by step, the backtracking through the three two-dimensional backtracking matrices is performed and the new alignment is written.

```
int scoreM(int a, int b, int c, int g, int h);
```

This function is called during the building of the score matrix for the match state in one of the initial pairwise sequence alignments. I takes over the numbers of the sequences, of the current alignment, and the positions that should be aligned. It calculates the score for nucleic and amino acid matches (it uses the `BLOSUM62` matrix), it writes an indicator into the appropriate traceback matrix, and it returns the score value.

```
int scoreIx(int a, int b, int c, int g, int h);
int scoreIy(int a, int b, int c, int g, int h);
```

Two short routines that are called and take over the same as the routine above and simply calculate the scores for gap opening and extension for both dimensions of the matrices (this means which sequence contains the gap). They write an indicator into the appropriate traceback matrices and return the score values.

```
int MscoreM(struct ALNdata tempaln[2], int c, int g, int h);
```

This function is called during the building of the score matrix for the match state in one of the profile alignments of groups of sequences. It takes over two elements of the structure variable type `ALNdata` (see below), an integer which counts the numbers of the alignments, and the compared positions. It calculates the score for nucleic and amino acid matches (it uses the `BLOSUM62` matrix), it writes an indicator into the appropriate traceback matrix, and it returns the score value.

```
int MscoreIx(struct ALNdata tempaln[2], int c, int g, int h);
int MscoreIy(struct ALNdata tempaln[2], int c, int g, int h);
```

Two short routines that are called and take over the same as the routine above and simply calculate the scores for gap opening and extension for both dimensions of the matrices (this means which profile contains the gap). They write an indicator into the appropriate traceback matrices and return the score values.

```
void check();
```

This routine makes a final check if there have been made changes to the input sequences during the alignment process. If an error is found, the alignment is aborted. This never should happen.

```
void output(int c);
```

The resulting, the final alignment containing, file is written and formatted.

11 Appendix C - The Manual Page

11.1 NAME

code2aln

11.2 SYNOPSIS

`code2aln [-c [CTN]] FN FN FN ... [-c [CTN]] FN FN FN ...`
where `FN` is the path and name of the input file, and `CTN` is the shortcut for the desired codon table.

11.3 DESCRIPTION

`Code2aln` produces multiple nucleic acid alignments using information on coding and non-coding regions as part of the scoring function. This is done in order to prevent the problem of higher sequence divergency on the level of nucleic acids as compared to the underlying protein sequences in the case of coding at a certain region of the input nucleic acid sequences.

`Code2aln` reads the input nucleic acid files as arguments. The possible input sequence file formats are Pearson's format (FASTA) or `GenBank` file format or sequence data in one or more lines without any format. `Code2aln` automatically detects the types of the various input sequence files and handles them accordingly. All data may be read as separate files or merged to one file.

It is possible to define one or more than one codon tables for each sequence or groups of sequences. Default is the universal genetic code. Entering '`code2aln`' without any options or input files displays a short help and a list of the various available codon tables. The codon tables are important for searching for start and stop codons and for translation of the detected open reading frames.

`Code2aln` detects all theoretically possible open reading frames which have a minimal length of 300 and one fifteenths of the sequence length. Divided coding regions and exons are joined, translated, and the amino acid sequences, beside the nucleic acid sequences, are used for the scoring function.

All pairwise alignments are done using all scoring parameters. A guide tree is built which defines the order of the profile alignments. An output file is created that gives a textual representation of this guide tree (`cluster.txt`). Further output files are `ORF.ps`, a `PostScript` display which shows the read and found open reading frames and exons, and `info.txt`, a file containing the same information in text format.

The profile alignments are done respecting the guide tree and using all scoring parameters. And, finally, the resulting multiple nucleic acid sequence alignment is written to the output file `aln.aln`.

11.4 OPTIONS

The following codon tables (and shortcuts) are available:

```
    univ: universal genetic code (default)
    acet: Acetabularia
    ccyl: Candida cylindrica
    tepa: Tetrahymena, Paramecium,
          Oxytrichia, Stylonychia, Glaucoma
    eupl: Euplotes
    mlut: Micrococcus luteus
    mysp: Mycoplasma, Spiroplasma
 mitocan: canonical mitochondrial code
 mitovrt: Vertebrates -  mitochondrial code
 mitoart: Arthropods -  mitochondrial code
 mitoech: Echinoderms -  mitochondrial code
```

```
mitomol: Molluscs -  mitochondrial code
mitoasc: Ascidians -  mitochondrial code
mitonem: Nematodes -  mitochondrial code
mitopla: Plathelminths -  mitochondrial code
mitoyea: Yeasts -  mitochondrial code
mitoeua: Euascomycetes - mitochondrial code
mitopro: Protozoans - mitochondrial code
```

11.5 VERSION

This man page documents version 1 of `code2aln`.

11.6 AUTHOR

Roman R. Stocsits

11.7 BUGS

Comments and bug reports should be sent to `roman@tbi.univie.ac.at`.

References

[1] J. P. Abrahams, M. van den Berg, E. van Batenburg, and C. Pleij. Prediction of RNA secondary structure, including pseudoknotting, by computer simulation. *Nucl. Acids Res.*, 18:3035–3044, 1990.

[2] H. W. Ackermann and M. S. DuBow. Viruses of prokaryotes. *Vol II. CRC Press*, pages 171–218, 1987.

[3] V. V. Anshelevich, A. V. Vologodskii, A. V. Lukashin, and M. D. Frank-Kamenetskii. Slow relaxational processes in the melting of linear biopolymers: A theory and its application to nucleic acids. *Biopolymers*, 23:39–58, 1984.

[4] A. R. Banerjee, J. A. Jaeger, and D. H. Turner. Thermal unfolding of a group I ribozyme: The low-temperature transition is primarily disruption of tertiary structure. *Biochemistry*, 32:153–163, 1993.

[5] G. J. Barton and M. J. E. Sternberg. A strategy for the rapid multiple alignment of protein sequences. confidence levels from tertiary structure comparisons. *J. Mol. Biol.*, 198:327–337, 1987.

[6] R. Bellman. On the theory of dynamic programming. *Proc. Natl. Acad. Sci. USA*, 38:716–719, 1952.

[7] C. K. Biebricher and R. Luce. In vitro recombination and terminal elongation of RNA by Qbeta replicase. *EMBO J.*, 11:5129–5135, 1992.

[8] C. K. Biebricher and R. Luce. Sequence analysis of RNA species synthesized by Qbeta replicase without template. *Biochemistry*, 32:4848–4854, 1993.

[9] M. Borodovsky and J. McIninch. Genmark: parallel gene recognition for both DNA strands. *Comp. & Chem.*, 17:123–133, 1993.

[10] J. Boyle, G. T. Robillard, and S. H. Kim. Sequential folding of transfer RNA. A nuclear magnetic resonance study of successively longer tRNA fragments with a common 5' end. *J. Mol. Biol.*, 139:601–625, 1980.

[11] N. Breton, C. Jacob, and P. Daegelen. Prediction of sequentially optimal RNA secondary structures. *J. Biomol. Struct. Dyn.*, 14:772–740, 1997.

[12] C. Büchen-Osmond. `ICTVdB Virus Database`, the universal virus database of the international committee on taxonomy of viruses. `http://www.ncbi.nlm.nih.gov/ICTVdb`, 2002. (database).

[13] K. Bucka-Lassen, O. Caprani, and J. Hein. `ComAlign`. `http://www.daimi.au.dk/ ocaprani`, 1999. (Free Software).

[14] K. Bucka-Lassen, O. Caprani, and J. Hein. Combining many multiple alignments in one improved alignment. *Bioinformatics*, 15:122–130, 1999.

[15] J. H. Cate, A. R. Gooding, E. Podell, K. Zhou, B. L. Golden, A. A. Szewczak, C. D. Kundrot, T. R. Cech, and J. H. Doudna. Crystal structure of a group I ribozyme domain: Principles of RNA packing. *Science*, 273:1678–1685, 1996.

[16] J. H. Cate, A. R. Gooding, E. Podell, K. Zhou, B. L. Golden, A. A. Szewczak, C. D. Kundrot, T. R. Cech, and J. H. Doudna. RNA tertiary structure mediation by adenosine platforms. *Science*, 273:1696–1699, 1996.

[17] T. R. Cech and B. L. Bass. Biological catalysis by RNA. *Annu. Rev. Biochem.*, 55:599–630, 1986.

[18] K. M. Chao. `Calign`. `http://iubio.bio.indiana.edu`, 1999. (Free Software).

[19] K. M. Chao. Calign: aligning sequences with restricted affine gap penalties. *Bioinformatics*, 15:298–304, 1999.

[20] M. Chastain and I. Tinoco. Nucleoside triples from the group I intron. *Biochemistry*, 32:14220–14228, 1993.

[21] D. M. Crothers, P.E. Cole, C. W. Hilbers, and R. G. Shulman. The molecular mechanism of thermal unfolding of *escherichia coli* formylmethionine transfer RNA. *J. Mol. Biol.*, 87:63–88, 1974.

[22] J. A. Cuff, E. Birney, M. E. Clamp, and G. J. Barton. `ProtEST`. `http://barton.ebi.ac.uk/servers/protest.html`, 2000.

[23] J. A. Cuff, E. Birney, M. E. Clamp, and G. J. Barton. ProtEST: protein multiple sequence alignments from expressed sequence tags. *Bioinformatics*, 16:111–116, 2000.

[24] Jan Cupal, Ivo L. Hofacker, and Peter F. Stadler. Dynamic programming algorithm for the density of states of RNA secondary structures. In R. Hofstädt, T. Lengauer, M. Löffler, and D. Schomburg, editors, *Computer Science and Biology 96 (Proceedings of the German Conference on Bioinformatics)*, pages 184–186, Leipzig (Germany), 1996. Univeristät Leipzig.

[25] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. *Atlas of Protein Sequence and Structure*, volume 5 of *3*, chapter 3, pages 345–352. NBRF Washington, 1978.

[26] D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, 25:351–360, 1987.

[27] W. Fiers. Structure and function of RNA bacteriophages. *Comprehensive Virology*, 13:69–204, 1979.

[28] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[29] W. Fontana and P. Schuster. Continuity in evolution. on the nature of transitions. *Science*, 280:1451–1455, 1998.

[30] W. Fontana, P. F. Stadler, E. G. Bornberg-Bauer, T. Griesmacher, I. L. Hofacker, M. Tacker, P. Taranzona, E. D. Weinberger, and P. Schuster. RNA folding and combinatory landscapes. *Phys. Rev. E*, 47:2083–2099, 1993.

[31] S. M. Freier, R. Kierzek, J. A. Jaeger, N. Sugimoto, M. H. Caruthers, T. Neilson, and T. H. Turner. Improved free energy parameters for prediction of RNA duplex stability. *Proc. Natl. Acad. Sci. USA*, 83:9373–9377, 1986.

[32] T. C. Gluick and D. E. Draper. Thermodynamics of a pseudoknotted mRNA fragment. *J. Mol. Biol.*, 241:246–262, 1994.

[33] W. B. Goad and M. I. Kanehisa. Pattern recognition in nucleic acid sequences. A general method for finding local homologies and symmetries. *Nucl. Acids Res.*, 10:247–263, 1982.

[34] J. Gorodkin, L. J. Heyer, and G. D. Stormo. Finding the most significant common sequence and structure motifs in a set of rna sequences. *Nucleic Acids Res.*, 25:3724–3732, 1997.

[35] O. Gotoh. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *CABIOS*, 9:361–370, 1993.

[36] Brad Gulko and David Haussler. Using multiple alignments and phylogenetic trees to detect RNA secondary structure. In L. Hunter and

T. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 350–367, Singapore, 1996. World Scientific.

[37] A. P. Gultyaev. The computer simulation of RNA folding involving pseudoknot formation. *Nucl. Acids Res.*, 19:2489–2494, 1991.

[38] A. P. Gultyaev, F. H. D. van Batenburg, and C. W. A. Pleij. The computer simulation of RNA folding pathways using a genetic algorithm. *J. Mol. Biol.*, 250:37–51, 1995.

[39] A. P. Gultyaev, F. H. D. van Batenburg, and C. W. A. Pleij. Dynamic competition between alternative structures in viroid RNAs simulated by an RNA folding algorithm. *J. Mol. Biol.*, 276:43–55, 1998.

[40] E. Halperin, S. Faigler, and R. Gill-More. `FramePlus`. `ftp.compugen.co.il/pub/research`, 1999. (Free Software).

[41] E. Halperin, S. Faigler, and R. Gill-More. Frameplus: aligning DNA to protein sequences. *Bioinformatics*, 15:867–873, 1999.

[42] R. W. Hamming. *Coding and Information Theory*, pages 44–47. Prentice-Hall, Englewood Cliffs, 2 edition, 1989.

[43] Kyungsook Han and Hong-Jin Kim. Prediction of common folding structures of homologous RNAs. *Nucl. Acids Res.*, 21:1251–1257, 1993.

[44] T. P. Hausner, J. Atmadja, and K. H. Nierhaus. Evidence that the G2661 region of 23S rRNA is located at the ribosomal binding site of both elongation factors. *Biochemie*, 69:911–923, 1987.

[45] L. He, R. Kierzek, J. SantaLucia, A. E. Walter, and D. H. Turner. Nearest-neighbour parameters for G-U mismatches. *Biochemistry*, 30:11124–11132, 1991.

[46] R. Hecker, Z. Wang, G. Riesner, and D. Steger. Analysis of RNA structures by temperature-gradient gel electrophoresis: Viroid replication and processing. *Gene*, 72:59–74, 1988.

[47] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.

[48] D. Herschlag. RNA chaperones and the RNA folding problem. *J. Biol. Chem.*, 270:20871–20874, 1995.

[49] P. G. Higgs and S. R. Morgan. Thermodynamics of RNA folding. when is an RNA molecule in equilibrium. In F. Morán, A. Moreno, J. J. Merelo, and Chacón, editors, *Advances in Artificial Life*, pages 852–861, Berlin, 1995. ECAL 95, Springer Verlag.

[50] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Comm. Assoc. Comp. Mach.*, 18:341–343, 1975.

[51] I. L. Hofacker, M. Fekete, C. Flamm, M. A. Huynen, S. Rauscher, P. E. Stolorz, and P. F. Stadler. Automatic detection of conserved RNA structure elements in complete RNA virus genomes. *Nucl. Acids Res*, 26:3825–3863, 1998.

[52] I. L. Hofacker, W. Fontana, P. F. Stadler, and P. Schuster. `Vienna RNA Package`. `http://www.tbi.univie.ac.at/~ivo/RNA/`, 1994. (Free Software).

[53] I. L. Hofacker and P. F. Stadler. Automatic detection of conserved base pairing patterns in RNA virus genomes. *Comp. & Chem.*, 23:401–414, 1999.

[54] Ivo L. Hofacker, Walter Fontana, Peter F. Stadler, Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast folding and comparison of RNA secondary structures. *Monatsh. Chemie*, 125:167–188, 1994.

[55] Ivo L. Hofacker, Martijn A. Huynen, Peter F. Stadler, and Paul E. Stolorz. Knowledge discovery in RNA sequence families of HIV using scalable computers. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR*, pages 20–25, Menlo Park, CA, 1996. AAAI Press.

[56] P. Hogeweg and B. Hesper. Energy directed folding of RNA sequences. *Nucl. Acids Res.*, 12:67–74, 1984.

[57] I. Holmes and W. J. Bruno. Evolutionary HMMs: a Bayesian approach to multiple alignment. *Bioinformatics*, 17:803–820, 2001.

[58] I. Holmes and W. J. Bruno. `Handel`. `http://www.biowiki.org/Handel`, 2001. (Free Software).

[59] J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Technical Report Comp. Sci. 41, Bell Laboratories, 1976.

[60] J. A. Jaeger, D. H. Turner, and M. Zuker. Improved predictions of secondary structures for RNA. *Proc. Natl. Acad. Sci. USA*, 86:7706–7710, 1989.

[61] B. R. Jordan. Computer generation of pairing schemes for RNA molecules. *J. Theor. Biol.*, 34:363–378, 1972.

[62] G. F. Joyce. In vitro evolution of nucleic acids. *Curr. Opin. Struct. Biol.*, 4:331–336, 1994.

[63] V. Juan and C. Wilson. RNA secondary structure prediction based on free energy and phylogenetic analysis. *J. Mol. Biol.*, 289:935–947, 1999.

[64] M.I. Kanehisa and W. B. Goad. Pattern recognition in nucleic acid sequences. An efficient method for finding locally stable secondary structures. *Nucl. Acids Res.*, 10:265–277, 1982.

[65] A. H. Kidd and K. Kidd-Ljunggren. A revised secondary structure model for the 3'-end of hepatitis B virus pregenomic RNA. *Nucl. Acids Res.*, 24:3295–3301, 1996.

[66] K. Kidd-Ljunggren, Y. Miyakawa, and A. H. Kidd. Genetic variability in hepatitis B viruses. *Journal of General Virology*, 83:1267–1280, 2002.

[67] M. Kunze and G. Thierrin. Maximal common subsequences of pairs of strings. *Congr. Num.*, 34:299–311, 1982.

[68] S-Y. Le and M. Zuker. Predicting common foldings of homologous RNAs. *J. Biomolecular Structure & Dynamics*, 8:1027–1044, 1991.

[69] C. Lee, C. Grasso, and M. F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18:452–464, 2002.

[70] C. Lee, C. Grasso, and M. F. Sharlow. `POA`. `http://www.bioinformatics.ucla.edu/poa`, 2002. ().

[71] H. P. Lenhof, B. Morgenstern, and K Reinert. `DIALIGN`. `http://bibiserv.TechFak.Uni-Bielefeld.DE/dialign/`, 1999.

[72] H. P. Lenhof, B. Morgenstern, and K Reinert. An exact solution for the segment-to-segment multiple sequence alignment problem. *Bioinformatics*, 15:203–210, 1999.

[73] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci. USA*, 86:4412–4415, 1989.

[74] P. Loss, M. Schmitz, G. Steger, and D. Riesner. Formation of a thermodynamically metastable structure containing hairpin II is critical for infectivity of potato spindle tuber viroid RNA. *EMBO J.*, 10:719–727, 1991.

[75] M. Lu and D. E. Draper. Bases defining an ammonium and magnesium ion-dependent tertiary structure within the large subunit ribosomal RNA. *J. Mol. Biol.*, 244:572–585, 1994.

[76] R. Lück, S. Gräf, and G. Steger. ConStruct: A tool for thermodynamic controlled prediction of conserved secondary structure. *Nucl. Acids Res.*, 27:4208–4217, 1999.

[77] R. Lück, G. Steger, and D. Riesner. Thermodynamic prediction of conserved secondary structure: Application to the RRE element of HIV, the tRNA-like element of CMV, and the mRNA of prion protein. *J. Mol. Biol.*, 258:813–826, 1996.

[78] A. V. Lukashin and J. J. Rosa. Local multiple sequence alignment using dead-end elimination. *Bioinformatics*, 15:947–953, 1999.

[79] C. W. Mandl, H. Holzmann, T. Meixner, S. Rauscher, P. F. Stadler, S. L. Allison, , and F. X. Heinz. Spontaneous and engineered deletions in the 3' noncoding region of tick-borne encephalitis virus: construction of highly attenuated mutants of a flavivirus. *J. Virology*, 72:2132–2140, 1998.

[80] H. M. Martinez. An RNA folding rule. *Nucl. Acids Res.*, 12:323–324, 1984.

[81] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990.

[82] D. L. Mills, editor. *A new algorithm to determine the Levenshtein distance between two strings*, Conference on Sequence Comparison. University of Montreal, 1978.

[83] B. Morgenstern. A space-efficient algorithm for aligning large genomic sequences. *Bioinformatics*, 16:948–949, 2000.

[84] E. W. Myers and W. Miller. Optimal alignments in linear space. *CABIOS*, 4:11–17, 1988.

[85] S. B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[86] J. M. Norman. *Elementary dynamic programming*. Crane, Russak and Co., New York, 1975.

[87] R. Nussinov and A. B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proc. Natl. Acad. Sci. USA*, 77:6309–6313, 1980.

[88] R. Nussinov, I. Tinoco, and A. Jacobsen. Secondary structure model for the complete simian virus 50 late precursor RNA. *Nucl. Acids Res.*, 10:351–363, 1982.

[89] R. Nussinov, I. Tinoco, and A. B. Jacobson. Small changes in free energy assignments for unpaired bases do not affect predicted secondary structures in single stranded RNA. *Nucl. Acids Res.*, 10:341–349, 1982.

[90] S. Pascarella and P. Argos. Analysis of insertions/deletions in protein structures. *J. Mol. Biol.*, 224:461–471, 1992.

[91] S. Rauscher, C. Flamm, C. Mandl, F. X. Heinz, and P. F. Stadler. Secondary structure of the 3' noncoding regions of flavivirus genomes:

Comparative analysis of base pairing probabilities. *RNA*, 3:779–791, 1997.

[92] K. Reinert, J. Stoye, and T. Will. An iterative method for faster sum-of-pairs multiple sequence alignment. *Bioinformatics*, 16:808–814, 2000.

[93] A. Rieger and M. Nassal. Distinct requirements for primary sequence in the 5'-and 3'-part of a bulge in the hepatitis B virus RNA encapsidation signal revealed by a combined in vivo selection/in vitro amplification system. *Nucl. Acids Res.*, 23:3909–3915, 1995.

[94] M. A. Roytberg, A. Y. Ogurtsov, S. A. Shabalina, and A. S. Kondrashov. A hierarchical approach to aligning collinear regions of genomes. *Bioinformatics*, 18:1673–1680, 2002.

[95] M. A. Roytberg, A. Y. Ogurtsov, S. A. Shabalina, and A. S. Kondrashov. `OWEN`. `ftp://ftp.ncbi.nih.gov/pub/kondrashov/owen`, 2002. (Free Software).

[96] D. Sankoff, R. J. Cedergren, and G. Lapalme. Frequency of insertion-deletion, transversion and transition in the evolution of 5S ribosomal RNA. *J. Mol. Evol.*, 7:133–149, 1976.

[97] D. Sankoff, C. Morel, and R. J. Cedergren. Evolution of 5S RNA and the nonrandomness of base replacement. *Nature New Biology*, 245:232–234, 1973.

[98] P. Schuster. How to search for RNA structures. theoretical concepts in evolutionary biotechnology. *J. Biotechnology*, 41:239–258, 1995.

[99] P. Schuster, W. Fontana, P. F. Stadler, and I. L. Hofacker. From sequences to shapes and back: A case study in RNA secondary structures. *Proc. R. Soc. Lond. B*, 255:279–284, 1994.

[100] G. J. Smith, J. E. Donello, R. Lueck, G. Steger, and T. J. Hope. The hepatitis B virus post-transcriptional regulatory element contains two conserved RNA stem-loops which are required for function. *Nucl. Acids Res.*, 26,:4818–4827, 1998.

[101] R. R. Stocsits. Improved alignments based on a combination of amino acid and nucleic acid sequence information. *diploma thesis*, 1999.

[102] R. R. Stocsits. Detection of conserved RNA secondary structures: Hepatitis B as an example. *Proceedings of the Complex Systems Summer School in Santa Fe CSSS02*, 2002.

[103] R. R. Stocsits, I. L. Hofacker, and P. F. Stadler. Conserved secondary structures in hepatitis B virus RNA. *Proceedings of the German Conference on Bioinformatics GCB 1999*, pages 73–79, 1999.

[104] J. Stoye. Multiple sequence alignment with the divide-and-conquer method. *Gene*, 211:GC45–GC56, 1998.

[105] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignments through sequence weighting, position specific gap penalties and weight matrix choice. *Nucl. Acids Res.*, 22:4673–4680, 1994.

[106] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Improved sensitivity of profile searches through the use of sequence weights and gap excision. *CABIOS*, 10:19–29, 1994.

[107] J. L. Thorne, H. Kishino, and J. Felsenstein. An evolutionary model for maximum likelihood alignment of DNA sequences. *J. Mol. Evol.*, 33:114–124, 1991.

[108] U. Tönges, S. W. Perrey, J. Stoye, and A. W. M. Dress. A general method for fast multiple sequence alignment. *Gene*, 172:GC33–GC41, 1996.

[109] U. Tönges, S. W. Perrey, J. Stoye, and A. W. M. Dress. `DCA`. `http://bibiserv.techfak.uni-bielefeld.de`, 2000.

[110] D. H. Turner, N. Sugimoto, and S. Freier. RNA structure prediction. *Ann. Rev. Biophys. Chem.*, 17:167–192, 1988.

[111] K. Valegaard, L. Liljas, K. Fridborg, and T. Unge. The three-dimensional structure of the bacterial virus MS2. *Nature*, 345:36–41, 1990.

[112] J. van Duin. Single-stranded RNA phages (Leviviridae). *Encyclopedia of Virology*, pages 1663–1668, 1999.

[113] M. Vingron and M. S. Waterman. Sequence alignment and penalty choice. Review of concepts, case studies and implications. *J. Mol. Biol.*, 235:1–12, 1994.

[114] A. E. Walter, D. H. Turner, J. Kim, M. H. Lyttle, P. Mueller, D. H. Mathews, and M. Zuker. Co-axial stacking of helices enhances binding of oligoribonucleotides and improves predictions of RNA folding. *Proc. Natl. Acad. Sci. USA*, 91:9218–9222, 1994.

[115] M. S. Waterman. *Studies on foundation and combinatorics, advances in mathematics supplementary studies*, volume 1, chapter secondary structures of single stranded nucleic acids, pages 167–212. Academic Press N. Y., 1978.

[116] M. S. Waterman and T. Byers. A dynamic programming algorithm to find all solutions in the neighborhood of the optimum. *Math. Biosci.*, 77:179–188, 1985.

[117] W. J. Wilbur and D. J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proc. Natl. Acad. Sci. USA*, 80:726–730, 1983.

[118] M. Zuker and D. Sankoff. RNA secondary structures and their prediction. *Bull. Math. Biol.*, 46(4):591–621, 1984.

[119] M. Zuker and P. Stiegler. Optimal computer folding of larger RNA sequences using thermodynamics and auxiliary information. *Nucl. Acids Res.*, 9:133–148, 1981.

Curriculum Vitae

Name:	Roman Rudolf Stocsits
Date of Birth:	1971-02-04
Place of Birth:	Vienna
1977 - 1981:	Elementary School, Volkschule Knöllgasse
1981 - 1985:	Secondary School, BG V Rainergasse
1985 - 1989:	Classical Secondary School, Humanistisches Gymnasium V
1989:	School Leaving Certificate, passing with distinction
1989:	Entering University of Vienna, study of biology and molecular biology
1994, Nov. 15th:	Achieving B.Sc.
1998:	Start of diploma work at the Institute for Theoretical Chemistry, working with Ao. Prof. Dr. Peter F. Stadler on the field of multiple sequence alignments
1999:	Achieving M.Sc. (Magister rerum naturalium) and start of work as a Ph.D. student with Prof. Dr. Peter F. Stadler on the field of multiple sequence alignments and extended scoring functions
Current Address:	Angeligasse 59/14 A-1100 Vienna Tel.: +43 1 941 46 31 e-mail: roman@tbi.univie.ac.at