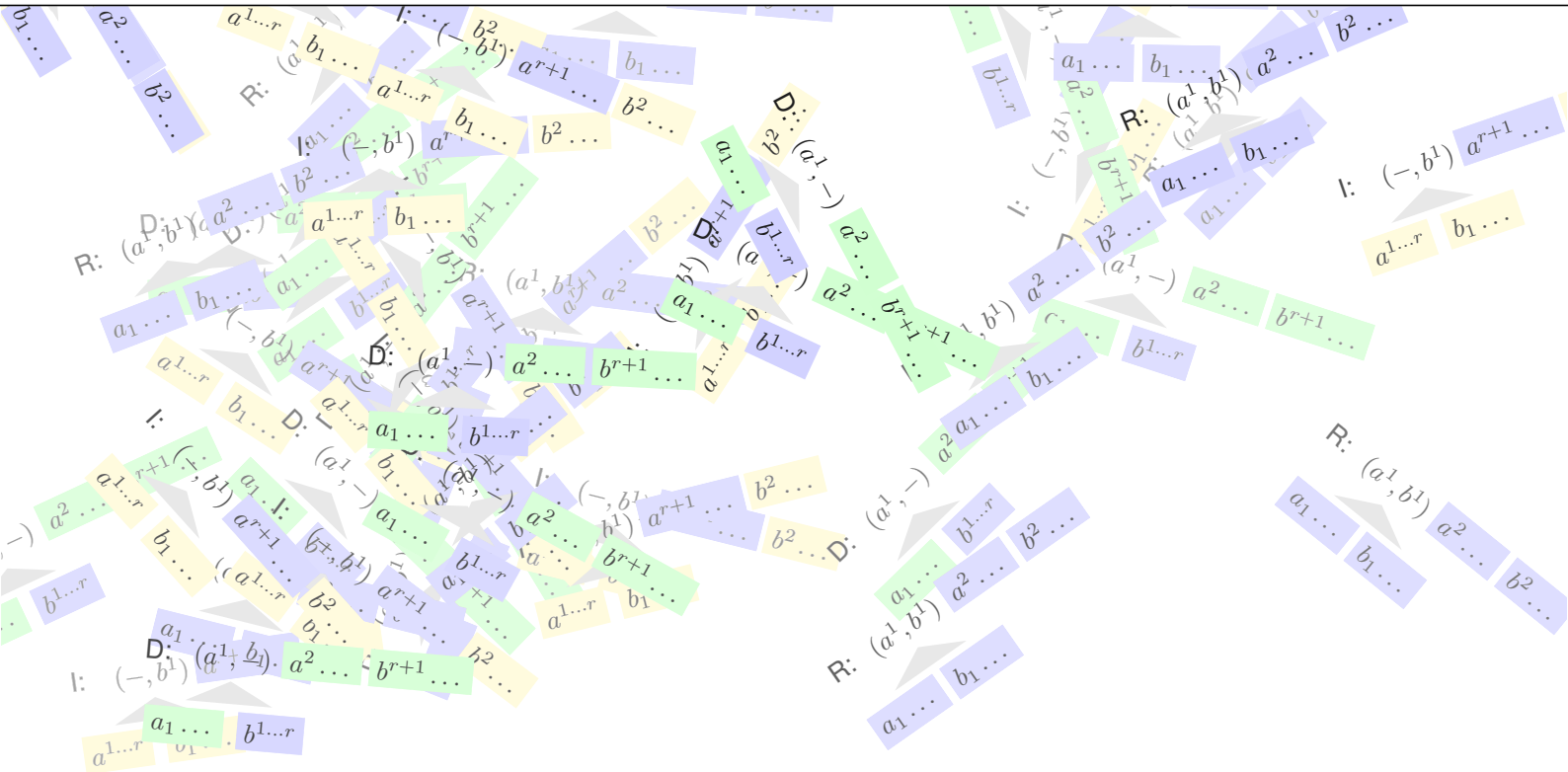
Universität Bielefeld

Technical Faculty
Practical Computer Science

Comparing Forests

A thesis submitted for the degree of Doctor rerum naturalium.

Stefanie Schirmer
August 2011



Printed on age-resistant paper according to DIN-ISO 9706.

Acknowledgements

Thanks to Robert Giegerich and Ellen Baake for their supervision during my PhD work. Apart from letting me learn a lot from your lectures and valuable discussions, both of you are inspiring personalities for me when it comes to making complex ideas easy to understand.

Many thanks to all proofreaders and helpers: Stefan Janssen, Jan Reinkensmeier, Ute von Wangenheim & Timo, Sebastian Zehe, Jane Tingley, Georg Sauthoff and all members of the Practical Computer Science group.

I also thank Jan Anlauff, my girls and my family for helping me not to get lost in the forest.

Various methods have been proposed for RNA secondary structure comparison, and new ones are still being developed. It seems that there is no appropriate distance measure for structure comparison yet.

Such a distance measure should appropriately capture the features of the secondary structure to determine the distance. Thus, the measure also depends on the representation of the secondary structure, which may introduce artefacts in the distance computation. Our goal is to find a distance function that avoids artefacts caused by the representation, and is based on a reasonable representation of the secondary structure. After a discussion of common distance functions for RNA secondary structures, we focus on the forest alignment distance, which represents the secondary structures in a natural way with regard to the nesting and adjacency relation of substructures.

In the main part of this work, we extend the gap model of the forest alignment distance to make it suitable for affine gap costs. This leads to a new algorithm variant, which is explained in this thesis, and is implemented in the new version of the tool *RNAforester 2.0*.

In addition, we provide a mechanism to speed up the alignment process by anchoring of subalignments. The anchoring information is based on the overall shape of the molecule, and is obtained by the method of abstract shape analysis.

Another contribution is the discussion of the well-formed RNA forest alignment concept. I adapt the case distinction in the recurrences that were designed to construct well-formed RNA forest alignment, to ensure that the deletion and insertion of a pairing relation between two bases is handled in an appropriate way.

The affine gap scoring scheme brings an additional constant factor of ≈ 7 into the computation. It improves structure alignments in many cases, if combined with the right scoring parameters. The anchoring of subalignments leads to an average speedup of factor ≈ 3 compared to the usual computation, dependent on number and placement of the anchors.

Contents

1 Introduction

1.1 Motivation: the RNA structure alignment problem

To determine the relations within a set of RNA molecules regarding their structural similarity, the problem of RNA secondary structure comparison has to be solved. To solve the structure comparison problem for two given input structures, a distance measure is applied which computes the distance between these two input structures based on secondary structure features such as paired bases or more abstract building blocks.

Various methods have been proposed for RNA secondary structure comparison, and new ones are still being developed. It seems that there is no appropriate distance measure for structure comparison yet.

Such a distance measure should appropriately capture the features of the secondary structure to determine the distance. Thus, the measure also depends on the representation of the secondary structure, which may introduce artefacts in the distance computation.

Our goal is to find a distance function that avoids artefacts caused by the representation, and is based on a reasonable representation of the secondary structure. The common distance functions for RNA secondary structure comparison were not always able to avoid these artefacts.

In the discussion of common distance measures we will address their strengths and weaknesses, and focus on the forest alignment distance, which represents RNA secondary structures as forests and performs a forest alignment to compute the distance between two secondary structures. We classify the forest alignment distance as one of the best suited distance measures to capture the features of an RNA secondary structure without introducing artefacts. The forest alignment distance is analyzed in depth in this thesis, the underlying structure alignment algorithm is extended to a new affine gap cost model, and an anchored alignment variant is developed, which uses additional information constraining the alignment as a speedup opportunity.

1.2 Introduction: RNA, a molecular contortionist

The RNA molecule RNA (ribonucleic acid) is a chain molecule of nucleotides, each one consisting of a ribose with one of the bases adenine (A), cytosine (C), guanine (G) and uracil (U), with an additional phosphate group. Adenin and guanin are purines with two ring structures as the base, cytosine and uracil are pyrimidines with one ring structure as the base. See Figure 1.1 for the chemical structure of the four bases. The nucleotides are linked to a chain molecule via the sugar-phosphate backbone (depicted in Figure 1.2).

The molecule can fold back onto itself, forming basepairings and intricate structures. The basepairings are established by hydrogen bonds between two bases. The bases that pair in the standard Watson-Crick basepairs are adenine and uracil, pairing via three hydrogen bonds, and guanine and cytosine, pairing via three hydrogen bonds. A less stable pair between guanine and uracil, called the GU wobble pair, is also possible.

Other interactions are also possible, but occur more rarely and are based on subsequences with certain bases (sequence motifs) [70]. The stable energy of the structure does not mainly

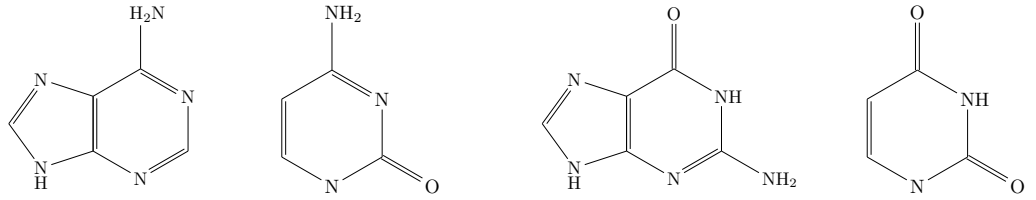


Figure 1.1: The four bases adenine (A), cytosine (C), guanine (G), and uracil (U). Together with the phosphate-sugar backbone (not shown) they are the building blocks of RNA.

come from the hydrogen bonds, but rather from neighboring basepairs that stack on each other supported by Van-der-Waals forces and π electron interaction at the purine and pyrimidine rings, building stable helix structures.

Structure-wise, an RNA molecule in our model has a primary structure which is just the sequence of nucleotides in the molecule, a secondary structure which adds pairing relations to the bases, and a tertiary structure which determines the three dimensional form (see Figure 1.3).

Properties of RNA RNA molecules are known to be integral parts of the cellular machinery for protein biosynthesis and RNA processing. They also work as catalysts [18], control gene expression, or sense and communicate responses to cellular signals. The vast diversity of functions implies structural complexity.

During the course of evolution, many RNA molecules conserve their structure more than they conserve their sequence. Thus, it is reasonable to compare RNAs by their structure rather than by sequence comparison. Therefore, a method for RNA structure comparison is needed.

As the secondary structure of an RNA molecule is the premise for its tertiary structure, and is also easier to obtain, we will focus on the comparison of RNA secondary structures.

RNA 2D structure The secondary (or 2D) structure level is an intermediate level of structural abstraction. In terms of information, an RNA secondary structure consists of the primary structure, i.e. the nucleotide sequence, plus additional basepairing information, i.e. a list of bases which are paired by hydrogen bonds in the molecule. If we restrict these basepairings to be not crossing, we can represent each secondary structure as a tree, as all structural elements are either nested or adjacent.

Biological significance of RNA structure comparison Separate analysis of single biosequences is not suitable for studying them in a wider context. Comparative analysis provides this context by giving insights about the relations within a set of sequences. Therefore, a central discipline in computational biology is sequence comparison. Sequence comparison is closely linked with the idea of producing an alignment of sequences, in order to find similarities and dissimilarities in the compared sequences.

However, the functionality of most biomolecules, such as RNA and Proteins, depends on their structure rather than their sequence in the first place (see e.g. [129] and several bioinformatics textbooks). This structure, and along with it, the function, may be preserved over the course of

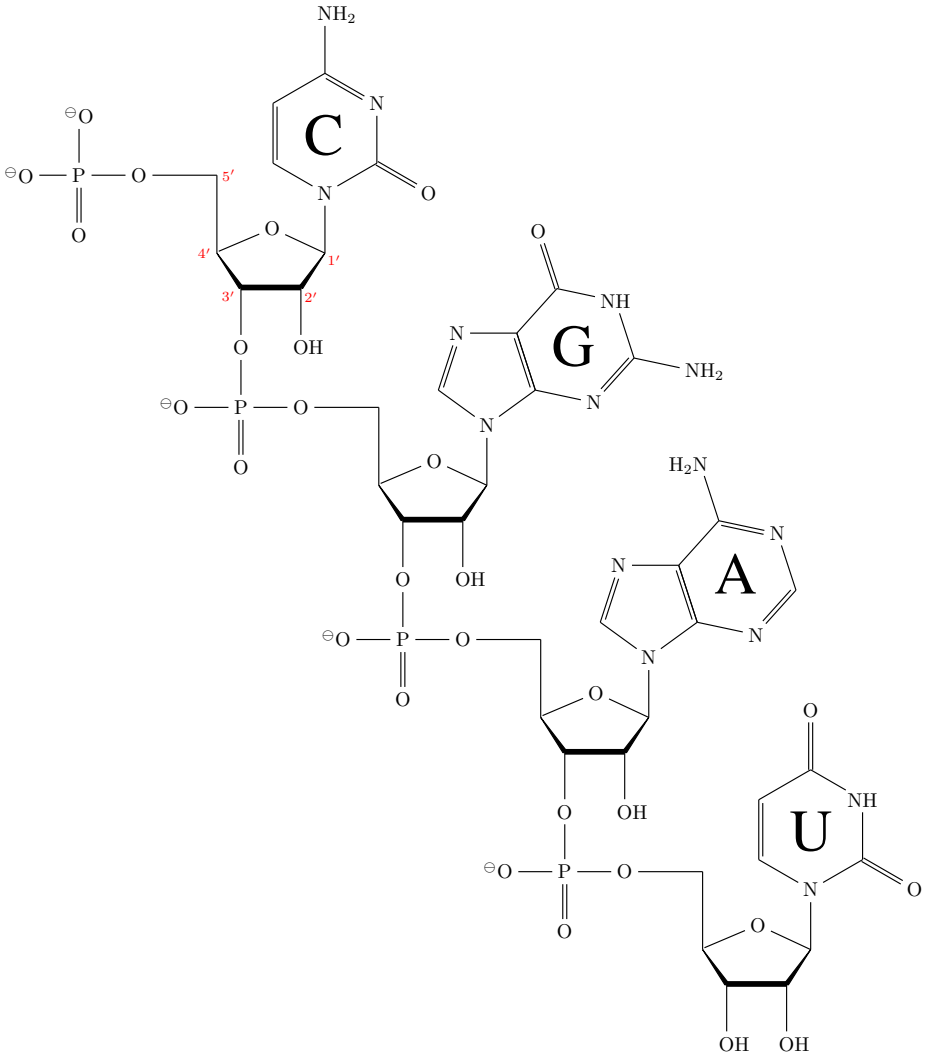


Figure 1.2: By the sugar and phosphate backbone, the bases are linked to a continuous chain, the RNA molecule. The order in which the bases appear in sequence encodes information and determines the structure of the molecule. The phosphodiester bonds link the 5' and the 3' carbon atoms of two neighboring ribose sugar molecules (depicted in red). An order in the chain molecule is imposed by the enzyme polymerase, which adds nucleotides to the 3' end, making RNAs grow in 5'-3' direction.

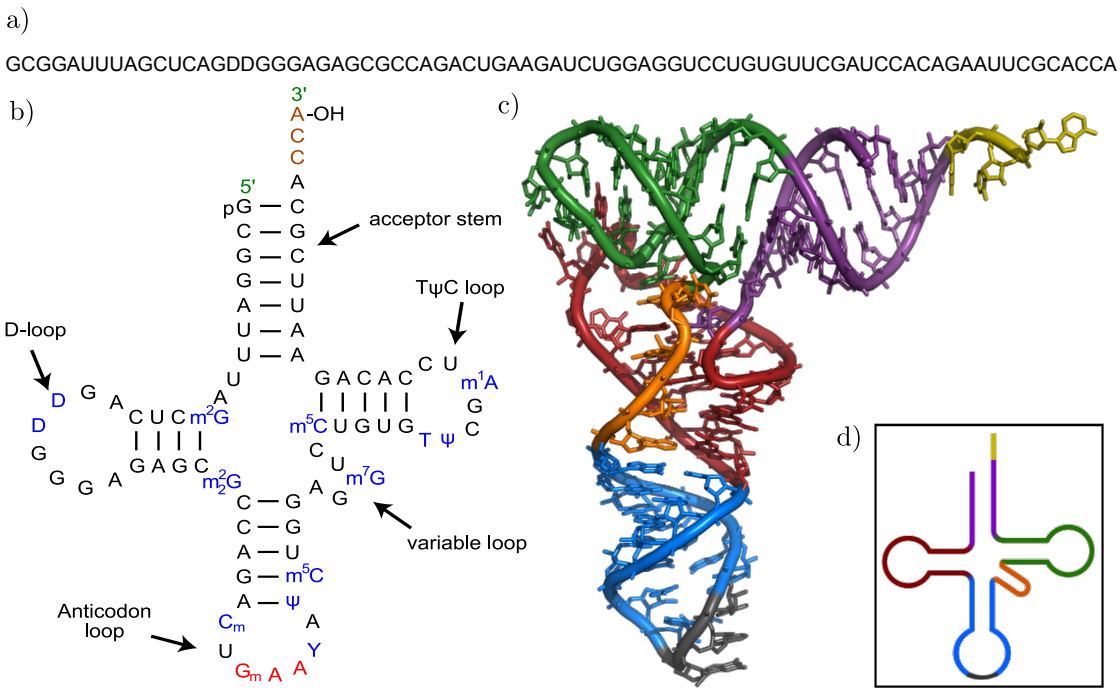


Figure 1.3: The three structure levels of an RNA molecule. The example molecule is the t-RNA for phenylalanine in yeast. a) On top is the primary structure. b) On the left is the secondary structure with base modifications. Modified bases are shown in blue. The anticodon is shown in red. c) On the right is the tertiary structure with a color-mapping of the parts. The structural parts encoded in color are the CCA tail in yellow, the acceptor stem in purple, variable loop in orange, D arm in red, Anticodon arm in blue with Anticodon in black, T arm in green. d) On the bottom right is also a map which indicates the position of the secondary structure elements in the tertiary structure by a color code. The example is modified from [130].

evolution by compensatory basepair replacements. These replacements conserve the basepairing structure, while changing the sequence by replacing the former pairing bases by two other pairing ones. Due to structural conservation in not so closely related organisms, a comparison by structure can give a better measure of relatedness than simple sequence comparison in many cases.

The prediction of the tertiary structure of an RNA molecule from its primary structure is known to be an infeasible problem in computational biology, although research is done on this problem. There exist, however, several algorithms and programs to predict the secondary structure of a molecule, for example by free energy minimization (see [145]). This approach presumes that an RNA molecule always folds into the structure with minimum free energy (MFE structure).

The biologically relevant structure is not always the optimal one in terms of computed free energy [80]. Computational approaches to RNA folding by minimum free energy may suffer from incorrectly measured stacking energies as energy parameters for the folding algorithms [51, 80]. These parameters are measured under lab conditions and may depend on temperature, overall size of the molecule, and other factors the folding algorithms do not account for. Also, the MFE methods do not take modifications of bases into account, such as e.g. pseudouridine (denoted Ψ), which is the most common modified nucleoside in RNA [19]. It is present in most RNA classes and is an isomer of the nucleoside uridine (uracil and sugar), which is modified post-transcriptionally by a Ψ-synthetase. Another difficult case for MFE folding are conformational switches [12], also called riboswitches. These RNA molecules have two energetically favorable structures, and may switch their conformation between these to have different functionality. The in vivo folding process may also depend on other external factors as well, e.g. on the RNA folding chaperon Hfq.

Hence, we also have to take suboptimal solutions of folding the same molecule into account, and are interested in the structural divergence of these suboptimal solutions. Determining the structural divergence requires structural comparison of RNA.

Of course, structure comparison is also required for the analysis of structural conservation between functionally related RNA sequences across different species.

Posed problems To solve these problems, we need a way to compare the secondary structures of an ensemble of (suboptimal) structures. Hence, a measure of structural similarity expressed as a distance is required.

A variety of measures exist for comparing RNAs on the level of secondary structures. A good overview over some basic metrics on RNA secondary structures is given in [85]. Several strategies for measuring secondary structure similarity were also analyzed by Gruber et al. [37]. We study various strategies, and focus on the tree alignment distance in depth.

The tree alignment distance The tree alignment distance was first introduced in [55]. In spite of their equivalency on sequences, the edit distance [103] and the alignment distance on trees refer to two different concepts.

The common definitions of both tree edit and tree alignment distance are based on linear gap costs, scoring each gap node in a tree individually. One of our contributions is to extend the tree

alignment distance algorithm to work with affine gap costs, as they model the biological reality in a more detailed way.

Gap costs Gap costs affect the overall score of the alignment, and determine whether mismatching characters are aligned or a gap is introduced. Affine gap costs (or gap penalties, gap scores) penalize gaps using a linear function depending on the length of the gap. The cost function in this case receives two scoring parameters, an initial gap opening cost and a lower gap extension cost, to favor gap extension over the opening of a new gap.

Biologically, the affine cost model is more reasonable, as it is more likely to have one bigger gap by one insertion or deletion event than having many small gaps (and thus multiple deletion/insertion events) in close vicinity, because deletion/insertion mainly happens due to slipping of the DNA polymerase, or (more rarely) by unequal crossover during meiosis. Both events can insert/delete ("indel") an arbitrary number of bases at once.

The alignment concept The notion of an *alignment* is known in bioinformatics from the sequence alignment problem.

Such an alignment of two sequences has the advantage of working as a measure of relatedness on two levels of abstraction. Firstly, the score of the alignment may be used as a distance measure between two sequences. Such a score is an abstract number representing the distance, and may further be used in other applications, such as a clustering algorithm or a visualization. Secondly, the optimal alignment as a data structure itself reflects the relatedness of the sequences, obtained by evolutionary mutations on the sequence level. This is a major advantage over other distance methods, such as simple basepair distances (see Section 3.1) and even the tree edit distance, as both do not produce an alignment. The alignment itself is a useful construct for further biological analysis, maybe with focus on interesting regions of the alignment, and also for detailed inspection by eye which is conducted by experts.

Another advantage of the alignment model is its versatility. The model is suited to solve a number of related alignment problems, as will be discussed in Section 5. The alignment can be computed either with distance or with similarity score. We may at the same time choose whether to compute either global alignments of the whole input structures, local alignments of substructures, or small-in-large alignments. All of these alignment problems can be computed efficiently with dynamic programming algorithms.

Tree alignment has multiple applications Besides sequences, trees are a widely used data structure in bioinformatics, and, as mentioned above, we may also want to align them to get an alignment and a similarity score. Unordered trees may be used to model phylogenetic trees, and ordered trees may be used to model structured molecules, such as RNA 2D structures, on which we focus.

Our goal of aligning ordered trees is not restricted to the field of bioinformatics. In the same way, we could align a grammar tree used for language translation in the field of syntactic pattern recognition. We might also use our algorithm for image clustering, or chemical structure analysis of other molecules. Or we can use our alignment algorithm as an alternative to the Unix tool

"diff", in order to compare directory trees and structured documents, such as an XML (extensible markup language) file. In general, we can align any data represented as a tree.

Höchsmann et al. [43] developed *RNAforester*, a tool which compares RNA secondary structures based on the tree alignment model. We extend this model, and the corresponding tree alignment algorithm, to work on affine gap costs.

1.3 Overview of this work

In the introduction we discussed the central quest of this work, how to find an adequate measure for RNA secondary structure comparison. In the second chapter, we will introduce basic foundations and definitions from the area of RNA structure comparison. Afterwards, we will review the current state of the art in RNA 2D tree comparison, introducing the most important ideas from the vast number of proposed algorithms for our problem. In the forest alignment chapter, we introduce the original tree/forest alignment distance and separate it from the tree edit distance. We discuss the application of the tree edit distance to different types of trees, such as the well-formed RNA alignment representation. The next chapter follows up the general forest alignment chapter by extending the basic idea to the usage of affine gap costs. The subsequent seventh chapter displays an evaluation of the developed *RNAforester* software program in terms of computational speed, and performance with regards to biological reality, both tested with all different new features of the forest alignment algorithm. In the application chapter, we introduce two direct applications of the new *RNAforester*. Firstly, it may be used in the structure comparison step of the pipeline *planACstar* [16], to get more biologically meaningful results by using the affine gap cost model. Secondly, it may be used in the *RNAforecast* [90] pipeline (combination of *RNAcast* and *RNAforester*), where we also make use of the fact that the input structures share the same shape. The common shape provides anchoring information, which is used to speed up the alignment. The work is concluded by a discussion of the achieved results and the method in general.

2 Foundations and Definitions

As a starting point, we will present the basic notions on sequences, graphs, trees and forests. Then we will define the three structure levels of an RNA molecule as concepts for theoretical analysis. Based on these definitions, we introduce the most common representations for RNA secondary structures. We conclude this chapter with a basic concept of a distance measure on RNA secondary structures, the type of function that is the central subject of this thesis.

2.1 Basic Definitions

Some basic definitions will be given following [85].

2.1.1 Sequences

Definition 2.1.1 (set of strings over $\mathcal{A}$, length of a string, empty string)**.** Let $\mathcal{A}$ be an alphabet, a non-empty finite set of symbols, in our case characters. A *string* S over $\mathcal{A}$ is a finite ordered list of characters from $\mathcal{A}$ written contiguously from left to right. The expression $|S|$ denotes the (non-negative) number of characters in the string S, also called length of string S. The *empty string* is the unique string over $\mathcal{A}$ of length 0, and is denoted ε. The set of all strings over $\mathcal{A}$ of length n is denoted $\mathcal{A}^n$.

Note that $\mathcal{A}^0 = \varepsilon$ for any alphabet $\mathcal{A}$. The set of all strings over $\mathcal{A}$ of any length is the Kleene closure (Kleensche Hülle) of $\mathcal{A}$ and is denoted $\mathcal{A}^*$.

$$\mathcal{A}^* = \bigcup_{n \in \mathbb{N}} \mathcal{A}^n \tag{2.1}$$

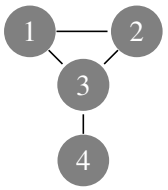
For example, if $\mathcal{A} = \{0, 1\}$, then 0101 is a string over $\mathcal{A}$, and $\mathcal{A}^2 = \{00, 01, 10, 11\}$.

Definition 2.1.2 (i-th character, substring, start position, end position, prefix, suffix, proper)**.** $S[i]$ is the i-th character of string S. For any string S, $S[i..j]$ is the (contiguous) *substring* of S starting at position i and ending at position j in S. We get the position index by counting the characters up to the position, starting at 1. In particular, $S[1..i]$ is the *prefix* of string S ending at potion i, and $S[j..|S|]$ is the *suffix* of string S beginning at position j. A *proper* prefix, suffix or substring of S is a prefix, suffix or substring that is neither the entire string S, nor the empty string.

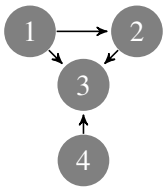
For example, if a string S =“`treealignment`” is given, the substring $S[5..9]$ is the string “`align`”.

2.1.2 Graphs

Definition 2.1.3 (directed graph, vertices, edges, adjacent)**.** A directed graph (or digraph) G is a pair (V, E), where V is a finite set and E is a set of pairs of V, establishing a direction from the first to the second component of an edge via the pair relation. The set V is called the vertex set of G, and its elements are called vertices (singular: vertex). The set E is called the edge



(a) Example: A graph with four nodes.



(b) A directed graph with four nodes.

Figure 2.1: Example directed and undirected graphs

set of G, and its elements are called edges. As the edges are pairs of vertices, they induce a binary relation on the set of vertices, the *adjacency relation*. Vertices connected in such a pair are *adjacent* to each other. Undirected graphs have no directed edges, their edges are subsets of size two rather than pairs.

Definition 2.1.4 (path, cycle/circle, acyclic, connected)**.** A path of length k form a vertex u to a vertex u' in a graph G is a sequence $v_0, v_1, .., v_k$ of vertices such that $u = v_0$,$u' = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, .., k$. The length of the path is the number of edges in the path. In a directed graph, a path $v_o, v_1, .., v_k$ forms a cycle, if $v_0 = v_k$ and the path contains at least one edge. In an undirected graph, a path $v_o, v_1, .., v_k$ forms a cycle, if $v_0 = v_k$ and $v_1, v_2..v_k$ are distinct. A graph with no cycles is acyclic. A graph (V, E) is connected if there is a path $v_i, ..., v_k$ from any vertex $v_i \in V$ to any other vertex $v_j \in V$.

Definition 2.1.5 (labeled graph)**.** Given a graph $G = (V, E)$, a *vertex labeling* is a function from the vertices of a graph to a set of labels L. A graph with such a function defined is called a *vertex-labeled graph*. Likewise, an edge labeling is a function from the edges of the graph to a set of labels, and such a graph is called an *edge-labeled graph*. An *L-labeled graph* for a given set L of labels consists of a graph and labeling mappings from the vertices to labels and from the edges to labels.

Definition 2.1.6 (graph relations)**.** We refer to two equal graphs as isomorphic graphs. Two graphs G and G' are isomorphic if there exists a bijection $f : V- > V'$, such that $(u, v) \in E$ if and only if $(f(u), f(v)) \in E'$. That means we can relabel the vertices of G to the vertices of G', maintaining the corresponding edges in G and G'. We say that graph G' is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$.

2.1.3 Trees

Definition 2.1.7 (tree, rooted tree, unrooted tree, ordered tree)**.** A connected acyclic graph is a *tree*. The vertices of a tree are also known as nodes. If a tree is directed, the direction implies *parent*, *child* and *sibling* node relations and a root node (with 0 parent nodes, all edges point away from the root), and we call the tree a *rooted tree*. Otherwise, we call it an *unrooted tree*. Nodes with one or more child nodes are called *internal nodes*, and nodes with no child nodes are called *leaves*. An *ordered tree* is a tree in which the order of the subtrees is significant.

2.1.4 Forests

Definition 2.1.8 (set of all rooted ordered forests, subtree rooted at i, subforest, closed subforest)**.** A sequence of trees is called an ordered forest, a sequence of rooted trees is a *rooted forest*. The *set of all rooted ordered forests* on an alphabet $\mathcal{A}$ we denote as $\mathcal{F}(\mathcal{A})$.

A *subtree rooted at* i of forest F is a tree with root node i, whose vertices and edges are subsets of the ones of F. A *subforest* of F is a sequence of (sub-) trees of F. A *closed subforest* or CSF is a sequence of consecutive sibling trees in F, in our notation characterized by two indices, a *node index* for the start position i and a *length index* j indicating the length.

Remark 1 (notation for forests). For a node i in a forest, $pre(i)$ is its index in preorder numbering, according to the step when it is visited by preorder traversal. The preorder traversal visits each node in an ordered tree by recursive traversal of subtrees, first visiting the root node of the subtree and then recursively visiting its children in sequence of their ordering.

The *size* of a forest, and the *label(i)*, *rightbrother(i)*, *number of children(i)*, and *rightmost brother(i)* properties of a node at position i in preorder traversal are notable, because the forest will be represented by tables with these values for the computation in *RNAforester*. Besides these, there are two other tables for the index mapping and the anchoring, but the above properties suffice to define a forest.

Also notable are the concepts of child(ren) forest of node i (= the children of node i constitute this forest) and right sibling forest of node i (= the right siblings of node i constitute this forest), which are used to describe the forest alignment distance algorithm later.

2.2 Structure levels of RNA

Three levels of abstraction are used to describe RNA molecules.

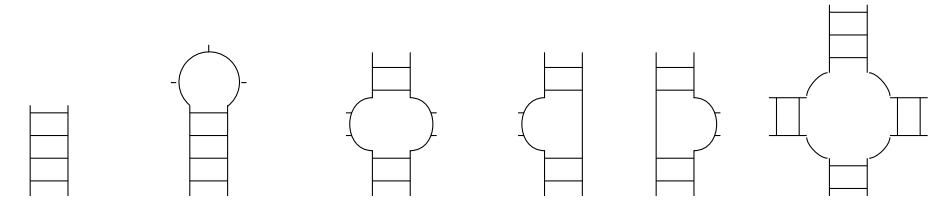
2.2.1 The primary structure

The primary structure is the sequence of bases in the RNA molecule. It is usually represented as a string over the *RNA alphabet*.

Definition 2.2.1 (RNA alphabet)**.** $\mathcal{A} = \{A, C, G, U\}$ is the base alphabet used for modeling RNA sequences, also called the *RNA alphabet*.

The primary structure's main function is the encoding of information, which is used for example in messenger RNA. Messenger RNA (mRNA) transfers information from the DNA to the ribosomes, which then conduct the protein biosynthesis based on the information encoded in the mRNA molecule. Another example for the information encoding function of the RNA are RNA viruses, in which the genetic information of the virus is encoded in the primary structure of a single- or double stranded RNA molecule.

The primary structure determines the tertiary and secondary structure, which results from the sequence of bases that fold into a structural conformation.



(a) Helix (b) Hairpin loop (c) Internal loop (d) Bulge left (e) Bulge right (f) Multiloop

Figure 2.2: Structural elements of RNA.

2.2.2 RNA secondary structure

The secondary structure consists of the sequence of bases, plus information about the bases that pair. Basepairs and stacking work as explained in 1.2.

Taking the basepairing information, the secondary structure defines several structural elements, also called motifs. An RNA strand folding back onto itself forms a *hairpin loop* consisting of a helical region and a loop. At least two unpaired bases, one in each strand opposite to one another, which separate the paired regions to their left and their right, form an *internal loop*. A *bulge* has unpaired bases only in the left or the right strand, whereas in the other strand, all bases are paired. If several helices are enclosed by one or more basepairs, where the helices are linked by unpaired regions of arbitrary length, they form a *multiloop* or multibranch loop. See Figure 2.2 for an illustration.

A first representation of RNA secondary structure Following [85], we define the *secondary structure* S. Given an RNA sequence R of length n, its secondary structure is an undirected graph with vertex set $\{1, .., n\}$, representing positions in R, and an edge set $\{(i, i+1)|1 \leq i \leq n-1\}$, reflecting the order of nucleotides (primary structure) in R. It also consists of another set of edges B_S, describing base pairings, such that if $\{i, j\}, \{k, l\} \in B_S$ with $i < j$ and $k < l$, then

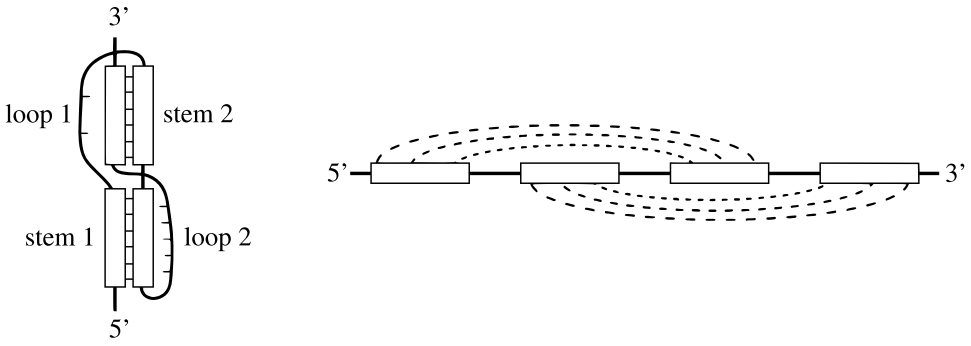
1. $i = k$ iff $j = l$ (one-to-one), and
2. $k \leq j$ implies $i < k < l < j$ (no crossing, no pseudoknots).

A *base pair* in R is modeled by an edge contained in B_S, and may also be written as $i \cdot j$. Vertices not connected by any edge in B_S model *unpaired bases* in R.

A pseudoknot [108] is a structure with crossing basepairs. It is a structure that contains at least two stems (contiguous regions of baspairs), where one of the stems is in between the two halves of the other stem. A typical H form pseudoknot is depicted in Figure 2.3. Forbidding pseudoknots is not necessary for all methods of structure comparison that we discuss in this thesis, but we do it for the sake of simplicity.

The above definition is a graph representation of an RNA secondary structure. An overview of common representations will follow in Section 2.3.

The secondary structure may either be obtained by experimental analysis or be predicted by folding algorithms. A common method for determining the structure in vitro is to partially



(a) H form pseudoknot (b) Straightened H form pseudoknot

Figure 2.3: A typical H form pseudoknot, a simple RNA structural motif with crossing basepairs. The right image depicts a straightened H type pseudoknot to show the linear arrangement of the basepairing regions. Basepairings are indicated with dashed lines in this picture. Figure adapted from [108].

digest the molecule with ribonuclease enzymes under different conditions and isolate the double stranded fragments. The fragments are then subject to sequence analysis.

In silico prediction of the secondary structure can for example be done by folding algorithms using free energy minimizations such as in [145, 81], or by comparative approaches, which we will revisit in Section 11.4.

2.2.3 The tertiary structure

The tertiary structure captures the three dimensional conformation of an RNA molecule. It forms from the secondary structure (hierarchical folding). The three dimensional structure is stabilized by van-der-Waals-Interactions, additional Watson-Crick pairings and/or unusual pairings between hairpin loops, internal loops and bulges.

Obtaining the tertiary structure of an RNA molecule is more complicated, as their prediction is not possible with bioinformatic methods yet, although research is done in this area. Some approaches have been published, but they are limited to small molecules or require experimental data for phylogenetic techniques or computer modeling. The tertiary structure can be determined by experimental methods, but these are complicated, time consuming and expensive [128]. The most common method is based on X-ray crystallography of single crystals of RNA molecules [61]. For this method, the molecules have to be isolated, purified and crystallized first, which is often the biggest obstacle. The second important method is based on nuclear magnetic resonance (NMR) [68].

As the tertiary structure forms based on the secondary structure, which is much easier to obtain both experimental and computationally, most research on RNA structure focuses on the secondary structure.

2.3 RNA secondary structure representations

Before we are able to compare RNA secondary structures, we first have to represent them somehow, so that the comparison method(s) can work on the representation. As the development of comparison algorithms for RNA secondary structures is deeply intertwined with the representations, we will discuss them together in detail in Chapter 3.

A common foundation to almost all secondary structure representations is the RNA alphabet as defined in 2.2.1.

2.3.1 Basepair representation

A very simple representation of an RNA secondary structure is the representation as a sequence accompanied by a list of basepairs (*basepair representation*). Here you see two secondary structures represented in basepair notation, which we will use as examples for structure comparison later.

$$
\begin{array}{rl}
 & \texttt{1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20} \\
S_1 = & \texttt{C A G C U C U C A U G U C C C A C A T A} \\
S_2 = & \texttt{C A G C U C U G U G U C C C A C A C A A} \\
B_{S_1} = & \{\{9,19\},\{10,18\},\{11,17\},\{12,16\}\} \\
B_{S_2} = & \{\{8,18\},\{9,17\},\{10,16\},\{11,15\}\}
\end{array}
$$

2.3.2 Vienna dot-bracket representation

In the *(vienna) dot-bracket representation*, each secondary structure of length n is represented as a sequence of length n that consists of parentheses and dots. Each base pair $\{i, j\}$ in the sequence is represented by an opening and closing bracket at the i-th and j-th position. Each unpaired base is replaced by a dot.

$$
\begin{array}{rl}
 & \texttt{1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20} \\
S_1 = & \texttt{C A G C U C U C A U G U C C C A C A T A} \\
S_2 = & \texttt{C A G C U C U G U G U C C C A C A C A A} \\
Str_{S_1} = & \texttt{. . . . . ((((. . .)))) .} \\
Str_{S_2} = & \texttt{. . . . . ((((. . .)))) . .}
\end{array}
$$

2.3.3 Tree representation

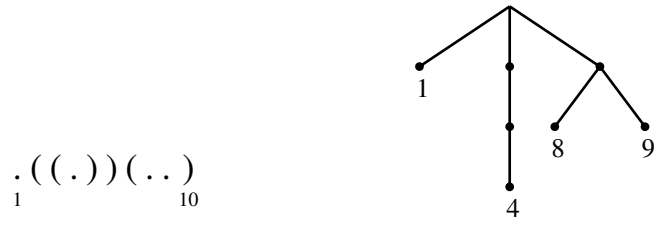
The *tree representation* is in fact a class of various representations that differ in their level of detail.

The secondary structure is represented by an ordered rooted tree. The natural tree representation which corresponds to the dot-bracket notation is shown in Figure 2.4. The root does not correspond to any part of the RNA secondary structure, internal nodes correspond to base pairs, whereas leaves correspond to unpaired bases. A detailed description of this tree representation is given in [30]. This tree representation can also be reduced into a homeomorphic irreducible tree (HIT) representation, in which all sequences of internal nodes which have only one child

are contracted into one node with a weight reflecting the number of nodes that are combined into this one node (see also [30]).

The coarse grained tree representation, which has structural elements as building blocks, can be seen in Figure 2.5. Again, a root node that does not correspond to a structural element, prevents the formation of a forests for structures that are not closed by basepairings. The elements are stems, hairpin loops, internal loops, bulges, multibranch loops (multiloops).

In [102] stems are not included as nodes, but just loops. In [103], the stems may be included and the stem loop size can be used to influence the score function.



(a) Dot-bracket representation (b) Tree representation

Figure 2.4: On the right is a tree representation that corresponds to the dot-bracket representation on the left.

2.3.4 Mountain representation

The *mountain representation* can be derived from the bracket representation. Each base pair is represented by a horizontal line above the primary sequence, drawn at a height that is given by the depth of the nesting brackets representing that base pair.

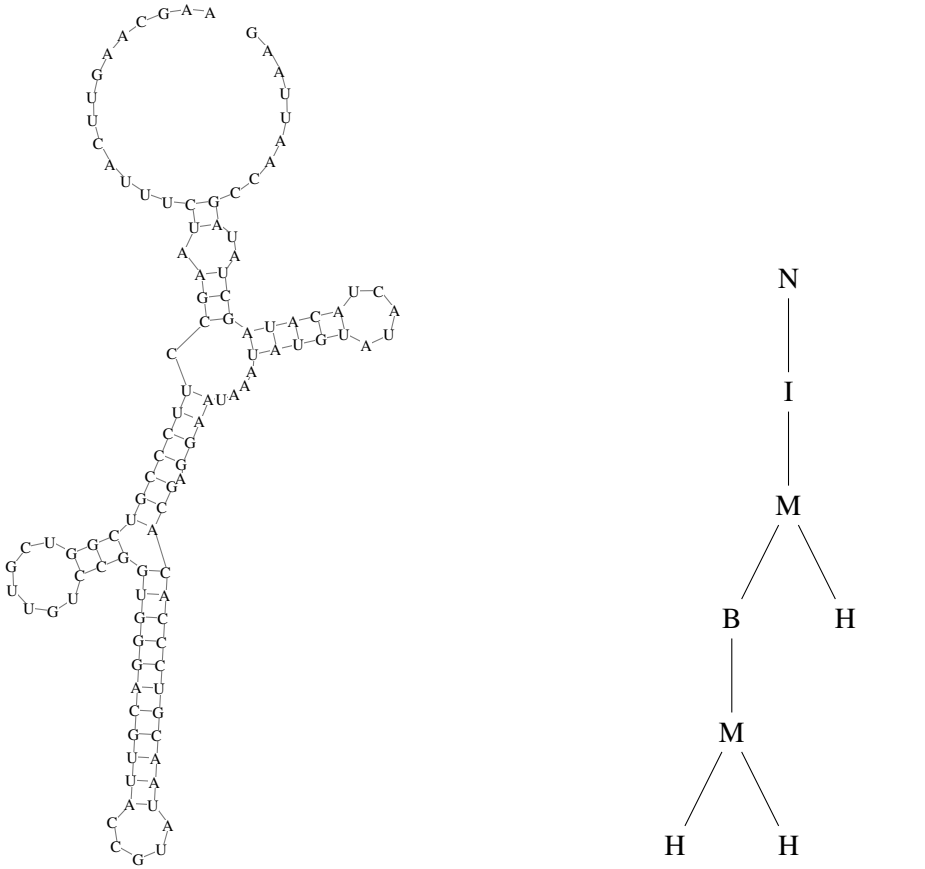
2.3.5 Dome representation

Another common representation for RNA secondary structures is the *dome representation*, in which the primary sequence is just printed as a string or a connected line of dots in one line, and the basepairs are represented as arrows on top of the sequence, which connect the paired bases at their ends (Figure 2.6).

It is closely related to the representation as an *arc annotated sequence* (Figure 2.7), where the sequence is printed as a string, and the basepairs are printed as arcs on top of the sequence, as in the dome representation.

2.3.6 Circle representation

The *circle representation* depicts the RNA sequence in a circle, and shows the base pairings as arcs from position to position in the circle, crossing the central area of the circle. The aforementioned two representations are suitable for depicting RNA molecules with crossing basepairs, such as pseudoknots. If the arcs of a dome representation are drawn on top of a primary sequence, we speak of arc annotated sequences. Similar algorithms for RNA secondary structure



(a) RNA secondary structure

(b) The coarse tree representation

Figure 2.5: Coarse grained tree representation, which represents an RNA secondary structure as a tree of structural building blocks such hairpin loops (H), multiloops (M), bulges (B), internal loops (I). Node N does not represent a structural element, it closes the secondary structure and makes sure the representation forms a tree.

comparison are often expressed in both the terminology of arc annotated sequences, and in the terminology of trees and forests. See Figure 2.8 for a graphical depiction.

2.3.7 Graph representation

The most human readable representation of RNA secondary structures is probably a squiggle plot, in which the molecule's bases are depicted as letters, the basepairings are drawn as lines and the molecule is flattened in the plane.

If we interpret the bases as vertices and the basepairs and backbone as edges, the squiggle plot may be seen as a *representation as a graph*, as illustrated in Figure 2.9. Several representations as graphs exist.

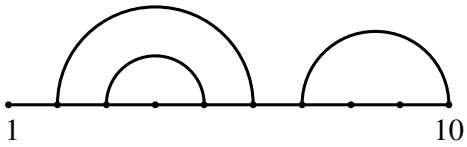


Figure 2.6: Dome representation

Figure 2.7: Arc annotated sequence

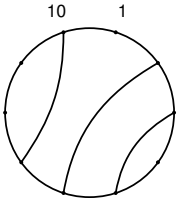


Figure 2.8: Circle representation

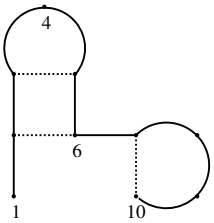


Figure 2.9: Graph representation

2.3.8 Other representations

Other representations, which represent the underlying base sequence in a more implicit way, are also possible. This is the case with hidden Markov models (HMMs, [26]) and stochastic regular grammars (SRGs), and with stochastic context free grammars (SCFGs, [98]) and covariance models (CMs, [28]). Note that SCFGs and CMs are equivalent.

Both HMMs and SCFGs are probabilistic models, more precisely finite models that describe a probability distribution over an infinite number of possible sequences.

A HMM consists of a number of states, which represent for example positions in an RNA sequence. These states are connected by *transition probabilities*. The states also emit symbols, for example from $\{A, C, G, U\}$, according to the *emission probabilities* of the HMM. Like in formal language or automaton theory, the HMM generates a sequence by starting from an initial state, and a series of transitions is made (according to the transition probabilities) until an accepted end state is reached. The visited states in this process are the *sequence of hidden states*. While traversing the states, they emit symbols according to the emission probabilities, and these symbols can be observed. They form the *sequence of observed states*. For the sequence of hidden states, the following state is chosen based on the previous one only, so there is no memory of the states that where visited before, it is memoryless. This property is called the *Markov property*. A good example for a hidden Markov model can be found in [27].

HMMs are used for modeling sequences and sequence alignments. Profile HMMs [66] are common to describe the primary structure consensus of a family of sequences. To represent and analyze RNA secondary structure information, HMMs have to be equipped with additional properties [135], or other full probabilistic approaches have to be considered, such as SCFGs.

A SCFG is a context free grammar with additional probabilities for each production of the grammar, so that a probability distribution exists over the set of productions. The RNA sequence and structure information is encoded in these grammar productions and probabilities. The probabilities of the grammar productions give rise to a probability for each parse tree, which is the product of the probabilities of the productions used to construct the parse tree. So the parse tree is the result of a stochastic process where the nonterminals are replaced independently at random according to the probability distribution of the productions of the grammar. See [98] for examples and further information.

2.3.9 Discussion

We have to distinguish between graphical representations (visualizations) of secondary structures that have been developed for the human eye, and representations that have been primarily designed for computer processing of structural information. Some representations can be used by trained humans and computers, such as the Vienna dot-bracket notation. The mountain representation can also be used for the graphical comparison of secondary structures. The squiggle plot is of course not a good representation for computational methods, as it was designed for the human eye.

As we will focus on trees and forests in the following, let us recall that we have seen multiple

tree representations of RNA secondary structure, with varying level of detail. The tree alignment algorithm which we will discuss, can be used to compare arbitrary trees. It is not even restricted to RNA structure comparison.

2.4 Distance measures for structure comparison

Given a representation of RNA 2D structure, we may define distance measures on RNA structures. To this end, we recall the concept of a metric.

A function which defines the distance of elements in a set, also called a distance function, can be described in mathematics by the concept of a metric.

2.4.1 Metrics

Definition 2.4.1 (metric, pseudo-metric)**.** A function $d : S \times S \to \mathbb{R}$ is called a *metric* on S iff it satisfies for all $x, y \in S$

1. $d(x, y) \geq 0$ (non-negativity),
2. $d(x, y) = 0$ iff $x = y$ (identity of indiscernibles),
   (1 and 2 together: positive definiteness)
3. $d(x, y) = d(y, x)$ (symmetry),
4. $d(x, z) \leq d(x, y) + d(y, z)$ (subadditivity/triangle inequality).

If the second requirement is relaxed to

- $d(x, x) = 0$,

objects need not to be distinguishable, and one may have $d(x, y) = 0$ for distinct values $x \neq y$. A function with these properties is called a *pseudo-metric*.

In other words, if we measure the distance between two objects with a distance function that is a metric, the metric axioms assure that equal objects have distance 0, that the distance function does not take the order of the two objects into account, and that the direct distance between two objects is the shortest.

Example 2.4.1.1 The hamming distance is a metric.
In information theory, the Hamming distance between two strings of equal length is the minimum number of substitutions required to change one string into the other by replacing characters.

For a fixed length n, the Hamming distance is a metric on the vector space of the words of length n.

Let: $x = x_1x_2 \dots x_n \in X$ $y = y_1y_2 \dots y_n \in X$ where X is the vector space of words of length n. Then $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$, where $|x_i - y_i| = \begin{cases} 0 & \text{if} x_i = x_j \\ 1 & \text{otherwise.} \end{cases}$

1. Positivity: Each term in the summation is non-negative, so the sum is also nonnegative, and will be 0 if and only if $x_i = y_i$ for all $i = 1, \ldots, n$. But this means $x = y$, so positivity is proven.

2. Symmetry is also established, as for every i, $|y_i - x_i| = |x_i - y_i|$.

3. For the triangle inequality, let $z = z_1 z_2 \ldots z_n \in X$. Then $d(x, z) = \sum_{l=1}^{n} |x_i - z_i|$. Now for each i, $|x_i - z_i| \leq |x_i - y_t| + |y_t - z_i|$ (the triangle inequality for the absolute value of real numbers). If we sum up both sides for i= 1 ... n, we get $d(x, z) \leq d(y, y) + d(y, z)$. □

In RNA structure comparison, the distance function d operates on the set of secondary structures S, which may be of arbitrary length. However, the length of a secondary structure is the same as the length of the underlying primary structure.

As in other alignment problems [107, 35], we may either calculate the distance between the objects we want to align, or we may calculate their similarity. For an optimal solution, we maximize over a similarity score, or minimize over a distance measure. As these scoring models are interchangeable, we may sometimes casually refer to distances in this work, although actually a similarity score is computed.

In Chapter 3 we will introduce some well-known distance functions. Not all of these fulfill the properties of a metric.

3 Methods for RNA structure comparison

In this chapter we will introduce different models for RNA structure comparison. These models need a representation of the RNA structure to work upon, therefore we introduced the various types of RNA structure representation in Section 2.3. Now we will give a brief overview of the available methods of RNA structure comparison.

The methods will be presented in the order of the underlying representations.

3.1 Basepair distances

A good general introduction to metrics on RNA secondary structures which rely on the basepair representation is given in [85]. The authors discuss different basepair metrics, starting from the symmetric set difference of set theory. Base pair and mountain metrics will be defined based on [85]. For the following explanations, let $S_1, S_2 \in \mathcal{S}_n$, the space of RNA secondary structures of length n (compare Section 2.2.2). The accompanying basepair sets are denoted B_{S_1} and B_{S_2}.

The basepair sets are finite and for some distance measures they also have to be nonempty.

3.1.1 Naive base pair metric

For comparing two secondary structures, we compare their basepair sets. Methods for comparing these sets can be transferred from set theory.

If we compute a distance between two objects, we are interested in their difference rather than their similarity. The method of comparison should account for differences in the sets, i.e. basepairs which the compared sequences do not share. Likewise, it should not report similarities, in this case shared basepairs.

The *symmetric set difference* ("ssd") from set theory is a good first approach to these goals.

Definition 3.1.1. The naive base pair distance d_{ssd} is the cardinality of the symmetric difference between the sets of base pairs B_{S_1} and B_{S_2} of S_1 and S_2,

$$d_{ssd}(S_1, S_2) = |\,(B_{S_1} \backslash B_{S_2}) \cup (B_{S_2} \backslash B_{S_1})\,|.$$

where a base pair is given as a pair of positions in the sequence.

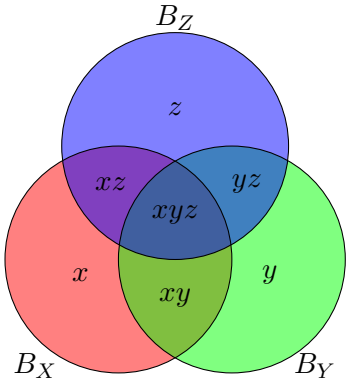
The naive basepair distance d_{ssd} based on the cardinality of the symmetric set difference is a metric.

Proof. Let B_X, B_Y, B_Z be sets of basepairs of RNA secondary structures S_X, S_Y, S_Z.

1. By definition, $d_{ssd}(S_X, S_X) = 0$, and only 0 if the sets are equal.
2. Also, d_{ssd} is symmetric due to the symmetric expression in its definition.
3. For the triangle inequality, we have to show that

   $$d_{ssd}(S_X, S_Y) + d_{ssd}(S_Y, S_Z) \geq d_{ssd}(S_X, S_Z).$$

   We illustrate the proof by a Venn diagram of the union of sets B_X, B_Y and B_Z.



We partition the union of B_X, B_Y and B_Z and write the partitions' cardinalities as follows:

$$\begin{aligned}
x &= |B_X \backslash (B_Y \cup B_Z)| \\
y &= |B_Y \backslash (B_X \cup B_Z)| \\
z &= |B_Z \backslash (B_X \cup B_Y)| \\
xy &= |(B_X \cap B_Y) \backslash (B_X \cap B_Y \cap B_Z)| \\
yz &= |(B_Y \cap B_Z) \backslash (B_X \cap B_Y \cap B_Z)| \\
xz &= |(B_X \cap B_Z) \backslash (B_X \cap B_Y \cap B_Z)| \\
xyz &= |B_X \cap B_Y \cap B_Z|.
\end{aligned}$$

Then the distances d_{ssd} between the three sets B_X, B_Y and B_Z are

$$\begin{aligned}
d_{ssd}(S_X, S_Y) &= x + y + xz + yz \\
d_{ssd}(S_Y, S_Z) &= y + z + xy + xz \\
d_{ssd}(S_X, S_Z) &= x + z + xy + yz.
\end{aligned}$$

Thus,

$$\begin{aligned}
d_{ssd}(S_X, S_Y) + d_{ssd}(S_Y, S_Z) &= 2 \cdot xz + yz + x + 2 \cdot y + xy + z \\
&= d_{ssd}(S_X, S_Z) + 2xz + 2y
\end{aligned}$$

and

$$d_{ssd}(S_X, S_Y) + d_{ssd}(S_Y, S_Z) \geq d_{ssd}(S_X, S_Z).$$

□

Extended variants have been introduced, one of them is based on the Hausdorff distance.

3.1.2 Hausdorff distance

The Hausdorff distance also measures the distance between sets. Named after Felix Hausdorff (1868-1942), it is the "maximum distance of a set to the nearest point in the other set". The Hausdorff distance is a classic maximin function which fulfills metric properties. The general concept of the Hausdorff distance can be applied to RNA secondary structures to compare their basepair sets. The sets of basepairs must not be empty, because the original Hausdorff distance is defined on nonempty sets.

Definition 3.1.2. The distance d_h is defined in three steps. First, the distance between two base pairs $(i,j) \in B_{S_1}$ and $(i',j') \in B_{S_2}$ is defined as

$$d\left((i,j),(i',j')\right) = \max\left(|i-i'|,|j-j'|\right).$$

We also formulate the distance between a basepair and a set:

$$d_a\left((i,j),B_{S_2}\right) = \min_{(i',j')\in B_{S_2}} d\left((i,j),(i',j')\right)$$

Then, the distance d_a between two sets of base pairs is defined as:

$$d_a\left(B_{S_1},B_{S_2}\right) = \max_{(i,j)\in B_{S_1}} \min_{(i',j')\in B_{S_2}} d\left((i,j),(i',j')\right).$$

Symmetrizing gives the final distance d_h:

$$d_h\left(B_{S_1},B_{S_2}\right) = \max\left(d_a(B_{S_1},B_{S_2}),d_a(B_{S_2},B_{S_1})\right).$$

We compute the Hausdorff distance d_h of the following two structures S_1 and S_2 as an example:

$$\begin{aligned} S_1 &= \texttt{.....((((...)))).} \\ S_2 &= \texttt{.....((((...))))..} \\ B_{S_1} &= \{\{9,19\},\{10,18\},\{11,17\},\{12,16\}\} \\ B_{S_2} &= \{\{8,18\},\{9,17\},\{10,16\},\{11,15\}\} \end{aligned}$$

We number the basepairs as they appear as bp_1 to bp_4 in B_{S_1} and bp_5 to bp_8 in B_{S_2}. First, we collect the absolute values between the left and right positions of the pairs in a table.

$d(bp_x,bp_y)$	bp_5	bp_6	bp_7	bp_8
bp_1	$\max(1,1)$	$\max(0,2)$	$\max(1,3)$	$\max(2,4)$
bp_2	$\max(2,0)$	$\max(1,1)$	$\max(0,2)$	$\max(1,3)$
bp_3	$\max(3,1)$	$\max(2,0)$	$\max(1,1)$	$\max(0,2)$
bp_4	$\max(4,2)$	$\max(3,1)$	$\max(2,0)$	$\max(1,1)$

For the computed maxima, we get:

$d(bp_x, bp_y)$	bp_5	bp_6	bp_7	bp_8
bp_1	1	2	3	4
bp_2	2	1	2	3
bp_3	3	2	1	2
bp_4	4	3	2	1

We then compute the column minima, of which we take the maximum to get $d_a(B_{S_1}, B_{S_2})$, the asymmetric distance between B_{S_1} and B_{S_2}. We also compute the maximum of the row minima to get $d_a(B_{S_2}, B_{S_1})$.

	bp_5	bp_6	bp_7	bp_8	row min
bp_1	1	2	3	4	1
bp_2	2	1	2	3	1
bp_3	3	2	1	2	1
bp_4	4	3	2	1	1
col min	1	1	1	1	

$$\begin{aligned} d_a(B_{S_1}, B_{S_2}) &= 1 \\ d_a(B_{S_2}, B_{S_1}) &= 1 \end{aligned}$$

The maximum of these two maxima is the symmetric distance d_h.

$$d_h(S_1, S_2) = 1$$

The distance d_h is a metric.

Proof. Let B_X, B_Y, B_Z be sets of basepairs of RNA secondary structures S_X, S_Y, S_Z.

1. From the definition,

$$\begin{aligned} d_h(S_X, S_X) &= max\{d_a(B_X, B_X), d_a(B_X, B_X)\} = d_a(B_X, B_X) \\ &= \max_{(i,j)\in B_X} d_a\left((i,j), B_X\right) = 0. \end{aligned}$$

   $d_h(S_X, S_Y) = d_a(x, y)$ for some $x \in B_X$ and $y \in B_Y$ (as sets of basepairs are compact). Hence, $0 \leq d_h(S_X, S_Y) < \infty$. If $B_X \neq B_Y$ we can assume that there is an $x \in B_X$ such that $x \notin B_Y$. Then $d_h(S_X, S_Y) \geq d_a(x, B_Y) > 0$.

2. Symmetry is established, as d_h explicitly symmetrizes the asymmetric distance d_a by the definition of $d_h(S_X, S_Y) = \max\left(d_a(B_X, B_Y), d_a(B_Y, B_X)\right) = d_h(S_Y, S_X)$.

3. To show that $d_h(S_X, S_Y) \leq d_h(S_X, S_Z) + d_h(S_Z, S_Y)$, we first show that $d_a(B_X, B_Y) \leq d_a(B_X, B_Z) + d_a(B_Z, B_Y)$ (for the asymmetric distance function).

   For any $x \in B_X$, we have:

$$\begin{aligned}
d_a(x, B_Y) &= \min_{y \in B_Y} (d(x, y)) \\
&\leq \min_{y \in B_Y} (d(x, z) + d(z, y)) \forall z \in B_Z, \text{ as } d \text{ is clearly a metric, and} \\
&= d(x, z) + \min_{y \in B_Y} (d(z, y)) \forall z \in B_Z, \text{ so} \\
d_a(x, B_Y) &\leq \min_{z \in B_Z} (d(x, z)) + \max_{z \in B_Z} \min_{y \in B_Y} d(z, y) \\
&= d_a(x, B_Z) + d_a(B_Z, B_Y), \text{ and because this holds for any } x \text{ we get:} \\
d_a(B_X, B_Y) &\leq d_a(B_X, B_Z) + d_a(B_Z, B_Y).
\end{aligned}$$

   Similarly, we can prove that

$$d_a(B_Y, B_X) \leq d_a(B_Y, B_Z) + d_a(B_Z, B_X).$$

   And we get that

$$\begin{aligned}
d_h(S_X, S_Y) &= \max(d_a(B_X, B_Y), d_a(B_Y, B_X)) \\
&\leq \max(d_a(B_Y, B_Z), d_a(B_Z, B_Y)) + \max(d_a(B_X, B_Z), d_a(B_Z, B_X)) \\
&= d_h(S_Y, S_Z) + d_h(S_X, S_Z).
\end{aligned}$$

This proof is a special case of [11]. □

3.1.3 Prohorov metric

Inspired by the (Lévy-)Prohorov (also: Prokhorov) distance in probability theory, a distance d_p has been defined for basepair sets. Unfortunately, no exact definition of d_p is given in the literature that refers to it [85, 147]. An informal definition is however given in [85] as:

Definition 3.1.3. If $S_1, S_2 \in \mathcal{S}_n - \mathcal{S}_0^n$ [=$\mathcal{S}_n$ without the open structure], then $\mathcal{S}_1$ is said to be within distance d of S_2, if for every base pair $b_1 \in B_{S_2}$, with the possible exception of d base pairs, there is a base pair $b_2 \in B_{S_2}$, such that the distance between b_1 and b_2 is at most d. Symmetrizing this measure, we define $d_P(S_1, S_2)$ to be the maximum of the two distances between S_1 and S_2.

Let us compute the distance $d_P(S_1, S_2)$ of the following structures as an example:

$$\begin{aligned}
S_1 &= \texttt{...(..)............((((...))))} \\
&\quad bp_0 = \{5, 8\}, \quad bp_1 = \{20, 30\}, \quad bp_2 = \{21, 29\}, \quad bp_3 = \{22, 28\}, \quad bp_4 = \{23, 27\} \\
S_2 &= \texttt{..................((((...))))} \\
&\quad bp_1 = \{20, 30\}, \quad bp_2 = \{21, 29\}, \quad bp_3 = \{22, 28\}, \quad bp_4 = \{23, 27\}
\end{aligned}$$

First, we compute $\max(|i-i'|, |j-j'|)$ for all combinations of basepairs (i, j) and (i', j') like for the computation of d_h. This is illustrated in the auxiliary table below:

S_1/S_2	bp_1	bp_2	bp_3	bp_4
bp_0	$\max(15, 22)$	$\max(16, 21)$	$\max(17, 20)$	$\max(18, 19)$
bp_1	$\max(0, 0)$	$\max(1, 1)$	$\max(2, 2)$	$\max(3, 3)$
bp_2	$\max(1, 1)$	$\max(0, 0)$	$\max(1, 1)$	$\max(2, 2)$
bp_3	$\max(2, 2)$	$\max(1, 1)$	$\max(0, 0)$	$\max(1, 1)$
bp_4	$\max(3, 3)$	$\max(2, 2)$	$\max(1, 1)$	$\max(0, 0)$

After taking the maxima, we take the row and column minima as known from d_h.

Then we start crossing out outliers which we ignore according to the above definition of d_P. To get an overall distance d_P of d, we may ignore up to d entries from the row minima, and up to d entries from the column minima. The remaining minima must all be below d.

S_1/S_2	bp_1	bp_2	bp_3	bp_4	row min
bp_0	22	21	20	19	~~19~~
bp_1	0	1	2	3	0
bp_2	1	0	1	2	0
bp_3	2	1	0	1	0
bp_4	3	2	1	0	0
col. min	0	0	0	0	

If we ignore the row minimum entry of distance 19, all other row minima are below one. As all column minima are 0, symmetrizing the measure by maximization results in an overall distance of one. Ignoring more entries is not necessary and only increases the distance.

$d_P(S_1, S_2) = 1$.

The distance function d_p is not a metric, as it does not fulfill the triangle inequality [85]. (Counter-)Example:

$$
\begin{aligned}
S_1 &= \texttt{((((..(((((.....)))))..))))} \\
&\quad bp_1 = \{1, 28\}, \quad bp_2 = \{2, 27\}, \quad bp_3 = \{3, 26\}, \quad bp_4 = \{4, 25\}, \\
&\quad bp_5 = \{7, 22\}, \quad bp_6 = \{8, 21\}, \quad bp_7 = \{9, 20\}, \quad bp_8 = \{10, 19\}, \\
&\quad bp_9 = \{11, 18\} \\
S_2 &= \texttt{.......((((.....)))).((...))} \\
&\quad bp_6, \quad bp_7, \quad bp_8, \quad bp_9, \quad bp_{10} = \{23, 29\}, \quad bp_{11} = \{24, 28\} \\
S_3 &= \texttt{..................(((.....)))} \\
&\quad bp_{12} = \{19, 29\}, \quad bp_{13} = \{20, 28\}, \quad bp_{14} = \{12, 27\}
\end{aligned}
$$

S_1/S_2	bp_6	bp_7	bp_8	bp_9	bp_{10}	bp_{11}	row min
bp_1	7	8	9	10	22	23	~~7~~
bp_2	6	7	8	9	21	22	~~6~~
bp_3	5	6	7	8	20	21	~~5~~
bp_4	4	5	6	7	19	20	~~4~~
bp_5	1	2	3	4	16	17	1
bp_6	0	1	2	3	15	16	0
bp_7	1	0	1	2	14	15	0
bp_8	2	1	0	1	13	14	0
bp_9	3	2	1	0	11	13	0
col. min	0	0	0	0	~~11~~	~~13~~	

$d_P(S_1, S_2) = 4$

S_2/S_3	bp_{12}	bp_{13}	bp_{14}	row min
bp_6	11	12	13	~~11~~
bp_7	10	11	12	~~10~~
bp_8	10	10	11	~~10~~
bp_9	11	10	10	~~10~~
bp_{10}	4	3	2	2
bp_{11}	5	4	3	3
col min	4	3	2	

$d_P(S_2, S_3) = 4$

S_1/S_3	bp_12	bp_13	bp_14	row min
bp_1	18	19	20	~~18~~
bp_2	17	18	19	~~17~~
bp_3	16	17	18	~~16~~
bp_4	15	16	17	~~15~~
bp_5	12	13	14	~~12~~
bp_6	11	12	13	~~11~~
bp_7	10	11	12	~~10~~
bp_8	10	10	11	~~10~~
bp_9	11	10	10	~~10~~
col min	~~10~~	~~10~~	~~10~~	

$d_P(S_1, S_3) = 9$

The triangle inequality is not satisfied, because the direct distance from S_1 to S_3 $d_P(S_1, S_3) = 9$ is longer than going via S_2, which results in $d_P(S_1, S_2) = 4$ plus $d_P(S_2, S_3) = 4$.

For a proof that the general Prohorov distance is a metric, see Huber [49].

3.1.4 Mountain metric

Moulton et al. present the mountain metric [85]. The *mountain metric* d_m is based on the mountain representation [48]. It has also been used as a means of comparison, in a graphical way.

Definition 3.1.4. For each structure, a vector $f_S \in (\mathbb{N}_0)^N$ is defined with elements $f_S(i)$ being the difference between the number of opening and closing brackets from 1-st to i-th position. Vector f_S is called the *mountain function*.

The l_p-norm (for $p = 2$: root mean square distance, Euclidean norm) on these vectors is a metric

$$\begin{aligned} d_m(S_1, S_2) &:= ||f_{S_1} - f_{S_2}||_p \\ &:= (\sum_{i=1}^{n} |f_{S_1}(i) - f_{S_2}(i)|^p)^{1/p} \end{aligned}$$

for all $S_1, S_2 \in \mathcal{S}_n$ where $n = \max\{|S_1|, |S_2|\}$.

The metric has a scaled variant $w_S(k) = \begin{cases} 1/(l-k), & \text{if } (k,l) \in B_S \\ -1/(k-l), & \text{if } (l,k) \in B_S \\ 0, & \text{otherwise.} \end{cases}$

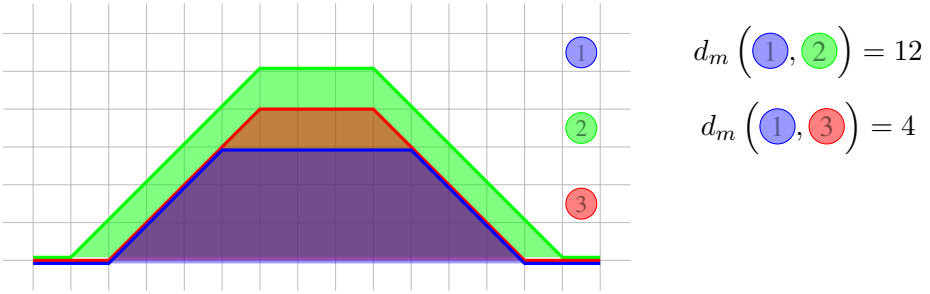


Figure 3.1: An example explaining the mountain metric. The structures S_1=..(((.....)))..., S_2=.((((.....)))). and S_3=..((((...))))... are shown in mountain representation. Their mountain vectors are $f_{S_1} = (0,0,1,2,3,3,3,3,3,3,2,1,0,0,0)$, $f_{S_2} = (0,1,2,3,4,4,4,4,4,4,3,2,1,0,0)$ and $f_{S_3} = (0,0,1,2,3,4,4,4,4,3,2,1,0,0,0)$. Applying the mountain metric with $p = 1$ is the same as comparing the area under the mountain functions and yields $d_m(S_1, S_2) = |24 - 36| = 12$ and $d_m(S_1, S_3) = |24 - 28| = 4$.

As the mountain metric employs a common l_p norm on the mountain vectors, it clearly fulfills the metric axioms.

3.1.5 Discussion of the basepair metrics

For the Hausdorff and Prohorov based metrics, the basepair sets have to be nonempty. This requirement means that the measures do not work on the open unpaired structure, which would be represented as a string of dots in the Vienna dot-bracket notation. This is not a big problem, as the affected metric is used as a filter for suboptimal structures in mfold, and gets folded sequences as input [147].

Naive basepair metric The naive basepair metric captures the difference of the basepair sets, while omitting common elements. This metric is very coarse, and it does not capture any secondary structure information. Different length and shifted substructures cause problems. These two secondary structures both have the same overall shape of an hairpin. On the sequence level, mutations of just two bases are needed to shift the hairpin to the left by one base.

$$\begin{aligned} S_1 &= \texttt{.....((((...)))).} \\ S_2 &= \texttt{.....((((...))))..} \\ B_{S_1} &= \{\{9,19\},\{10,18\},\{11,17\},\{12,16\}\} \\ B_{S_2} &= \{\{8,18\},\{9,17\},\{10,16\},\{11,15\}\} \end{aligned}$$

The shift leads to a distance $d_{ssd}(S_1, S_2) = 8$, but it should be close to zero for such a small change.

Hausdorff based metric Inspired by the *Hausdorff metric*, the distance d_h on RNA secondary structures was developed by Zuker et al. [146]. They defined the distance d_h between secondary structures to filter out suboptimal structures in the *mfold* program ([145], [146]) that folds RNA sequences into secondary structures. The distance d_h was in this way used to reduce the output of alternate structures with trivially different foldings in the early program versions. In the original definition, S_1 and S_2 are two secondary structures on the same sequence. A problem with d_h is that it cannot handle open structures without basepairings, because the sets of basepairs must not be empty.

Let us revisit the previous example, now with d_h as measure of comparison.We will see that with d_h, shifted substructures are represented in an adequate way and do not cause a big distance.

According to the example computation that was done in 3.1.2,

$$d_h(S_1, S_2) = 1,$$

which is more reasonable than the distance computed by d_{ssd}.

The metric d_h can handle shifted base pairs, but a single new pairing results in considerable difference.

Isolated base pairs can lead to high distance values depending on the distance to the next base pair, as the distance d_h works by seeking the maximum distance between any two base pairs in the sets.

As we have seen, shifted base pairs are reflected in an adequate way. But a single new pairing may cause difficulties, because it may give rise to a big difference. We can see this in another example with metric d_h.

$$\begin{aligned} S_3 &= ((((...))))................... \\ S_4 &= ((((...))))..........(...)... \end{aligned}$$

Again, we number the basepairs as they appear:

$$bp_1 = \{1, 11\}, \quad bp_2 = \{2, 10\}, \quad bp_3 = \{3, 9\}, \quad bp_4 = \{4, 8\}, \quad bp_5 = \{23, 27\}$$

.

We then collect the absolute values between the left and right positions of the pairs in a table again:

	bp_1	bp_2	bp_3	bp_4	bp_5
bp_1	$\max(0,0)$	$\max(1,1)$	$\max(2,2)$	$\max(3,3)$	$\max(22,16)$
bp_2	$\max(1,1)$	$\max(0,0)$	$\max(1,1)$	$\max(2,2)$	$\max(21,17)$
bp_3	$\max(2,2)$	$\max(1,1)$	$\max(0,0)$	$\max(1,1)$	$\max(20,18)$
bp_4	$\max(3,3)$	$\max(2,2)$	$\max(1,1)$	$\max(0,0)$	$\max(19,19)$

The maxima of both positions in each table cell are computed, and also the table row and column minima are computed.

	bp_1	bp_2	bp_3	bp_4	bp_5	row min
bp_1	0	1	2	3	22	0
bp_2	1	0	1	2	21	0
bp_3	2	2	0	1	20	0
bp_4	3	2	1	0	19	0
col min	0	0	0	0	19	

$$\begin{aligned} d_a(B_{S_1}, B_{S_2}) &= 19 \quad \text{(max of col minima)} \\ d_a(B_{S_2}, B_{S_1}) &= 0 \quad \text{(max of row minima)} \\ \rightarrow d_h(S_3, S_4) &= 19 \end{aligned}$$

Prohorov based metric As seen in the last example, outliers in the sets of basepairs that should be compared may cause problems with the Hausdorff distance. Two structures can have a big distance, while differing in just one or two base pairs.

To circumvent this problem, a new distance d_p is used in newer versions of mfold. This distance function is a variant of d_h that ignores up to d bases to obtain distance d. If two structures are more than a distance d apart, one of them must contain at least $d + 1$ base pairs that are not within distance d of any base pair in the other structure.

Mountain metric The mountain metric can handle shifted base pairs, single pair differences, but stem loop extension causes problems, because the position in the stem influences the resulting distance.

The mountain metric can be computed in $\mathcal{O}(n)$ time, and is handy in theoretical analyses. It is also easy to understand because it can always be visualized via the mountain representation.

With the usual mountain metric, single basepairs score differently depending on the number of bases they enclose.

$$\begin{aligned} S_1 &= \texttt{..(((.....)))..} \\ S_2 &= \texttt{.((((.....))))\,.} \\ S_3 &= \texttt{..((((...))))..} \end{aligned}$$

$$\begin{aligned} f_{S_1} &= (0,0,1,2,3,3,3,3,3,3,2,1,0,0,0) \\ f_{S_2} &= (0,1,2,3,4,4,4,4,4,4,3,2,1,0,0) \\ f_{S_3} &= (0,0,1,2,3,4,4,4,4,3,2,1,0,0,0) \end{aligned}$$

Both S_2 and S_3 are one base pair apart from S_1, but d_h to S_1 differs ($p = 1$):

$$\begin{aligned} d_m(S_1, S_2) &= 12 \\ d_m(S_1, S_3) &= 4 \end{aligned}$$

The scaled variant of the mountain metric was developed to avoid this effect, by scaling with the distance between left and right component of the basepair.

3.2 Encoding as a string and alignment

Several approaches to RNA secondary structure comparison use the idea of encoding structure information in a string, to be able to use established sequence alignment algorithms for comparison. Konings and Hogeweg [65] as well as Shapiro [102] developed the idea at the same time, starting from different initial representations. Other approaches followed [62], and are still being developed.

3.2.1 Konings and Hogeweg - string encoding from mountain representation

Konings and Hogeweg proposed a representation for RNA structures that derives a string encoding from the mountain representation [65]. The method to produce the encoding starts with a set of set of major minimal foldings of the molecule. These folded molecules are then transformed to the mountain representation. The next step is the translation to secondary structure strings. In these strings, > stands for ascension in mountain representation, < for descent, + for stay on same level, ^ for the mountain top level. In terms of bases and basepairs, an ascension corresponds to a left base of a basepair, a descent to a right base of a base pair, staying on the same level represents a general single stranded region, and the mountain top level corresponds to the single stranded loop region in a hairpin loop. In the comparison step, an all against all comparison is made by sequence alignment of the linear representations from the previous step. Comparison of these sequences can for example be done with a common dynamic programming algorithm (i.e. the Needleman-Wunsch algorithm [87]).

3.2.2 Shapiros encoding - string encoding from coarse grained tree representation

The string encoding idea of Shapiro is similar, but starts from a coarse grained tree representation [102]. Again, the first step is to fold the molecule. Then, the molecule is represented as a tree, with the following building blocks for RNA secondary structure components:

- M for multiloop
- B for bulge
- I for internal loop
- H for hairpin loop

and a root node N that does not correspond to a structural element, but ensures that the graph is connected. The tree of building blocks is then flattened to a string with brackets:
`(N(I(M(B(I(B(I(H)))))(M(I(I(H))))(H)(I(I(H))))))`
This representation can be simplified without loss of information, if linear stems are written in one sequence without brackets:
`(N I (M B I B I H) (M (I I H) (H) (I I H)))`
The string encoding is compared by means of common sequence alignment.

3.2.3 Tree representation distance - from tree to string by counting children

Another coarse approach of tree comparison by encoding as a string was proposed by Kitagawa et al. in [62].

The tree representation distance reduces the whole problem of tree comparison to sequence comparison, and the tree is mapped to a linear string representation. This translation is done via a depth-first traversal of the tree, during which each traversed vertex is represented by the

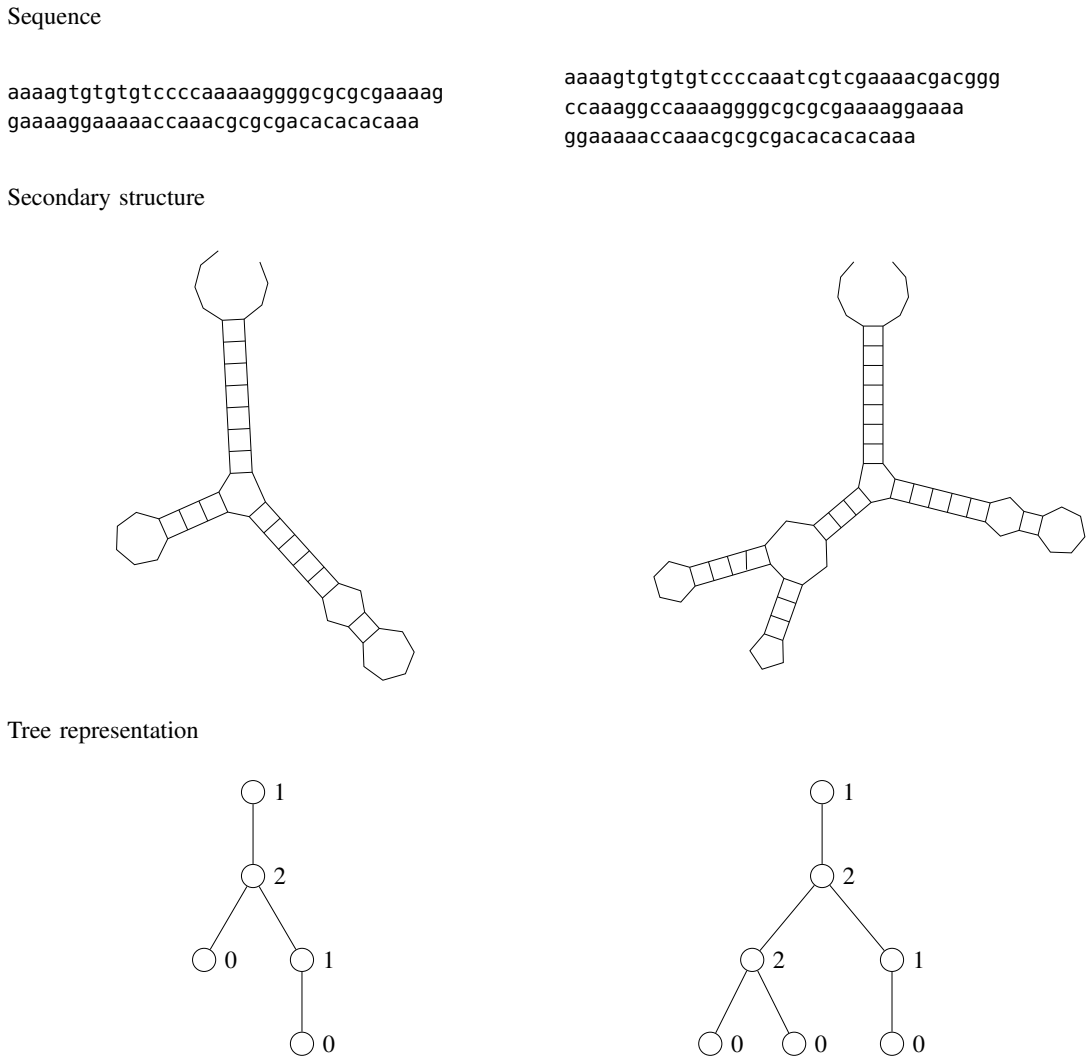


Figure 3.2: Tree representation distance, figure modified from [62]. The four steps from top to bottom include (1) the sequences, (2) the secondary structures, (3) the tree representations, and (4) the sequences of integers, aligned.

number of its children. The resulting strings of numbers are aligned by sequence comparison. See Figure 3.2 for an example of the tree representation distance.

3.2.4 Discussion of string encoding methods

A problem common to all methods of comparison that do not treat a pair of brackets as a unit, such as all sequence alignment methods, is the possibility to compute alignments that are based on wrongly formed basepairs:

```
(((...(((...)))...)))
(((.....---...)))....---
```

One possible remedy for this is combining sequence based algorithms with structural information that explicitly represents basepairs as units, which we will discuss in the following section.

3.3 Sequences with structure information, arc annotated sequences

As discussed in Section 3.2.4, sequence based approaches may compute alignments which contradict the correct nesting of basepairs.

The widely used tree edit distance which we will discuss in the next section, does not have this problem. It is limited to compare RNA secondary structures without pseudoknots, because of the underlying model. With the coarse grained representation which is usually used with the tree edit distance, edit operations are hard to motivate biologically. On the other hand, the natural tree representation, which models bases directly, cannot account for basepair bond deletions.

These drawbacks of the tree edit model caused an interest in sequence alignment algorithms with structural constraints.

One of the first algorithms that adds structural information to a sequence alignment algorithm is the Sankoff algorithm, which does however not only align structures, but align and fold the input sequences simultaneously [99].

RNAlign: alignment of RNA sequences using both primary and secondary structures An early example of an algorithm that uses using both primary and secondary structure information to align an unfolded RNA sequence to a folded one was introduced by Corpet and Michot in [23]. An anchoring heuristic is used to divide the alignment problem to small subproblems, on which a dynamic programming algorithm is used. As an application, aligning a "bank" of structures with a new structure is presented. The input consists of sequence information and structure information, where structure information encodes an unpaired region as space or dot, and a paired position as `-`, except first and last position of every stem strand. This first and last position of stem strand is encoded as > on 5' stem strand, < on 3' stem strand.

Bafna et al. [10] refined this approach and made it more applicable by reducing the runtime of the dynamic programming algorithm to $\mathcal{O}(n^4)$. They coined the term "RNA strings" for sequences annotated with structure information. An RNA string is an RNA sequence as a string

plus non crossing edges between A-U and G-C. Their algorithms use both sequential and structural information of the RNA strings. They compute corresponding versions of longest common subsequence and the shortest common supersequence from strings, which turn out to be computationally quite different on RNA strings. Like Corpet and Michot, they use their algorithm to align an unfolded sequence to a known folded one and in this way solve the folding problem for sequences with related structures whose folding is known.

A completely different approach to the problem of aligning an unfolded sequence to a folded one with a known structure, like in [23], is introduced by Lenhof et al. in [69]. They formulate the alignment problem as an integer linear programming problem. Then they use polyhedral combinatorics to solve the problem.

3.3.1 Evans/LAPCS problem

Evans works on sequences annotated with structure information in her PhD thesis [29]. She studies various methods to incorporate structural information into the primary sequence, such as symbol coloring, substring coloring, and arcs. The phrase "arc annotated sequences" she uses becomes common for these types of representations of RNA secondary structure.

General arc annotated sequences in Evans' model have arcs as a connection between two characters (binary relation on them) and allow one-to-many character connections by arcs. Based on imposed restrictions, the arc annotated sequences are then divided into classes: 1) crossing: one-to-one relation, crossing arcs are allowed 2) nested: one-to-one relation, no crossing arcs are allowed. The first class represents the class of tertiary structures, and the second one the secondary structures. The longest common subsequence problem from sequence analysis is extended to the longest arc preserving common subsequence problem on arc annotated sequences. This problem is shown in the thesis to be NP-hard even for secondary structures.

3.3.2 Zhang's model

Zhang et al. propose an edit model with tertiary interactions [144]. Their model basically corresponds to the tree edit algorithm and the natural representation of RNA secondary structures on base and basepair level. In their model, bases correspond to at most one pair and for secondary structures, the arcs are not crossing. Their similarity measure takes primary, secondary, and tertiary structure into account. The computation is NP complete, as the authors show by reducing the 3-sat problem to their problem. Although they agree that RNA secondary structures naturally have a tree like structure, they criticize most tree approaches as too abstract (those on loops and stems). That is why they propose a representation on bases and basepairs. Basepairs are treated as a unit in Zhang et al.s algorithm.

3.3.3 Jiang's model - A general edit distance between RNA structures

A first effort to generalize the different types of edit distances based on the arc annotated sequence model has been performed by Jiang et al. [56]. It is based on a superset of edit operations of the former model, here defined on bases and arcs. Several algorithms are presented to compute the edit distance for different arc structures which can be plain, nested or crossing. They also define a new edit distance between two RNA secondary or tertiary structures. The

authors prove that the problem is MAX-SNP-hard for arbitrary scoring schemes. A problem of the algorithms is that non-natural alignments are possible. This has to be solved via the scoring scheme.

By arc breaking and arc altering, a left base of a basepair could be aligned with a right base of some basepair, or a basepair in one RNA could cross a basepair in the other RNA. By setting the cost of arc operations, one could possibly avoid such wrong alignments.

3.3.4 Alignment hierarchy / Alignments of RNA structures

In [15], Blin et al. give an unifying theoretical framework for RNA structure comparison, called alignment hierarchy. It is based on the definition of common supersequences for arc-annotated sequences and tries to comprise the main existing alignment models for RNA structure comparison based on trees and arc annotated sequences. The alignment hierarchy is studied and a new polynomial time algorithm is presented, which involves biologically relevant evolutionary operations, such as pairing or unpairing nucleotides. The tree edit distance is embedded as a subcase in the alignment hierarchy. The algorithm is implemented in the software *gardenia*.

3.3.5 Discussion of Arc annotated sequence methods

Different RNA structure comparison problems can and have been formulated in terms of arc annotated sequence. The tree edit and tree alignment problems which we will discuss below, have been treated on arc annotated sequences as well as on trees. The community is mostly biased towards either trees and forests or arc annotated sequences, which has lead to confusion, as the same problems are solved in different guises.

Recent contributions on arc annotated sequences present alignment algorithms that include pseudo-knotted structures [84, 9], which cannot be represented as trees. Neither of these approaches considers affine gaps or anchorings.

3.4 SCFGS/HMMs

Stochastic context free grammars are able to capture the common primary and secondary structure information of RNA. In that way, they are useful for modeling, folding and aligning RNA secondary structures. They generalize the hidden Markov models (HMMs) which are often used for DNA sequence analysis.

Stochastic grammars assign probabilities to the grammar rules, and thus an overall probability for each generated sequence can be determined. Stochastic regular grammars are equivalent to HMMs, whereas stochastic context free grammars are a generalization. They are equivalent to the covariance models of Eddy and Durbin [28], but usually different algorithms are used to train them.

Of course, HMMs could be used to model families of RNA sequences, too. A problem with that is that HMMs treat all positions as if they have independent, non-interacting distributions, which is not suitable for modeling RNA, because basepaired positions are in general highly correlated.

They show a covariation in sequence over time, which explains why Eddy and Durbin's SCFGs with the special architecture to model RNA families are called covariance models.

3.4.1 Stochastic context-free grammars for tRNA modeling

An application of stochastic context-free grammars to the problems of modeling, folding and aligning RNA structures can be found in [98] by Sakakibara et al. In the model, both primary sequence and secondary structure of the tRNA are incorporated. The model is successfully used to differentiate general tRNA from other RNA sequences of similar length. After training the model only on 20 tRNAs from two subfamilies, it is able to distinguish tRNA from other RNA of similar length for other species, to fold new tRNA sequences, and to produce multiple alignments of tRNAs. Based on these results, Sakakibara et al. also propose improvements for alignments of the D- and T-domains of mitrochondrial tRNAs.

3.4.2 Small subunit ribosomal RNA modeling using stochastic context-free grammars

Another approach using SCFGs is described by Brown in [17]. Similar to [98], a model based on primary sequence and secondary structure is used to construct small subunit ribosomal RNA (SSU rRNA) multiple alignments. Brown shows that his method produces high quality alignments. They also introduce some constraints for SCFGs to reduce computation time, which is needed to use the method on large problems such as SSU rRNA. The reconstruction of phylogenetic trees is an intended application for the method.

3.5 Tree and forest comparison

Another remedy for the problem with sequence alignments for RNA secondary structure comparison, besides using arc annotated sequences, is to turn to trees and forests as means of representation. They naturally model nesting and adjacency, which are central relationships of RNA substructures. Two main approaches are most common, the tree edit distance and the tree alignment distance, but there are also some closely related problems. We start with introducing the tree edit distance, which has first been published in 1979 by Tai.

3.5.1 Tree edit distances

The tree edit distance problem is the problem of finding a series of edit operations that transforms one input tree into the other with minimum overall cost, where the overall cost results from the sum of costs for the basic edit operations. Overviews on the tree edit distance are given in the textbooks [119], [8] and the review article [14].

Different basic algorithmic techniques, such as divide and conquer, branch and bound, and of course dynamic programming and backtracking have been used for tree comparison.

Similarity between trees may be defined in different ways analogous to the similarity between sequences. One can look for the largest maximum agreement subtree [58, 7], the largest com-

mon subgraph, the minimum tree edit distance [141] or the smallest common supertree/the tree alignment distance [55]. Different basic edit operations may be allowed or not allowed.

The algorithms can be extended to more complex tree models, such as quotiented trees [88]. A quotiented tree is an ordered tree with an equivalence relation on its vertices, such that, when the equivalence classes are collapsed to super-nodes, the resulting graph is also an ordered tree.

We start by discussing the three main variants of the tree edit distance.

The Tree-to-Tree Editing Problem: First variant by Selkow, indels at leaves only

Determining the minimum cost sequence of edit operations for transforming a tree into another one by basic edit operations was first brought up in [101]. The basic edit operations insertion, deletion and replacement may be applied to a node in the tree, where in this case insertions and deletions are restricted to the leaves of the tree. The proposed algorithm is a top down dynamic programming approach. Due to the restriction that only the leaves may be edited, this variant is not so widely used. Mainly it is used for didactic purposes [100, 119], another application is the identification of syntactic differences between programs [134].

The Tree-to-Tree Correction Problem: Tai's classic

The problem of determining the minimum cost sequence of edit operations for transforming one (rooted) ordered, labeled tree into another one without any restrictions was first investigated in [112]. Tai's algorithm solves the problem for two trees T_1 and T_2 in polynomial time, in $\mathcal{O}(|T_1| \cdot |T_2| \cdot depth(T_1)^2 \cdot depth(T_2)'^2)$, where the depth of a tree is the maximal depth over all its nodes.

The tree edit model generalizes the edit distance for sequences to trees. With the basic edit operations we may relabel (R) a node, insert (I) one or delete (D) one, as shown in Figure 3.3.

An edit operation can be written as a pair $(a, b) \neq (\lambda, \lambda)$, or also as $a \rightarrow b$. The operation $a \rightarrow b$ is a *relabeling* if $a \neq \lambda$ and $b \neq \lambda$, a *deletion* if $b = \lambda$ and an *insertion* if $a = \lambda$ (compare [112, 103]).

Mappings are introduced to describe the edit operations to be applied to each node. Informally spoken, a mapping is a description of how a sequence of edit operations transforms one tree into another, without specifying the order of edit operations. Each sequence of basic edit operations corresponds to a mapping.

In a more formal way, a mapping M is a set of pairs of node indices (i, j) with the following properties:

1. $1 \leq i \leq |T|$, $1 \leq j \leq |T'|$, where T and T' are trees.
2. For any two pairs (i_1, j_1) and (i_2, j_2) in M,
   a) $i_1 = i_2$ iff $j_1 = j_2$ (one-to-one correspondence)
   b) $i_1 < i_2$ iff $j_1 < j_2$ (sibling preservation)
   c) $T[i_1]$ is an ancestor (descendant) of $T[i_2]$ iff $T'[j_1]$ is an ancestor (descendant) of $T'[j_2]$ (ancestor preservation).

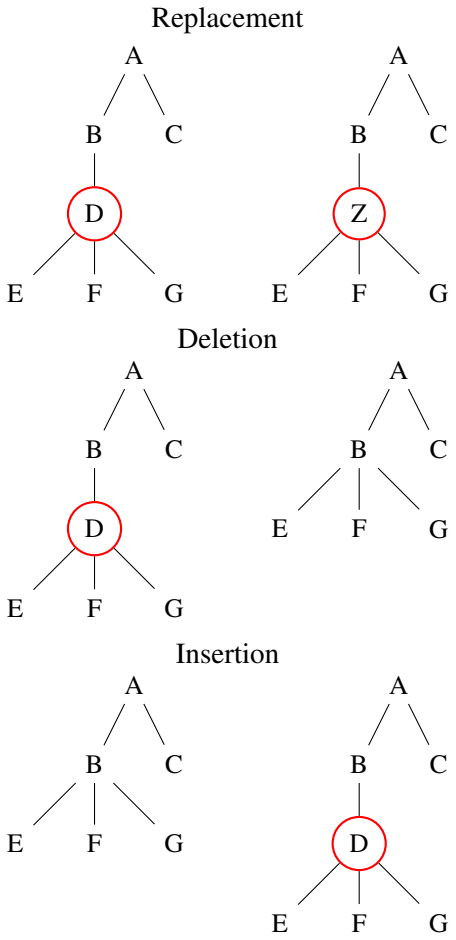


Figure 3.3: Tree edit operations: Relabeling, deletion and insertion of a subtree. Figure modified from [103].

Recall that $|T|$ gives the number of nodes of a tree T, and $T[i]$ is the node at index i.

See Figure 3.4 for an illustration of a mapping between two trees.

Mappings can be concatenated to obtain a new mapping, denoted by $M_1 \circ M_2 = M_3$. Assigning a cost to each basic edit operation, the distance between T_1 and T_2 is the smallest sum of costs along all editing paths between T_1 and T_2. We refer the reader to [103, 142, 63] for further details on the tree edit distance.

The algorithm computes a minimal mapping between the input trees. A suggested application of the algorithm could be problems of syntactic error recovery and correction for programming languages.

As a related theoretical problem, the author refers to the longest common subsequence problem [41, 50] from sequence analysis, and generalizes it to trees as the largest common substructure problem. Also the string-to-string-correction problem with the concept of traces [121] is related to this problem, and its extension by allowing to interchange adjacent characters might

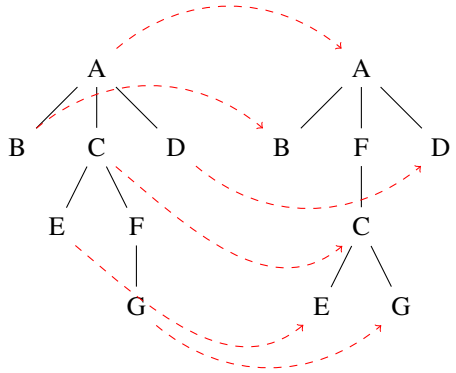


Figure 3.4: Mapping, based on [103]; Each sequence of basic edit operations corresponds to a mapping.

Figure 3.5: The classical edit distance is 1, we just have to delete node b.
Lu's edit distance is 2, we have to delete node a in T_1 and then replace node b by a.
A direct deletion of node b would map node a to a, and the subtree rooted at b could only map to one of the subtrees of T_2, thus resulting in a distance of 3.

be transferred to trees. Other suggested extensions are allowing the interchange of adjacent characters, or restriction of delete and insert to the leaves. The algorithm is developed for general tree comparison, and not for RNA secondary structures in particular.

Third variant by Lu: A tree matching algorithm based on node splitting and merging

Another approach to tree comparisons is to use node splitting and merging as basic edit operations. This approach is presented in [77, 78]. It computes a different edit distance than Tai's algorithm, as it considers each subtree as a whole entity and does not allow one subtree of T_1 to map to more than one subtree of T_2. With the classical edit distance, we may delete a root of one subtree, and then map the remaining subforest to more than one subtree. A minimal example, inspired by [8], is depicted in Figure 3.5:

For trees with depth two, Lu's algorithm computes the classical edit distance, but not for trees with a a greater depth. The classical algorithm developed by Tai is the most well known tree edit distance algorithm. Besides the three main variants of the tree edit distance, some other variants, improvements and applications have been developed using the above ideas as a starting point.

Tree edit distance with strongly preserving mapping

Tanaka's [114, 113] variant of the tree edit distance algorithm defines and uses the strongly structure preserving mapping. The main idea is that two separate subtrees of tree T_1 should be mapped to two separate subtrees of T_2, but this idea is not captured precisely in the formal definition. This mapping is supposed to make better correspondences between substructures because it further restricts the mapping by three required properties. Apart from using the different mapping, the algorithm is similar to Lu's algorithm.

Classical tree edit distance - Zhang-Shasha in the footsteps of Tai

More insights on Tai's classical tree comparison problem appeared in [141]. The distance between two trees is considered as the weighted number of edit operations to transform one tree into another, similar to [112]. Besides addressing the question of distance between two trees, the authors also compute the minimum distance between two trees with an arbitrary subtree removed, and as an extension, the minimum distance with zero or more subtrees removed. A special case of this problem solves the problem of approximate tree matching. The last question addressed is, whether two trees are within distance k of one another.

The first problem, the distance between two trees, is solved by a dynamic programming algorithm in sequential time $\mathcal{O}(|T_1|\cdot|T_2|\cdot depth(T_1)\cdot depth(T_2))$, an improvement compared to [112]. The authors also parallelize the algorithm, hereby achieving a runtime of $\mathcal{O}(|T_1| + |T_2|)$. The algorithm is extended to a variant with the same time complexity to address the second problem.

For the last problem of whether two trees are within a certain distance k of each other, another variant is presented with a runtime of $\mathcal{O}(k^2 \cdot \min(|T_1|, |T_2|) \cdot \min(depth(T_1), depth(T_2)))$. As opposed to [112], the authors consider distances between ordered forests in an intermediate step.

Restriction to unit costs

In [104], Shasha and Zhang show that distance computations of the best preceding algorithm in [112] may be simplified further if unit costs are used. They first present a dynamic programming algorithm with runtime $(k^2 \cdot N \cdot \min(depth(T_1), depth(T_2)))$ if k is the actual distance between T_1 and T_2 and N is $\min(|T_1|, |T_2|)$. This is an improved runtime compared to Tai's algorithm.

Two other algorithms are presented, which parallelize the solution of the same problem by using suffix trees. They have a runtime complexity of $k \cdot log(k) \cdot log(N)$, and $(k^2 \cdot log(k)) + log(N)$ on $k^2 \cdot N$ processors. RNA secondary structure comparison is discussed as an application for the algorithms, as comparison is necessary in understanding the comparative functionality of different RNAs.

An application to RNA secondary structures

A first application of tree comparison in the field of RNA secondary structures was made in [67]. RNA secondary structures are represented as ordered labeled trees, in which structural units are encoded as tree nodes.

The authors construct several alternative folding structures for a single RNA molecule, encode them as trees, and search for frequently reoccurring subtrees in this set of trees. The found secondary structure consensus motifs are used to construct a structure consensus model of the RNA. So in this case structure comparison is used as a means for structure prediction. Algorithmically, the method differs from the previous tree comparison algorithms. It computes the transferability ratio between two trees, that is the proportion of identical subtrees between two trees. So instead of an actual distance, the proportion of the common subtrees occurring in both trees is computed.

The algorithm is successfully used to detect the correct order of two tentative splicing mechanisms in the human $\alpha 1$ globin pre-mRNA splicing pathway. This is done by computing the transferability ratio between the mature mRNA and its precursor via the two splicing mechanisms in both possible orderings. A larger transferability ratio suggests that it is easier to change one structure into another, and in this way the correct pathway is chosen.

Multiple RNA secondary structure alignment

Based on a multiple RNA secondary structure alignment program which uses sequence alignment in previous publications [102], and the algorithms presented by [112, 141], a new algorithm is developed to compare multiple RNA secondary structures by [103]. Due to its definition on trees, the algorithm has a function for computing the distance of two trees and another function for computing the distance of two forests.

The algorithm is based on mappings as well, but precomputes a set of keyroots for each input tree for speedup. These keyroots are the roots of all subtrees that have left siblings, and the root of the tree. The other roots will be captured within the computation of the tree distance of the keyroots. By comparisons of the keyroots, the algorithm is able to compare two trees in $\mathcal{O}(|T_1| \cdot |T_2| \cdot L_1 \cdot L_2)$ with $L_i = min(depth(T_i), leaves(T_i))$ in the worst case, and is close to $\mathcal{O}(|T_1| \cdot |T_2|)$ for average 2D structure trees.

The pairwise comparison is done for multiple RNA secondary structures. Afterwards, a cluster algorithm produces a multiple structure clustering from the results, which can be depicted as a taxonomy tree.

Parents in the mapping

In Yangs variant [134] of the tree edit distance algorithm, the fact that two nodes are in a mapping implies that their parents are in the mapping, too.

Transfer to unordered trees

A different question is how to compute the editing distance between *unordered* labeled trees. Such trees are used for example in genealogical studies or phylogenetic analysis. For modeling RNA secondary structures we need an order to represent the sequence of bases in the molecule. A transfer of the edit distance algorithm to these unordered trees is presented in [142]. The authors show that the edit distance problem is NP-complete for unordered labeled trees. The constrained edit distance, which is discussed in the next subsection, has been transferred to unordered trees in [138].

Constrained edit distance

In [136, 138], algorithms are presented which compute a new version of the tree edit distance for both ordered and unordered labeled trees, and also approximate unordered tree matching. The main idea is that two separate subtrees of tree T_1 should be mapped to two separate subtrees of T_2. The authors are influenced by [114], who define their structure preserving mapping based on the same idea, but do not capture the idea precisely.

The shown algorithms compute the new edit distance in sequential time, in $\mathcal{O}\left(|T_1| \cdot |T_2| \cdot \max(deg(T_1), deg(T_2)) \cdot log_2(\max(deg(T_1), deg(T_2)))\right)$, where the degree $deg(T)$ of tree T is the maximum degree over all nodes, whereas computing the usual edit distance between unordered labeled trees has been shown to be NP-complete in [60, 142]. These previous NP-completeness results are reviewed, and the new "constrained" edit distance is introduced based on the aforementioned idea. Another constrained edit distance algorithm for ordered trees is presented in [93].

Tree pattern matching

The algorithm by Wang et al. [124] can be used for (approximate) tree pattern matching. A good general overview on tree pattern matching, tree edit and tree alignment distance is given in the book "pattern matching algorithms" by Apostolico and Galil [8]. The authors discuss exact and approximate tree pattern matching, tree edit and tree alignment algorithms on ordered trees, also defining the tree alignment problem and the tree pattern discovery problem. For unordered trees they discuss hardness results (= proved absolute or relative statements about the problem complexity) and give a tree alignment algorithm and a heuristic for the tree edit distance.

Efficient parallel algorithms for tree editing problems

Based on the preceding definitions of the tree edit distance, Zhang in [137] develops a framework to solve tree edit problems in parallel. Using this framework, he develops some poly-logarithmic time algorithms addressing the tree edit problem.

Some MAX SNP-hard results concerning unordered labeled trees.

In [139], Zhang and Jiang address the two problems of finding the largest common sub-tree of two trees, and of finding the edit distance between the trees. The authors prove that both problems are MAX SNP-hard, which means that they do not have a polynomial time approximation scheme (PTAS) unless $P = NP$, i.e. they cannot be approximated by a solution within $1 + \varepsilon$ for a given factor ε within polynominal time.

Approximate common substructures based on a restricted edit distance

A dynamic programming algorithm for identifying the largest approximately common substructures (LACSs) of two ordered labeled trees is presented in [123]. In this context, substructures of trees are connected subgraphs. The LACS problem is to find a substructure of the first tree and a substructure of the second tree, such that the first substructure is within a given distance d of the second substructure. Additionally, there must not exist any other substructure of the first

tree and of the second tree such that the substructure of the first tree and the substructure of the second tree satisfy the distance restriction and the sum of the sizes of these other substructures is greater than the sum of the sizes of the aforementioned two substructures. In this publication, the used edit distance is based on Selkow's distance [101], but has some restrictions. An algorithm is given which solves the problem in $\mathcal{O}(d^2 \times |T_1| \times |T_2|)$, so it is as fast as the best known algorithm for Selkow's distance, given the allowed distance within two substructures as a constant.

An improvement of Zhang-Shasha, especially for unrooted trees

In the publication by Klein [63], the edit distance between ordered trees is studied. The problem is again to compute a minimum series of edit operations, but the edit operations are now label modification and edge contraction. In the algorithm, the trees are represented as Euler strings (parenthesized strings) and the algorithm deals with a string alignment problem. For the comparison of substrings of Euler strings, a simplified and less efficient variant of the Zhang-Shasha algorithm is presented. From this starting point, a faster dynamic programming algorithm is developed by taking care of special substrings and by using a decomposition of the tree into paths. The improved algorithm has a better worst case bound than the Zhang-Shasha algorithm for rooted ordered trees, and it also is suited for the unrooted case (Zhang-Shasha does not work here). For some cases, Zhang-Shasha may be still faster, due to its different path decomposition strategy.

Tree inclusion as a related problem

A variant of the tree edit distance problem is the tree inclusion problem, which asks whether a pattern is embeddable in a tree. This could of course also be done by deleting nodes from the tree until it is transformed to the pattern - that means the underlying algorithm is similar to the one that solves the tree edit distance problem, but insertions are forbidden.

The tree inclusion problem is studied by Kilpelainen and Mannila in [60] on both ordered and unordered trees. To avoid searching all exponentially many embeddings of the pattern in the tree, Kilpelainen and Mannila use "left-embedding", i.e. they embed the pattern in the tree by embedding the subtrees as deeply and as far to the left as possible. For this variant, their algorithm needs a runtime complexity of $\mathcal{O}(|T_1| \times |T_2|)$. Kilpelainen and Mannila also prove that the tree inclusion problem for unordered trees is NP-complete.

More recent work on the tree inclusion problem appeared in [6].

Improvement: A Linear Tree Edit Distance Algorithm for Similar Ordered Trees

To make a linear tree edit distance algorithm possible, Touzet in [117] improves the original Zhang-Shasha-Algorithm by pruning the search space, if a bounding number of errors k is given in advance.

For the search space, they represent the possible edit operations to transform one tree into the other as an edit graph, where the vertices are the Cartesian product of the two tree's node indices in postorder traversal. The incident arcs represent deletion, insertion and substitution edit operations.

To prune the search space, such that only relevant vertices and arcs remain, she uses three optimization strategies. The first strategy is similar to the k band alignment algorithm for strings, as it computes a band of the edit graph only, constrained by the number of allowed errors k. Errors are operations that affect skeleton of tree, that means insertions and and deletions. The best mappings between two trees are near the diagonal of the edit graph, depending on the number of errors. For k errors they define the k strip of two trees as the set of basepairs that are at most k errors apart from each other. If there is a mapping between two trees, that visits a node (x, y) in the edit graph, and has at most k errors, then (x, y) is within k strip of the two trees. The next property is a boundary for the maximal number of errors, and defines a vertex to be k-relevant if it is within said bound. Other vertices may be pruned. The third optimization strategy uses the fact that when calculating the distance between two nodes (x, y) with at most e errors for two subtrees $A(x)$ and $B(x)$, pairs (i, j) have the property that $depth(i) \leq depth(x)+e+1$. All other vertices with $depth(i) > depth(x) + e + 1$ may be pruned.

For the trivial bound of $k = |A| + |B|$, the algorithm has the same complexity as the Zhang-Shasha algorithm ($\mathcal{O}(n^4)$), and the average complexity would be worse compared to Zhang-Shasha's, as some optimizations have been left out.

Generalization: Analysis of tree edit distance algorithms, Decomposition Algorithms for the Tree Edit

In [24, 25], Dulucq and Touzet generalize several tree edit distance algorithms by describing them as decomposition strategies. The main algorithms they study are the ones of Klein [63] and Zhang-Shasha [140].

In analogy to string comparison, they describe that the recursive call decomposes the remaining part of the data structure. For forests, these decompositions have two directions, as forests are recursive in two directions. Dulucq and Touzet define them as left and right decomposition. Both left and right decomposition are applied during tree comparison algorithms in an alternating fashion. Such an alternating fashion they call a (decomposition) strategy. Each strategy has a set of recursive calls. The forests that are involved in these recursive calls are called relevant forests (compare terminology of [63]).

The central idea is the concept of cover strategies: Let $l(F) \circ T$ be a relevant forest for a given strategy, where $\circ$ combines two trees to a forest, and $l(F)$ is the tree with root l and forest F as children. Now if the direction of $l(F) \circ T$ is left, this decomposition yields three relevant forests: $l(F)$, T and $F \circ T$. The forest T is a subforest of $F \circ T$. The goal is then to eliminate nodes of F in $F \circ T$, so that $F \circ T$ and T share as many relevant forests as possible. This idea is formalized in the definition of covers.

Dulucq and Touzet introduce a general framework of cover strategies, analyze the complexity of cover strategies and develop a new tree edit distance algorithm, optimal for such cover strategies.

Tree Edit Distance With Gaps

Starting from the special case of strings, Touzet studies the definition of edit distance with convex gap weights in [116]. She proves that there is no polynomial algorithm for the tree edit distance

problem with convex gap weights, unless $P = NP$. In this case, a convex gap function satisfies $Gap(t_1(t_2)) \leq Gap(t_1) + Gap(t_2)$, where t_1 and t_2 are subtrees, and $t1(t2)$ is a subtree such that t_2 is attached to one of the leaves of t_1.

Furthermore, she restricts the definition of gaps to complete subtrees (such that all descendants belong to the subtree), and presents a quadratic algorithm for the according tree edit distance.

As most of the standard tree alignment algorithms work with linear gap costs accounting for each single deleted node, they may be adapted to deal with affine gap costs, with gap opening and extend costs.

Implementation of tree Edit distance in the *Vienna package* - Fast folding and comparison of RNA secondary structures

The *Vienna RNA package* [45], a software package for RNA analysis, folding and comparison, provides an implementation of the tree Edit distance in the tool RNAdistance. As the tree edit distance algorithm itself is not dependent on the type of tree representation, several representations are available (compare 2.3). A full resolution structure representation as introduced by [30] is supported by default, which can be input in the dot-bracket notation like e.g. `.((..(((...)))..((..)))).`. This structure may be transformed to homeomorphically irreducible trees (HIT) by contracting each stretch of `m` unpaired bases into `(Um)`, and each stretch of `n` nested brackets enclosing substructure `s` into `(sPn)`, as shown in [30]. These HIT structures are also allowed as input for RNAdistance. For the above example the HIT structure would be `(U)((U2)((U3)P3)(U2)((U2)P2)P2)`. Then there is the coarse grained structure [102, 103], which may represent stems like e.g. `((((H)S)((H)S)M)S)`, or abstract from them as in `((H)(H)M)`. A weighted coarse grained structure representation is also possible, given as e.g. `((((H3)S3)((H2)S2)M4)S2)E2)`.

Tree edit distances of unrooted unordered trees

Algorithms have also been proposed for the tree edit distance of unrooted unordered trees, e.g. [113, 143]. However, we focus on rooted ordered trees because we want to compare RNA secondary structures, which are represented as these types of trees.

Discussion of tree edit distances

There are a lot of different tree edit distance variants, which also make different assumptions about the underlying model. Some algorithms allow insertions and deletions only at the leaves, some require that the edit operations are performed in a certain order, and also the types of basic edit operations and the trees the algorithms work upon differ. In [131] the tree alignment problem is treated as an optimization problem, and solved via recurrences and backtracking, but not in true dynamic programming spirit, and the runtime is exponential. The tree edit distance is a metric by definition [141], as the cost function for edit operations γ is constrained to be a metric, and the metric properties are transferred to the edit distance function, which is defined as the minimal possible sum of costs for basic edit operations transforming one tree into another.

Both known normalized variants of the tree edit distance [94] are no metric any more as they do not fulfill the triangle inequality. Recently, Li and Chenguang proposed a normalization in the range [0, 1], which still has metric [71] properties.

3.5.2 Tree alignment distances

Tree *alignment* as an alternative to tree edit

The tree alignment algorithm, an alternative to the tree edit algorithm, is the foundation for the method described in this thesis. It was first mentioned Jiang et al. in [55]. The authors address the question of comparing both ordered and unordered trees. Solutions for unordered trees and multiple alignment are discussed as well.

Jiang et al. propose that both tree edit distance [100, 141, 112] and transferable ratio [67] are valid distance measures, but capture not all functional similarity.

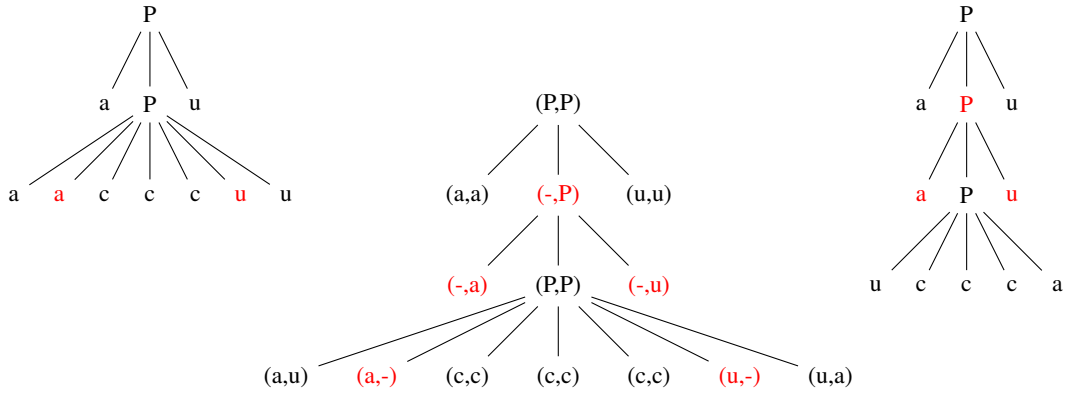


Figure 3.6: An example for the tree alignment model.

The tree alignment The important new concept is the concept of an alignment: An alignment of two structures with labels from some alphabet $\mathcal{A}$ is the same type of structure with labels from the pair alphabet $\mathcal{A}_{pair} = \mathcal{A} \cup \{-\} \times \mathcal{A} \cup \{-\} \setminus \{(-,-)\}$.

The pair nodes represent the edit operations, where nodes of type (a, b) are *replacements*, $(a, -)$ are *deletions* and $(-, a)$ are *insertions*.

Applied to rooted ordered trees that represent RNA secondary structures, an alignment A is a tree on the pair alphabet of RNA with gaps. An example tree alignment of two hairpin structures is shown in Figure 3.6.

The projections to the first and second component, $A|_1$ and $A|_2$, are trees on the alphabet of RNA with gaps. These trees may be contracted by π, where $\pi(T)$ is the tree that results from deleting all nodes with the gap symbol from tree T. The result of this contraction are the input trees.

This observation gives us an elegant definition for a tree alignment:

Definition 3.5.1. $A \in \mathcal{T}(\mathcal{A}_{pair})$ is an *alignment of trees* $T_1, T_2 \in \mathcal{T}(\mathcal{A})$ iff $T_1 = \pi(A|_1)$ and $T_2 = \pi(A|_2)$.

The trees where the deletion or insertion of a node produces a forest are not included in this definition. Nevertheless, they appear as extra subcases in the calculation.

The score of an alignment is the sum of the scores for the edit operations which are indicated by the pair node labels of the result tree.

$$\sigma(A) = \sum_{v \in V(A)} \sigma(label(v))$$

The tree alignment distance between two trees is the minimum cost over all possible alignments of the two trees.

$$\sigma(T_1, T_2) = min\{\sigma(A) \mid A \text{ is an alignment of } T_1 \text{ and } T_2\}$$

By reading the node labels in sequence of preorder traversal, it is possible to construct a corresponding edit sequence from any tree alignment. Also, due to the properties of the alignment tree on the pair alphabet, it is possible to construct a mapping from it.

On the contrary, it is not always possible to construct an alignment from a mapping as can be seen in Figure 3.7.

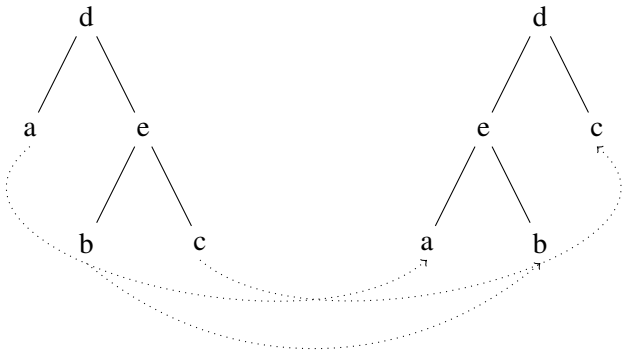


Figure 3.7: Example of a mapping without a corresponding alignment. Node b of T_1 is mapped to node b of T_2, so a node (b, b) should be present in the corresponding alignment forest. This node must be descendant of the nodes $(e, -)$ and $(-, e)$, which contradicts the definition of a tree, as a node can have only one direct ancestor.

The tree alignment distance algorithm computes a subset of the search space of the tree edit distance algorithm, as the alignments are a subset of the tree edit distance mappings. Jiang states that the tree alignment corresponds to a restricted tree edit distance, "in which all the insertions precede all deletions" [55].

The tree alignment distance is not a metric, as it does not fulfill the triangle inequality. The counter-example in Figure 3.7 is adapted from [42].

The presented algorithm for ordered trees has time complexity $\mathcal{O}(|T_1| \cdot |T_2| \cdot (deg(T_1) + deg(T_2))^2)$, where $deg(T_i)$ is the degree of T_i, so the algorithm is faster than all known ones for the tree edit distance, if the degrees are smaller than the depths of the trees.

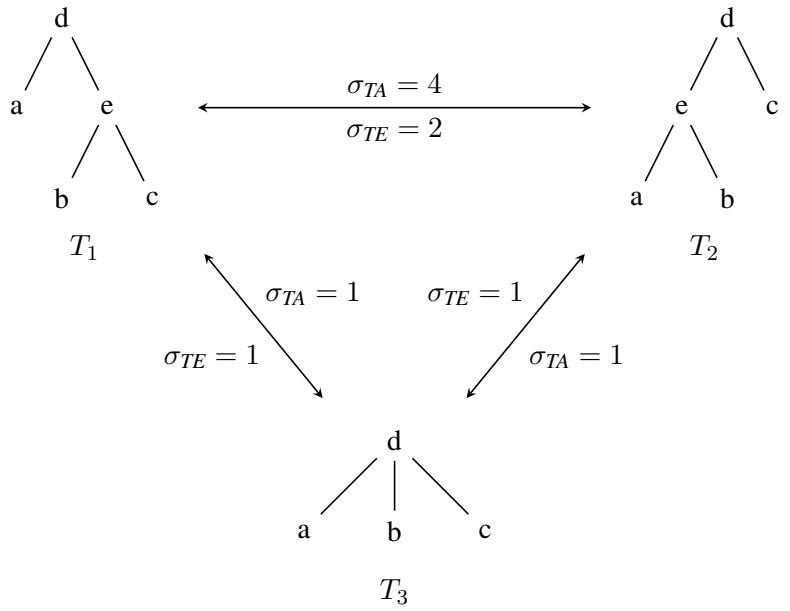


Figure 3.8: Counter-example to show that the tree alignment distance does not fulfill the triangle inequality. With the unit cost, $\sigma(T_1, T_2) > \sigma(T_1, T_3) + \sigma(T_2, T_3)$. The tree edit distance fulfills the triangle inequality. Figure adapted from [42].

Besides this comparison of tree alignment and tree edit, as a theoretical insight, the tree inclusion problem is classified as a special case of a tree alignment. It is also mentioned that the optimal tree edit distance leads to the largest common subtree of two trees, and the optimal tree alignment distance leads to the smallest common supertree of two trees. These statements are assessed in detail by other researchers, which will be discussed in Section 3.5.3.

Approximate local alignment: Finding approximate common substructures

Both tree edit and tree alignment distance were introduced before, and the tree edit distance was generalized for both rooted and unrooted trees, and extended to the problem of graph comparison. Wang et al. in [125] extend this foundational work by addressing the problem of finding the largest approximate common subtrees for ordered labeled trees. Their algorithm is as fast as the one for the edit distance, given a maximum allowed distance for common substructures. They use their solution to find motifs in RNA molecules, represented as trees, with respect to the structure of the motif. The algorithm is based on the same edit operations and the idea of computing a minimal mapping. For an easy definition of the term "substructure", a cut operation is defined. The authors also discuss that there are several variants ([72, 73, 74]) of this problem, depending on the definition of the term "substructure".

Restricted tree alignment distance between similar ordered trees

A fast algorithm for aligning two similar ordered trees with node labels was presented in [53]. In this variant, the runtime of the algorithm is depending on the number of used gap symbols, which are introduced into the tree by deletions and insertions. The general idea of this algorithm is to modify the tree alignment distance by restricting it to a special type of subtrees and subforest called *d-relevant*. Two subtrees are d-relevant if their numbers of nodes are within distance d, and the numbers of leaves to the left of the root of each subtree are within distance d of each other. In this way, an optimal alignment using at most d blank symbols can be constructed in $\mathcal{O}(n \cdot logn \cdot (maxdeg)^4 \cdot d^2)$ time, where $n = \max |T_1|, |T_2|$ and $maxdeg$ is the maximum degree of all nodes in both trees.

Local similarity in RNA secondary structures

In [43], Hoechsmann et al. extend the tree alignment algorithm to compute local forest alignments with a time complexity of $\mathcal{O}(|F_1| \cdot |F_2| \cdot deg(F_1) \cdot deg(F_2) \cdot (deg(F_1) + deg(F_2)))$, where $|F_i|$ is the number of nodes in forest F_i and $deg(F_i)$ is the degree of forest F_i, the maximum degree over all nodes. The algorithm uses a dense two dimensional dynamic programming table, and considerably reduces the space requirements compared to previous versions of the algorithm. Hoechsmann et al. take a unified point of view and define and solve all alignment problems on forests, as this more general problem has to be solved in the dynamic programming recurrences anyway. Matching this generalization, a new representation for RNA secondary structures as forests is being presented. With their algorithm, Hoechsmann et al. are able to find potential regulatory motifs just based on their structure, and independent of their sequence.

Progressive RNA secondary structure alignment

Based on the forest alignment model for the alignment of two trees, a multiple alignment model is being developed by Hoechsmann et al. in [44]. This is done by using the profile alignment method, which can be transferred from strings to trees and forests. The central idea of profile alignment is a profile data structure, which consists of (e.g. nucleotide) frequency vectors, one for each position in the alignment. In each step, the next sequence from the input set is aligned to the profile, resulting in an new profile. A forest profile representation for RNA secondary structure alignments is presented, and along with it an algorithm to compute the profile alignment, and a visualization to illustrate the alignment is shown. Again, the alignment can also be computed as a pure structural alignment. A result of this work is the alignment tool for RNA secondary structures called *RNAforester*.

RNAforester has been successfully used in a large scale study by [95] to separate microRNA precursors from other hairpins by using secondary structure comparison to detect conserved structures. The structural features of the microRNA precursors are difficult to assess, as the MFE and the basepairs are not very different from other conserved hairpins. The candidates were human noncoding RNAs from an RNAz screening (Washietl et al., [127]), altogether 1200 noncoding RNA candidates. After folding them with RNAalifold, they were compared pairwise with *RNAforester* to construct a $1200 \cdot 1200/2$ size distance matrix. UPGMA clustering revealed that the microRNA precursors separate from other hairpin structures.

3.5.3 Tree edit and tree alignment

A few publications investigate the relationship between the tree edit distance and the tree alignment distance. At the moment, there is only one publication that tries to review most of the proposed methods in this area.

A survey on tree edit distance and related problems

A survey on the problem of comparing labeled trees based on simple edit operations has been conducted by Bille in [14]. As seen in the above paragraphs, the basic edit operations insertion, deletion and replacement lead to the tree edit distance, the tree alignment distance and the tree inclusion problem.

The published solutions for each problem are reviewed and presented in a nutshell. The publication makes a successful effort to clear up the field of tree comparison, however is mainly a synopsis of the common methods, and not so much concerned with relating them to each other, which is my goal and interest in this chapter of my thesis.

Seeded tree alignment

The method of seeded tree alignment [75] is an interesting combination of mapping and alignment. In spite of its name, it constructs mappings rather than alignments.

Seed mappings, a set of node pairs which have to map onto each other, are used to constrain the mappings. The seed mappings preserve the lowest common ancestor relationship, and in this way select a specific common super-tree structure. This makes the mappings compatible with, while still more abstract than, tree alignments.

On the other hand, the lowest common ancestor nodes of two seed nodes have to be mapped onto each other. This is more restrictive than requiring the existence of a compatible super-tree.

Based on these two main reasons, the seeded tree alignment approach cannot be compared to our use of anchors explained in Chapter 7, which does not enforce preservation of lowest common ancestors.

Finding Largest Subtrees and Smallest Supertrees

Tree comparison is used in the context of different application areas, and thus is discussed with different nomenclature. This problem is addressed in [38].

In [38], two problems are addressed, the isomorphic subgraph problem and the topological embedding. Both may be generalized as embedding problems. Two algorithms are presented, one algorithm for determining a largest tree embeddable in two trees, a largest subtree, and another algorithm for finding a smallest tree in which each of two trees can be embedded, a smallest supertree.

This is a more general formulation of what the tree edit and the tree alignment distance compute (see Figure 3.9).

The sub- and supertrees may be used to structure incomplete sets of evolutionary data, and to compute the similarity between two pieces of structured data, such as documents, images or molecular structures. The framework presented in this publication is suited for sequential

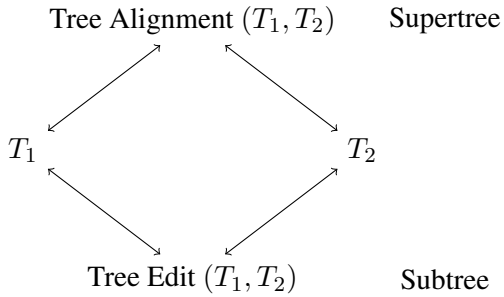


Figure 3.9: The tree edit distance, tree alignment distance, subtree and supertree.

and parallel algorithms, and new proposals are given for sequential an parallel algorithms. The algorithms work better on ordered than on unordered trees, so they are well suited for RNA secondary structure comparison.

An algebraic view of the relation between largest common subtrees and smallest common supertrees

The work by Rosselló and Valiente [96] gives further insights on the largest common subtree and the smallest common supertree problem. The authors study the relation between these problems by means of category theory. The central concept used in this publication is the concept of embeddings. A tree can be embedded into another tree when there exists an injective mapping that transforms edges into paths.

The properties of this mapping define the type of the embedding, and four types are presented: *isomorphic*, *homeomorphic*, *topological* and *minor* embedding. In this way, the authors construct a general framework, in which the other approaches of tree distances and related problems concerning subtree isomorphism and tree inclusion can be subsumed.

The authors relate the problem of finding the largest common subtree and the smallest common supertree for the different embedding cases, and describe a method for constructing a largest common subtree out of the smallest common supertree, and vice versa.

3.5.4 Discussion of the tree and forest comparison models

The tree edit and tree alignment distance and their underlying models both are based on the tree representation of RNA secondary structures.

Note that, with both models, the distance between two structures of different lengths can be computed.

The distances can be applied to rooted ordered trees with any kind of labels. To use the tree edit algorithm for RNA structure comparison, the coarse grained tree representation is widely used for the structures to compare (compare 2.3).

An advantage of the tree based models is that variation of the tree representation and editing costs facilitates to tailor metrics to either global or local analysis. A finer tree representation is better for local analysis, and a coarser tree representation is better for global analysis.

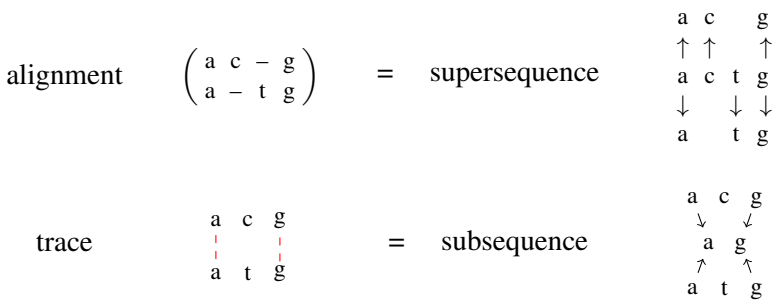


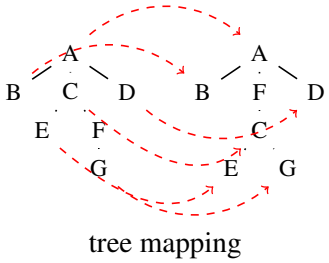
Figure 3.10: Alignment and trace in sequence comparison: the trace implicitly encodes a subsequence of the two input strings, whereas the alignment implicitly encodes a supersequence.

Differences between tree edit and tree alignment distance It is important to distinguish between tree edit and tree/forest alignment distance algorithms, although the two have been mixed up in the past, because both claim to extend the string edit distance to trees.

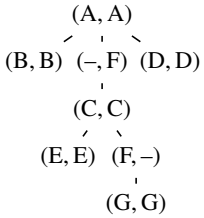
The tree edit distance computes a mapping, whereas the forest alignment algorithm computes an alignment forest as a data structure.

Transferred to sequence comparison, a mapping would be equivalent to the concept of a trace, as it maps replaced symbols to each other, but does not provide information about the placement of the gaps in the alignment. This difference is explained in Figure 3.10.

A tree alignment would be equivalent to an alignment or edit transcript in sequence comparison, a sequence of basic edit operations. In the edit transcript, the edit operations are represented as pairs. So this sequence of pairs is the result from comparing two sequences, and in the same way, a tree with pair nodes (= a tree alignment) is the result of aligning two trees.



tree mapping



tree alignment

sequence trace

sequence alignment

The set of alignments computed by the tree alignment distance is a subset of the mappings computed by the tree edit distance, as discussed in Section 3.5.2. The tree edit distance is

equivalent to computing the largest common substructure, whereas the tree/forest alignment distance computes the smallest common superstructure of the two input trees. These general observations have been further investigated by [38, 96].

Related distance measures Some distance measures have acquired different names over time, such as the isolated subtree distance [114], which is the same as in 3.5.1.

The important distance models on trees that we have discussed above, tree edit distance, tree alignment distance and isolated subtree distance, have two other less known relatives:

The *bottom up distance* computes the largest common forest between two trees, independent of the particular edit costs, on ordered and unordered trees [118]. In this measure, two nodes match only if all nodes in the subtrees below them match as well.

The *top down distance*, also *similar consensus problem*, computes the largest approximately common substructures [122].

Related other problems Several related problems on trees give more insight to the connection between tree edit and tree alignment distance as well.

The *tree inclusion problem* asks whether for two given rooted ordered trees S and T, tree S can be obtained from tree T by deleting nodes ([6, 92]. The problem of *(sub)tree isomorphism* is the isomorphic subgraph problem for G and H, when G and H are trees. The goal is to find a subtree of G which is isomorphic to H, or prove that such a tree does not exist. This problem has also been treated under different names.

Summary As the number of introduced algorithms for tree comparison is large, we sum up the most important properties in the following Tables 3.1 and 3.2.

Table 3.1: Synopsis of common tree comparison algorithms. In the table, D_i is the depth, L_i the number of leaves and I_i is the maximum degree of tree T_i. The value u is the unit cost edit distance between the two trees, and s is the number of spaces in the optimal alignment of the trees. $\sum_{T_i}$ is the set of labels of T_i and m_{T_1,T_2} is the number of pairs of nodes from both trees which have the same label.

Tree edit distance

variant	tree type	time	space	reference
indel leaves only	unordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	[101]
general	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot D_1^2 \cdot D_2^2)$	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot D_1^2 \cdot D_2^2)$	[112]
general	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot \min(L_1, D_1) \cdot \min(L_2, D_2))$	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	[141]
general	ordered	$\mathcal{O}(\|T_1\|^2 \cdot \|T_2\| \cdot log(\|T_2\|))$	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	[63]
general	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\| + L_1^2 \cdot \|T_2\| + L_1^{2.5} \cdot L_2)$	$\mathcal{O}((\|T_1\| + L_1^2) \cdot \min(L_2, D_2) + \|T_2\|)$	[21]
general	unordered	MAX-SNP hard	MAX-SNP hard	[139]
constrained	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	[136]
constrained	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot I_1 \cdot I_2)$	$\mathcal{O}(\|T_1\| \cdot D_2 \cdot I_2)$	[93]
constrained	unordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\|(I_1 + I_2) \cdot log(I_1 + I_2))$	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	[138]
less-constrained	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot I_1^3 \cdot I_2^3 \cdot (I_1 + I_2))$	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot I_1^3 \cdot I_2^3 \cdot (I_1 + I_2))$	[76]
less-constrained	unordered	MAX-SNP hard	MAX-SNP hard	[76]
unit cost	ordered	$\mathcal{O}(u^2 \cdot \min(\|T_1\|, \|T_2\|) \cdot \min(L_1, L_2))$	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	[103]

Table 3.2: Synopsis of common tree comparison algorithms, continued. In the table, D_i is the depth, L_i the number of leaves and I_i is the maximum degree of tree T_i (or forest F_i). The value u is the unit cost edit distance between the two trees, and s is the number of spaces in the optimal alignment of the trees. $\sum_{T_i}$ is the set of labels of T_i and m_{T_1,T_2} is the number of pairs of nodes from both trees which have the same label.

Tree (and forest) alignment distance

variant	tree type	time	space	reference
general	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot (I_1 + I_2)^2)$	$\mathcal{O}(\|T_1\| \cdot \|T_2\| \cdot (I_1 + I_2)^2)$	[55]
general	unordered	MAX-SNP hard	MAX-SNP hard	[55]
similar	ordered	$\mathcal{O}((\|T_1\| + \|T_2\|) \cdot log(\|T_1\| + \|T_2\|) \cdot (I_1 + I_2)^4 \cdot s^2)$	$\mathcal{O}(L_1 \cdot \min(D_2, L_2))$	[53]
forest	**ordered**	$\mathcal{O}(\|F_1\| \cdot \|F_2\| \cdot I_1 \cdot I_2 \cdot (I_1 + I_2))$	$\mathcal{O}(\|F_1\| \cdot \|F_2\|)$	**[43]**

Tree inclusion

variant	tree type	time	space	reference
general	ordered	$\mathcal{O}(\|T_1\| \cdot \|T_2\|)$	$\mathcal{O}(\|T_1\| \cdot \min(D_2, L_2))$	[59]
general	ordered	$\mathcal{O}(\|\sum_{T_i} \|\|T_2\| + m_{T_1,T_2} \cdot D_2)$	$\mathcal{O}(\sum_{T_i} \|\|T_2\| + m_{T_1,T_2})$	[92, 93]
general	ordered	$\mathcal{O}(L_1 \cdot \|T_2\|)$	$\mathcal{O}(L_1 \cdot \min(D_2, L_2))$	[20]
general	unordered	NP-hard	NP-hard	[60, 82]

3.6 Distances between graphs

3.6.1 Multiple layer secondary structure comparison

Another version of the tree edit distance is presented in [2], which uses two new operations, called node fusion and edge fusion, besides the tree edit operations of deletion, insertion, and relabeling. This is done to avoid limitations which arise when using the classical edit distance for RNA secondary structure comparison. The authors study the complexity of their proposed new distance algorithm and claim that it remains efficient in practice.

This edit distance is applied to a special representation of RNA secondary structure, called MiGal for "Multiple Graph Layers" [3]. It consists of several different graphs which represent an RNA secondary structure at different levels of abstraction, and the graphs are linked to each other. An algorithm is given to compare two MiGaLs, starting at the most abstract level, and communicating the result of comparison down to the next level in several steps. By doing so, it is possible to compare RNA structures on different levels of abstraction at the same time with the MiGal algorithm and software.

3.6.2 A graph theoretical approach to predict common RNA secondary structure motifs including pseudoknots of unaligned sequences

The approach in [54] predicts common RNA secondary structure motifs in an ensemble of functionally or evolutionarily related RNA sequences. No initial structural alignment of the input sequences is required. By graph-theoretical methods, the stems of the sequences are compared. The stable stems in the sequences are determined, and then compared pairwise between sequences. A maximum clique finding algorithm detects conserved stems across at least k sequences. Then, all conserved stems shared by at least k sequences are assembled in topological order. The best-assembled stem sets are output, as they are the best candidates for common structure motifs. Due to the graph-based approach, the algorithm can also detect pseudoknot structures. It is tested on sequences with known secondary structures, and is capable of detecting the real structures completely or parts of them.

3.7 Discussion of graph based methods

Graphs are data structures that are powerful enough to reflect the properties of RNA secondary structures which are desired to be modeled. Many of the graph based methods of structure comparison also include pseudoknots. On the other hand, there is no common concept among the graph based approaches concerning the representation of a secondary structure, as there are various possibilities to do so with a graph. Another drawback is that graph based approaches can become computationally very expensive. This is a problem with tree and forest based methods already for longer input structures (we will investigate a speedup method in Section 7), which are only a subset of graphs. Workarounds for graph based methods are needed, and have been proposed, such as the multiple level approach in MiGaL [2].

3.7.1 Strengths, weaknesses and artefacts

Several methods for RNA secondary structure comparison have been proposed, which all have their strengths and weaknesses. Some of the discussed distance measures fulfill the metric properties, as subsumed in Table 3.3.

Table 3.3: Structure comparison methods and their metric properties.

	pos. definiteness	symmetry	triangle inequality	proof/counterex.
d_{ssd}	✓	✓	✓	3.1.1
d_h	✓	✓	✓	3.1.2
d_p	✓	✓	↯	3.1
d_m	✓	✓	✓	from l_p norm
Tree Edit	✓	✓	✓	3.5.1, [141]
Tree Alignment	✓	✓	↯	3.8

The strengths and weaknesses of the methods also depend on the representation of the secondary structure.

Basepair distances are conceptually simple, and also easy to compute, but also introduce artefacts. Artefacts are score differences that are due to the representation only, while there is no reason for them in the input sequences. For example a difference of one single basepair that leads to a huge distance in the resulting score is considered an artefact, as a small difference in the structure should be represented as a small change in the resulting score of the distance measure.

Sequence based methods that do not represent basepairs as entities cannot properly align nested basepairs, which may lead to wrong alignments not respecting the nesting of the a stretch of basepairs in a subset of structures by aligning wrong pairing bases as discussed in Section 3.2.4.

The arc annotated sequence model is very flexible, and different kinds of distance algorithms can be defined on it, even including complex structural elements such as pseudoknots. It is a very general model, and can even be used to model the tree edit and tree alignment distance by defining different sets of edit operations on arcs and incident bases.

The tree/forest alignment model naturally describes two common properties of RNA secondary structure, nesting and adjacency. In the representation, structural elements are subtrees, which are in ancestor relation or sibling relation to each other. And the representation as a tree is also a natural data structure in many other areas of computer science, which allows for a general approach and makes the algorithms reusable for other areas, as well as generic algorithms for tree comparison may be used for RNA secondary structure represented as a tree.

Therefore, we focus on tree based alignment methods. Tree edit distance and tree alignment distance both claim to extend the sequence alignment model to trees, but as discussed, the tree edit distance's computed mappings should rather be seen as a transfer of the trace concept from sequence comparison to tree comparison.

As the tree edit distance does not produce an alignment, and is also computationally more expensive due to its bigger search space of mappings rather than alignments, the tree alignment distance is our method of choice. Also, we think that forests are the more suitable way for both

representing RNA structures and presenting the algorithm in comparison to trees. So from now on, we will talk about the *forest alignment distance*.

Based on the forest alignment distance, we discuss the forest alignment model and the forest alignment algorithm, extend the model to incorporate affine gap costs and anchors, provide new algorithms and introduce the implementation of these algorithms in the tool *RNAforester 2.0*.

4 The forest alignment model

We will now introduce the underlying concepts of the forest alignment distance. The forest alignment model is the foundation for the algorithms in the following chapters.

We structure the introduction of the forest alignment in spirit of the concept of Algebraic Dynamic Programming (ADP, see e.g. [33]), and present the forest alignment approach separated into search space construction, and evaluation via a scoring scheme and an objective function. Although the ADP concept is helpful to formulate and understand the following approach to the forest alignment problem, it is no necessary prerequisite for the reader.

4.1 RNA secondary structures as forests

An RNA sequence or primary structure is represented as a string on the alphabet $\mathcal{A}_{\text{RNA}} = \{A, C, G, U\}$, where each symbol represents one of the bases of the modeled RNA sequence, as explained in Section 1.2. This is also the starting point for the underlying alphabet of our model of RNA secondary structures.

For secondary structures, we take sequence and basepairings into account. We represent a secondary structure as a forest, where the sequence is at the leaves.

4.1.1 From tree to forest

Why do we define our model on forests rather than on trees, in spite of the original model [55] by Jiang et al. being based on trees? In the original model, both tree and forest alignment steps are performed in the recurrences of the algorithm. If a node is deleted, the remaining forest of children has to be aligned and a forest alignment step is performed. We simplify the model by viewing trees as forests with length one. So we align forests in each step of the algorithm, and the tree alignment cases can be omitted from the recurrences.

4.1.2 Representing base pair information

To represent base pairings, we introduce additional nodes.

Why do we need these, if the natural tree representation works without such a concept? The new nodes, called P-nodes, represent the base pair bond, so we can later explicitly score the removal of such a bond in the alignment. Nodes that represent a base are called B-nodes.

The base pair is represented by the three connected nodes.

1. A *P-node*, labeled with P, representing the base pair bond.

2. Two or more child nodes. They are ordered according to the 5' to 3' direction of the molecule, and the leftmost and rightmost child node are the bases that pair.

In between these child nodes that are the pairing bases, there can be an arbitrary number of P-nodes and bases, if the base pair encloses a loop or a bulge (see also Figure 4.2) as nested substructure.

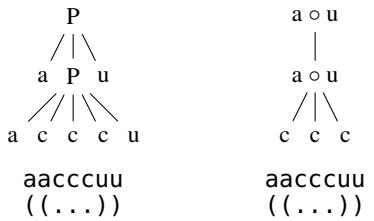


Figure 4.1: On the left you see the *extended forest representation* of a hairpin structure. As the hairpin is a closed structure, the forest has length one and is just a tree. Below is the dot bracket representation. On the right is the *natural tree representation* for comparison. It is more compact, but cannot handle base pair breaking operations or make a basepair out of two unpaired bases, as there is no way to relabel unpaired bases into basepairs and vice versa.

Thus, our representation of an RNA secondary structure is a forest on the alphabet $\mathcal{A}_{\text{forester}} = \{A, C, G, U, P\}$. We call this representation the *extended forest representation* (compare [42]). It is an advantage of the extended forest representation that each tree label is just one symbol (and not sometimes two, as in the natural tree representation), which keeps the alignment model much more simple and makes it more versatile.

See Figure 4.1 for an example an RNA secondary structure that forms a hairpin, shown in dot bracket, natural and extended forest representation.

The above definition is just one possible representation of an RNA structure as a forest. Like for trees, representations with different levels of detail can be used with the algorithm.

See Section 2.3 for an introduction of the most common tree representations for RNA secondary structure.

4.2 The forest alignment

Having a suitable representation for RNA secondary structures, the next step is to define the problem we want to address, the forest alignment problem.

The forest alignment problem answers the following question: Given two forests F and G, what is the minimum possible cost to transfer F and G into an alignment forest of F and G? We will now define what an alignment forest is.

4.2.1 The forest alignment

The tree alignment definition can be extended to the alignment of forests by seeing a forest as a tree with a fictitious root. An alignment of two forests with labels from some alphabet $\mathcal{A}$ is represented as a forest with labels from the pair alphabet, with an additional symbol "$-$" to represent gaps that are inserted into one of the input trees while constructing the alignment. The node labels (a, b), $(a, -)$, $(-, b)$ with $a, b \in \mathcal{A}$ represent the edit operations *replace (R)*, *delete (D)* and *insert (I)*.

The string alignment case is embedded into the generalization to forests, as a string is a forest of trees where each tree has just one leaf. As mentioned, a base pair is represented by three connected nodes. In the alignment forest, this allows explicit scoring of base pair deletions and insertions.

An example forest alignment of two hairpin structures is shown in Figure 4.2.

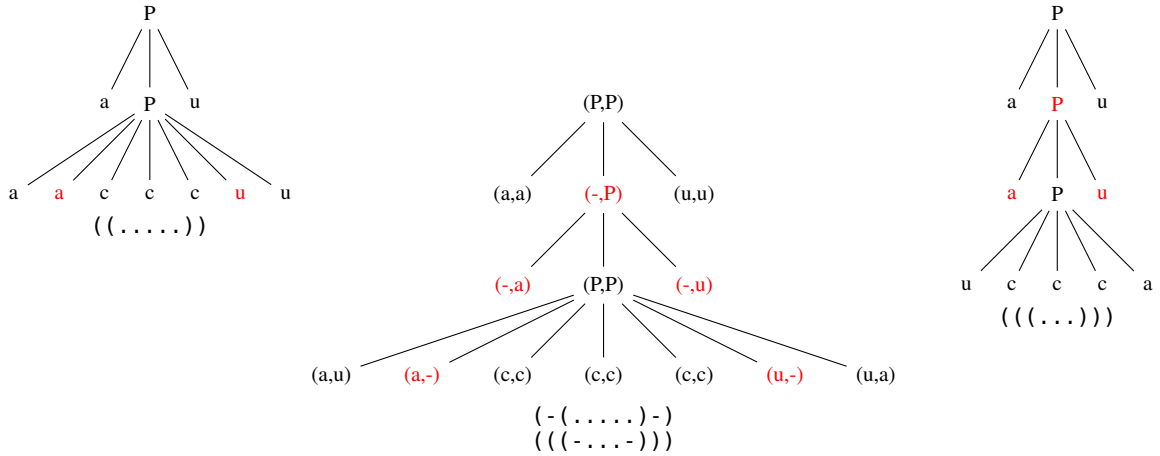


Figure 4.2: An example for the tree alignment model. At the top are two hairpin structures and their tree representations. Underneath is one possible alignment tree. The dot-bracket notation can be derived by traversing the trees and collecting the leaf node labels, where a P-label marks the position of a pair of brackets.

4.2.2 Defining a forest alignment

In the same way as for tree alignment, we can define a forest alignment. The alignment of two forests is a forest on the pair alphabet according to the following property:

Definition 4.2.1 (forest alignment)**.** $A \in \mathcal{T}(\mathcal{A}^2_{RNA\cup\{-\}})$ is an *alignment of forests* $F_1, F_2 \in \mathcal{F}(\mathcal{A})$ iff
$F_1 = \pi(A|_1)$ and $F_2 = \pi(A|_2)$

The projection function $|$ and contraction function π are defined for forests according to the definitions in Section 3.5.2 for trees.

By projection to the n-th component of the alignment forest, we can get back the n-th original forest that was input to the alignment algorithm, if we then contract the forest to get rid of all gap symbols that were introduced by the alignment algorithm.

RNA forests have to be aligned with attention to the P-nodes, which must be treated as an entity with their outermost children nodes, the bases that pair. An alignment adhering to these constraints is called a *well-formed RNA forest alignment*.

Definition 4.2.2. An alignment A of forests F and G is a *well-formed RNA forest alignment* iff for all relabeled P-nodes in A the corresponding B-nodes are relabeled in A.

To ensure that only well-formed RNA forest alignments are produced, some alterations have to be made to the general tree alignment distance algorithm as proposed by Jiang et al. in [55]. This will be discussed in Section 9.2.

4.3 Scoring the alignment

We compute the global similarity of two forests here, in analogy to the Needleman-Wunsch algorithm on sequences. A scoring function $\sigma : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ is associated with the edit operations.

The score $\sigma(A)$ of an alignment is the score sum of its edit operations.

To be able to compute local alignments, we use a score rather than a cost function for the algorithms in the following chapter. The optimal similarity score of the forest alignment is then the maximum score over all possible alignments:
$\sigma_{FA}(F, G) = max\{\sigma(A)|A$ is an alignment of F and $G\}$. Of course, it is also possible to minimize over a cost function for global alignments.

Scoring RNA alignments How do the associated scores for the edit operations look like in our forest alignment model? For sake of simplicity, let us see all nodes with a label $b \in \{A, C, G, U\}$ that stands for a base as *B-nodes*. Together with the remaining P-nodes, we now compare symbols from $\{P, B\}$ for the basic edit operations.

All possible combinations are (P, P), $(P, -)$, $(-, P)$, (B, B), $(B, -)$ and $(-, B)$, because in the well-formed RNA forest alignment model, a P-node is not allowed to be aligned to a B-node.

Insertions transforming the first into the second input tree could also be seen as deletions transforming the second into the first input tree. That is why we do not consider separate parameters for insertion and deletion of P- or B-nodes, but rather one "indel" parameter. As P-nodes are always labeled with a P, there is no such thing like a P-node mismatch. Taking these reasons into considerations, we have to score five different cases of edit operations, *base match*, *base replacement*, *base deletion*, *pair match*, and *pair deletion*.

The traditional term "pair deletion" is ambiguous, as it does not state, whether just the base pairing bond is deleted and the bases remain unpaired, or whether the bases are deleted together with their bond. For clarification, we could as well say "bond deletion", because the operation scores just the basepair bond which is deleted, and the pairing bases contribute their own scores.

A scoring scheme sets the parameters for the five edit operations. If base deletions, base matches and base mismatches are scored 0, the alignment becomes purely structural.

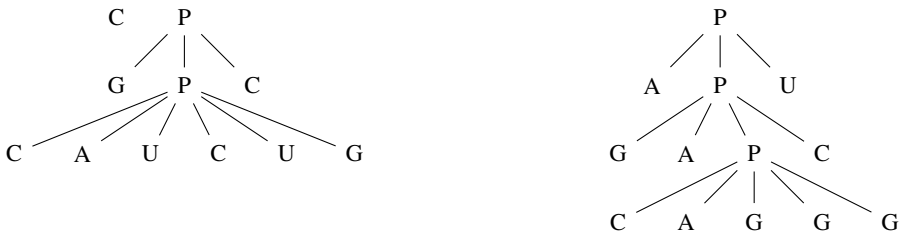
The forest alignment distance is not a metric, as the triangle inequality is not satisfied. The argument is the same as for the tree alignment distance, see a counterexample in Figure 3.8.

Scoring examples The following examples illustrate how a given forest alignment is scored under a given scoring scheme.

Let us assume that the following two structures in Vienna dot-bracket notation are our input structures.

```
Input 1                    Input 2
"CGCAUCUGC",               "AGACAGGGCU",
".((....))"                "((.(...)))"
```

In the forest representation, the input is as follows:

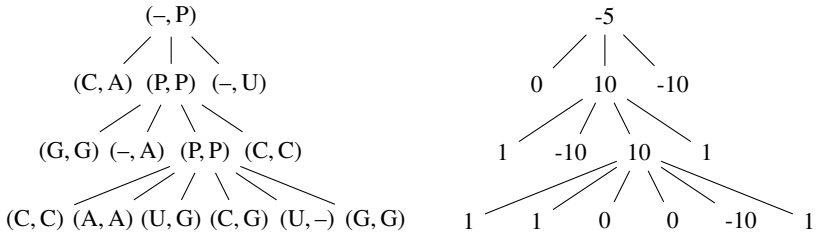


We will define the forest alignment algorithm which constructs the optimal forest alignments in the following Chapter. Assume for now that we already have such a forest alignment algorithm.

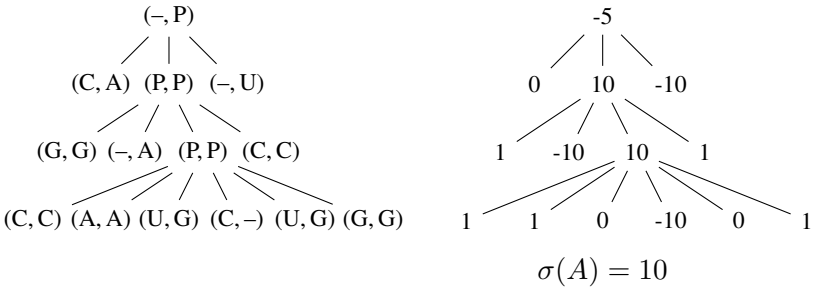
Default Scores Now let the given scores be the *RNAforester* default scoring scheme.

```
bpMatch  =  10
bpIndel  = -5
bMatch   =  1
bRep     = 0
bIndel   = -10
```

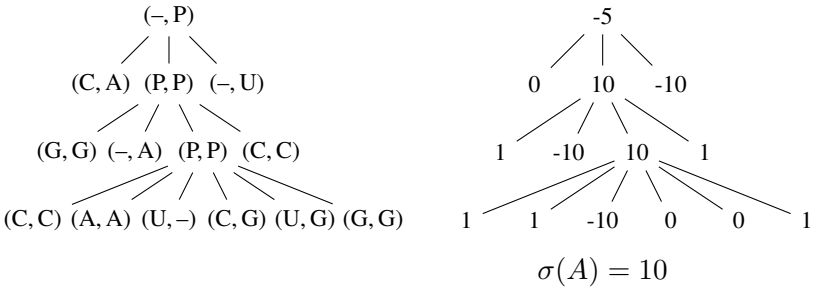
Several co-optimal solutions may exist, which all have the same optimal score, but the underlying alignments are different. Under the above default scoring scheme, three optimal solutions exist for the given input, all having the optimal score of 10. The three optimal alignments that give rise to this score are



$\sigma(A) = 10$



$\sigma(A) = 10$

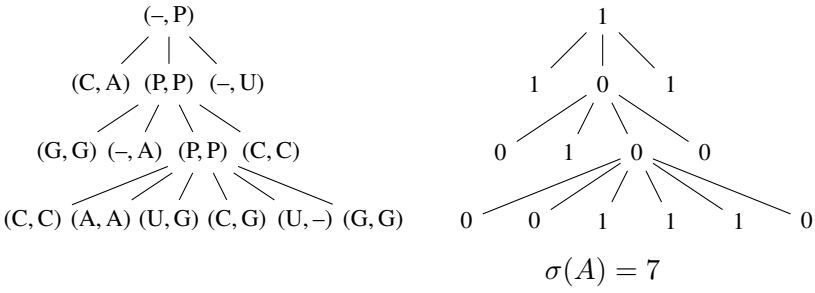


$\sigma(A) = 10$

Unit costs Of course, we cannot only compute a similarity score, but also a distance. We minimize over the costs of the edit operations. The computation can for example be done with the unit cost scheme.

```
bpMatch  =  0
bpIndel  = 1
bMatch   =  0
bRep     = 1
bIndel   = 1
```

Minimizing the cost, the algorithm creates several co-optimal alignments again. The scoring is performed according to the unit cost scheme. This is the scoring for one of the optimal alignments:

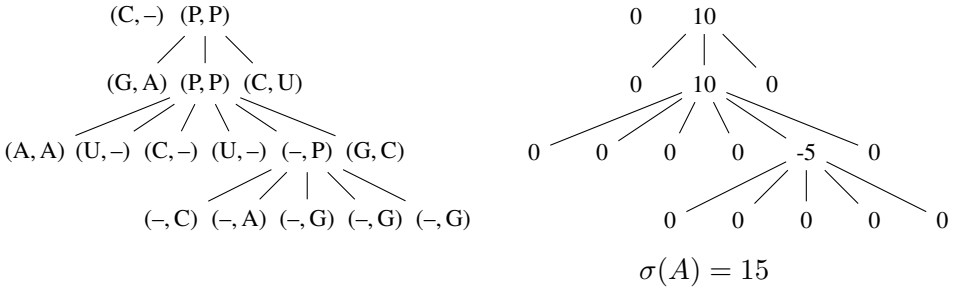


$\sigma(A) = 7$

Pure structural alignment Setting the cost for all base operations to zero, we get a purely structural alignment.

```
bpMatch  =  10
bpIndel  = -5
bMatch   =  0
bRep     = 0
bIndel   = 0
```

As deleting bases costs nothing, the algorithm will delete and insert whole substructures until before the next basepair, to delay the costly insertion/deletion of these basepairs as long as possible. This behavior is usually not desired, as it elongates the resulting alignment in an unnecessary way.



$\sigma(A) = 15$

This behavior can of course be avoided by adding a small benefit for base matches/replacements, while maintaining high costs for the pair operations (as in the default scores). These scores still ignore the sequence of bases, but at the same time do not favor unnecessary insertions and deletions of single bases. However, if the substructure is for example a very small hairpin, and the benefit from matching/replacing the bases inside does not outweigh the costs of deleting the bases that enclose it, the algorithm will still delete the whole substructure.

Discussion All the above cost and score models can be combined with the alignment distance algorithms on trees and forests. As also common in sequence alignments, costs are minimized and a score is maximized. In the following, we will use scores for the alignment distance algorithm, because in the same way as in sequence comparison, they can be used to compute both global and local forest alignments, whereas distances can be used to compute global alignments only.

5 Computing the forest alignment

In the previous chapter, we have defined the forest alignment problem, we have defined the alignments we want to construct, and we have discussed how to score them. Now we will study the algorithms that are able to solve the forest alignment problem by constructing the search space of alignments and finding an optimal solution via maximizing the score.

Based on the tree alignment distance by Jiang et al. in [55], Hoechsmann designed alignment distance algorithms that work on forests [42].

The forest alignment problem can be naturally solved in a Dynamic Programming (DP) fashion, tabulating intermediate results of forest alignment subproblems and reusing them for the computation of the overall solution. Each subproblem is computed only once, which considerably reduces the computation time for subproblems that occur multiple times, as common in the forest alignment problem. This way of efficiently solving the alignment problem is known from sequence alignment (e.g. [107]).

The purely forest based notation simplifies the presentation of the algorithm compared to the tree alignment, and the usage of a score rather than a cost function facilitates the computation of both global and local alignments.

We explain the alignment algorithms for forests, first for the global alignment problem, then we discuss the other known ones for local and small-in-large alignment. These algorithms can be used to compare two structures. We then discuss how to extend them for an alignment of multiple structures.

In chapter 6, we will be ready to extend the gap cost model of all these algorithm variants, to incorporate affine gap costs. This is a main contribution of my PhD project work.

The following terms will be used to describe the search space of the forest alignment problem. In the following, F and G are forests.

$i : F$ prefix of length i of forest F = first i trees of forest F

$F : j$ suffix of length j of forest F = last j trees of forest F

${}_i]F[_j$ forest F without prefix $i : F$ and suffix $F : j$ (subword)

$\circ]F[\circ$ forest F without leftmost and rightmost tree = ${}_1]F[_1$

$F[i]$ node by index, numbered in preorder traversal of forest F

$F_\perp$ root of the first tree of F

$F_\downarrow$ forest of children of $F_\perp$

$F_\rightarrow$ forest of right siblings of $F_\perp$

CSF of F closed subforest, consecutive sequence of sibling trees in forest F

$maxCSFlen_F(i)$ is the length of the maximum closed subforest at node index i in F

$noc_F(i)$ number of children of node at index i in forest F

$rb_F(i)$ rightbrother of node at index i in forest F

$rmbi_F(i)$ rightmost brother index of node at index i in forest F

$lb_F(i)$ label of node at index i in forest F

$len(F)$ length of a forest F

5.1 The search space of alignments for general forests

We introduce the forest alignment algorithm on general forests for simplicity. As a consequence, we will have only three edit operations instead of five for the RNA case as presented in 4.3 as no P-nodes are involved in the edit operations.

To solve the forest alignment problem by finding the alignment with the optimal score, all possible candidate alignments have to be constructed in order to be evaluated efficiently in a dynamic programming fashion using Bellman's principle of optimality. These candidate alignments constitute the *search space of forest alignments*.

Starting from the definition of the forest alignment in Section 4.2.1, we enumerate all possible alignments of two forests F and G according to the following case distinction:

Let A be an alignment of forests F and G. If F or G are empty, the alignment is also empty, or consists of insertions only, or of deletions only.

If the alignment is not empty, we distinguish three cases for structural recursion on the output alignment forest.

The *root of the first tree of the alignment forest A*, $A_\perp$, is one of the three basic edit operations:

- $A_\perp = (a, b)$
  - a is the label of $F_\perp$, and b is the label of $G_\perp$.
    - the forest of children of the alignment forest, $A_\downarrow$, is an alignment of the forest of children of F and the forest of children of G.
    - the forest of right sibling trees of the alignment forest, $A_\rightarrow$, is an alignment of the forest of right sibling trees of F, $F_\rightarrow$ and the forest of right sibling trees of G, $G_\rightarrow$.
- $A_\perp = (a, -)$
  - a is the label of $F_\perp$.
  - for some $r \in [0, ..., |G|]$
    - the forest of children of the alignment forest, $A_\downarrow$, is an alignment of $F_\downarrow$ and the prefix of length r of G.
    - the forest $A_\rightarrow$, is an alignment of $F_\rightarrow$ and the rest of G (the suffix of length $|G| - r$).
- $A_\perp = (-, b)$
  - b is the label of $G_\perp$.

- for some $r \in [0, ..., |F|]$
  - the forest of children of the alignment forest, $A_{\downarrow}$, is an alignment of the prefix of length r of F and $G_{\downarrow}$.
  - the forest $A_{\rightarrow}$, is an alignment of the rest of F (the suffix of length $|F| - r$) and $G_{\rightarrow}$.

With this case distinction over the first root of the alignment, we can recursively enumerate the complete search space by recursive application of the case distinction to compute the remaining subproblems of computing the $A_{\downarrow}$ and $A_{\rightarrow}$ subproblems.

The case distinction and the range for split index r determine the set of all possible alignments, the search space of our optimization problem.

5.1.1 Recurrences for search space construction

From the systematic enumeration of the alignments in the search space, we derive abstract recurrences to construct the search space using the same structural recursive pattern. Figure 5.1 visualizes the recurrences. As we see, there are three possible edit operations, and two recursive alignment computations for each edit oration. We have three edit operations for the alignment of general forests, where no P-nodes have to be treated in a special way, like in the alignment of RNA forests. The alignment of RNA forests will be treated in Section 5.4 and Section 5.6.

We specify the procedure without the use of Dynamic Programming. The Dynamic Programming recurrences in classical form require a tabulation scheme, which will be discussed later in this chapter.

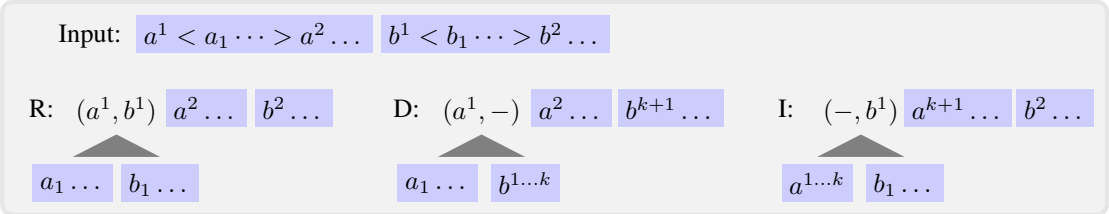


Figure 5.1: Case distinction for the forest alignment algorithm. We use a new notation to explicitly describe the forests that have to be aligned as subproblems. Here, $a^1 < a_1 \ldots > a^2$ denotes a forest whose first tree has root label a^1 and subtrees $a_1 \ldots$. On top are the two forests that we align, underneath are three cases corresponding to the edit operations.

5.2 Scoring

The scoring can be performed according to the same structural recursion.

The scores for the basic edit operations are given to the algorithm via a scoring scheme, and each edit operation that is needed to transform one input tree into the other contributes its score. During the recursive computation, scores for each of the three edit operations are added to the

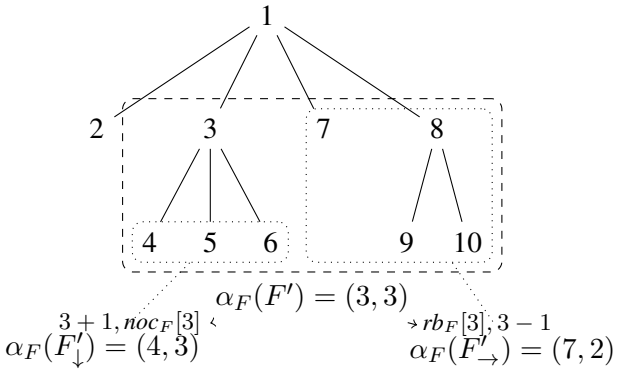


Figure 5.2: The three closed subforests are three exemplary substructures that occur in the solution of the forest alignment problem. They are shown with their index representations. The dashed closed subforest F' is aligned by aligning the first root node (node 3), and recursively aligning the children forest $F'_{\downarrow}$ (dotted) and the rightbrother forest $F'_{\rightarrow}$ (dotted). Figure modified from [42].

scores of the appropriate aligned subforests, computing an overall score for each alignment. By maximization, we optimize over these scores.

Because our scoring function is additive, Bellman's principle of optimality [13] can be applied. For a dynamic programming approach, we have to tabulate intermediate results.

5.3 Global alignment of general forests

5.3.1 Defining the ranges of recursive computation: Relevant Closed Subforests

Closed subforests

Based on the notion of *closed subforests (CSFs)* (see Figure 5.2), we calculate forest alignments in a structurally recursive way that adequately captures the recursive subproblem. The following definitions are adapted from [42].

Recall the definition of a closed subforest of F from Definition 2.1.8. To align two complete forests F and G, we start the recursive alignment algorithm with aligning the maximal closed subforests of F and G. To compare two CSFs F' and G', we recursively compare the closed subforests of the children $F'_{\downarrow}$ and $G'_{\downarrow}$, and the closed subforests of the right siblings $F'_{\rightarrow}$ and $G'_{\rightarrow}$.

As subproblems of the algorithm, we compare all *relevant* closed subforest pairs of F and G. These are all pairs of closed subforests which have to be aligned in the recursive computation to get the overall alignment of input forests F and G. The relevant CSF pairs depend on the algorithm variant, and for global alignments less CSFs are compared than in the local case.

Relevant Closed Subforest Pairs for global forest alignment

The basic variant of the forest alignment algorithm is the *global forest alignment*. To compute a global forest alignment, the complete input forests are aligned to each other, in terms of closed subforests that is an alignment of $CSF_F(i, maxCSFlen_F[i])$ and $CSF_G(k, maxCSFlen_G[k])$.

For the general forest alignment problem, the following transitions are made in the recursive alignment process of aligning forests F and G:

	Children	Right Siblings
Replacement	$RC = (F_\downarrow, G_\downarrow)$	$RR = (F_\rightarrow, G_\rightarrow)$
Deletion	$DC = (F_\downarrow, r : G)$	$DR = (F_\rightarrow, G : r)$
Insertion	$IC = (F : r, G_\rightarrow)$	$IR = (r : F, G_\downarrow)$

Table 5.1: Transitions between the closed subforests pairs are compared recursively in the general global forest alignment algorithm.

Starting from the global alignment problem of aligning the CSF pair

$$(CSF_F(i, maxCSFlen_F[i]), CSF_G(k, maxCSFlen_G[k])),$$

and performing all possible transitions to alignment subproblems that we get from the edit operations as stated in the table, we see that the CSF pairs that are aligned in the recursive step all have one of two distinctive forms. We call them the *relevant subforests* for the global alignment problem. A closed subforest is *maximal* if it cannot be extended to the left and to the right, and we indicate a maximal closed subforest of forest F by $\overline{F}$.

Lemma 1. *The* relevant CSFs for the global forest alignment *have either the form* $({}_a]\overline{F}[_0, {}_c]\overline{G}[_d)$ *or the form* $({}_a]\overline{F}[_b, {}_a]\overline{G}[_0$.

The closed subforest of form ${}_a]\overline{F}[_0$ is the subforest of F with omitting the first a trees at the front, and omitting 0 trees at the end. It could also be written as prefix of $\overline{F}$ with length $b = len(\overline{F}) - a$, but we need this more general notation when to discuss the relevant closed subforest pairs for related problems below.

The idea of the proof of the above Lemma is lined out in Figure 5.3 by application of all transitions to subproblems. We abbreviate the transitions at the arrows in the figure as introduced in Table 5.1. Compare to Hoechsmann in [42], a proof is also given there.

5.3.2 Stepping down in the recursion: Aligning a pair of CSFs

To solve the global forest alignment problem in a dynamic programming fashion, we first have to know what the subproblems are.

After finding out how do recurse to the relevant subproblems, we now have to find out how to solve a subproblem, i.e. how to compare a pair of CSFs. With this knowledge, we can construct the search space of possible solutions, and these solutions are scored to search for the optimal score.

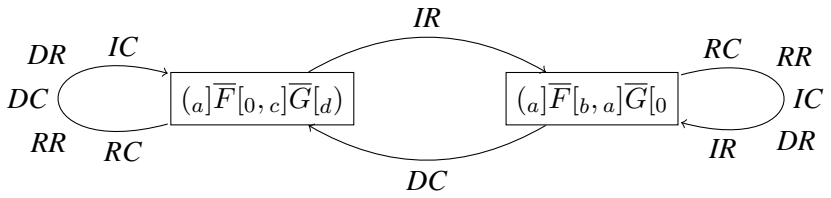


Figure 5.3: Relevant pairs of CSFs for the global forest alignment. Next to the arrows are possible transitions, which, following the arrow, result in the CSF pair of the form they point to. Following all arrows and applying all possible transitions from the overall global alignment problem of aligning the CSF pair $(\mathit{CSF}_F(i, \mathit{maxCSFlen}_F[i]), \mathit{CSF}_G(k, \mathit{maxCSFlen}_G[k]))$, we always end up in one of the two cases for the subproblems. Thus, all CSF pairs that have to be aligned as subproblems have one of the two forms. This figure is adapted from [42] with different terminology. See also Section 5.8.

To construct a solution in the Dynamic Programming approach, we tabulate the results for each subproblem, in our case we will collect the scores for comparing each relevant CSF pair in a table.

Depending on whether we align RNA or general forests, we perform different edit operations when aligning a pair of CSFs.

Alignment of a general CSF Pair

To align a CSF pair, we compute all possible edit operations for each node in the alignment forest, and recursively solve the remaining subproblems. Insertions and deletions insert a gap character "–" in the resulting alignment.

Let us assume that we have a given scoring scheme with scores for $match$, rep, $indel$, and recall the definitions of the recursive subproblems *RC*, *RR*, *DC*, *DR*, *IC* and *IR* in Section 5.3.1. Of course, it would also be possible to score insertions and deletions separately, instead of with the same indel score. The edit operations and their scores for aligning the closed subforests (i, j) of F and (k, l) of G are the following:

- A *replacement*, which produces the node $(\mathit{lb}_F(i), \mathit{lb}_G(k))$ in the resulting alignment forest, and is scored as

$$\mathit{replacement}(i,j,k,l) = \begin{cases} \mathit{match} + \mathit{score}(\mathit{RC}) + \mathit{score}(\mathit{RR}) & \text{if } \mathit{lb}_F(i) = \mathit{lb}_G(k) \\ \mathit{rep} + \mathit{score}(\mathit{RC}) + \mathit{score}(\mathit{RR}) & \text{otherwise .} \end{cases}$$

- A *deletion*, which produces the node $(\mathit{lb}_F(i), -)$ in the resulting alignment forest, and is scored as

$$\mathit{deletion}(i,j,k,l) = \mathit{indel} + \mathit{score}(\mathit{DC}) + \mathit{score}(\mathit{DR})$$

- An *insertion*, which produces the node $(-, \mathit{lb}_G(k))$ in the resulting alignment forest, and is scored as

$$\mathit{insertion}(i,j,k,l) = \mathit{indel} + \mathit{score}(\mathit{IC}) + \mathit{score}(\mathit{IR})$$

The subproblems for the general forest alignment have been defined in Table 5.1. Using the recursively computed score of the subproblems, we are able to score the edit operations and maximize over the scores by the scoring scheme.

5.3.3 Sketching a control structure

With the knowledge about the form of the relevant closed subforest pairs for the global alignment algorithm and how to align them, we now construct a loop structure that compares only the pairs of closed subforests that are involved in the construction of the overall solution, to keep the search space as small as possible.

The small search space is due to the nature of the global alignment problem, which explicitly requires a complete alignment of the input structures (or $(\mathit{CSF}_F(i, \mathit{maxCSFlen}_F[i]), \mathit{CSF}_G(k, \mathit{maxCSFlen}_G[k]))$ in closed subforest notation), and not an alignment of arbitrary closed subforests as in the local case. By our analysis of the relevant CSF pairs for the global alignment problem, we made sure that all required subproblems are solved. So although we construct a smaller search space than that of all possible (possibly local) alignments, no solutions are lost considering the global alignment problem.

Recall that we write CSF pairs as $(i, j), (k, l)$, where (i, j) is a CSF of F with node index i and length index j, and (k, l) is a CSF of G with node index k and length index l. The length of the maximal closed subforest starting at node index i is given by $\mathit{maxCSFlen}_F[i]$.

Base case, first row and column The base case of the forest alignment procedure is the empty alignment. In terms of closed subforests, this is any alignment of two CSFs that both have length 0, no matter what the node indices are. The result of the alignment of two CSFs of length 0, the empty alignment, is scored with 0, as known from scores in sequence comparison.

Some cases of CSF pair comparisons are the same in all algorithm variants that we will introduce in this chapter. These are the deletion of a complete CSF, and the insertion of a complete subforest. For global, local, and small in large forest alignment, all of these comparisons are relevant, as a whole subforest may be inserted or deleted in all three algorithm variants.

The subword notation from 1 is transferred to the node and length index notation of CSFs via the $maxCSFlen$ function.

See Figure 5.4 for the computation in pseudocode.

The problem of aligning CSF pairs of arbitrary lengths > 0 will be treated differently for the global, local and small-in-large forest alignment problem.

A general global forest alignment Using the computation for the first row and first column, and also using the knowledge about the relevant CSF pairs for the global alignment problem, we construct a loop structure for the global forest alignment. The subword notation from Lemma 1 is again transferred to the node and length index notation of CSFs via the $maxCSFlen$ function.

The pseudocode for this algorithm sketch is shown in Figure 5.5.

For a solution in a DP way, we now tabulate the intermediate results.

Input: Forests F and G
Output: Empty alignment, all deletions of complete CSFs, insertions of complete CSFs

```
1:  function FIRSTCOLANDROW(F, G)
2:      ALIGNCSFPAIR((_, 0), (_, 0))                    ▷ align two empty CSFs
3:      for all i ∈ {1, ..., |F|} do                    ▷ complete CSF deletion
4:          for all j ∈ {1, ..., maxCSFlen_F[i]} do
5:              ALIGNCSFPAIR((i, j), (_, 0))
6:          end for
7:      end for
8:      for all k ∈ {1, ..., |G|} do                    ▷ complete CSF insertion
9:          for all l ∈ {1, ..., maxCSFlen_G[k]} do
10:             ALIGNCSFPAIR((_, 0), (k, l))
11:         end for
12:     end for
13: end function
```

Figure 5.4: Compute the alignment of two empty CSFs, all deletions of complete CSFs and insertions of complete CSFs. All pairs of CSFs that both have length 0, regardless of their node indices – indicated by the _, result in an empty alignment. Therefore we can map all of them to the empty alignment. The same applies if only one CSF has length 0: for all node indices it also results in the same alignment. This property can be exploited in the tabulation of the results which will be discussed in Section 5.3.4.

5.3.4 Tabulation

Tabulation is an important component of Dynamic Programming, because the intermediate results kept in a table can be reused for the computation of the final result.

The comparison of two closed subforests F' and G', of input forests F and G, as done in our algorithm, requires four indices (i, j, k, l):

1. a node index i for the start of CSF F' in input forest F
2. a length index j for the length of CSF F' in F
3. a node index k for the start of CSF G' in input forest G
4. a length index l for the length of CSF G' in G

This representation of subproblems allows for straightforward tabulation, but Hoechsmann observed that the resulting four dimensional matrix is sparse and the entries can be mapped to a two dimensional matrix.

This is done via the function β, exploiting the ambiguous representation of the pairs with empty closed subforests (i.e. length index $= 0$) as noted in Figure 5.4.

```
Input: Forests F and G
Output: Alignment of CSF (i, maxCSFlen_F[i]) and CSF (k, maxCSFlen_G[k]) and relevant subproblems
 1: function GLOBALFORESTALIGNMENT(F, G)
 2:     FIRSTCOLANDROW(F, G)
 3:     for all i ∈ {1, ..., |F|} do                          ▷ remaining cells
 4:         for all k ∈ {1, ..., |G|} do
 5:             l ← maxCSFlen_G[k]
 6:             for all j ∈ {1, ..., maxCSFlen_F[i]} do
 7:                 ALIGNCSFPAIR((i,j),(k,l))
 8:             end for
 9:             j ← maxCSFlen_G[i]
10:             for all l ∈ {1, ..., maxCSFlen_G[k]} do
11:                 ALIGNCSFPAIR((i,j),(k,l))
12:             end for
13:         end for
14:     end for
15: end function
```

Figure 5.5: Loop structure for the global forest alignment comparing only the relevant closed subforests. The calculation order will be discussed in Section 5.3.6.

5.3.5 Index mapping

To store the result of a comparison of two closed subforests space-efficiently in a table, a two stage mapping is used as proposed by Hoechsmann.

The mapping function α maps the closed subforests to their representation as pairs (i, j) of node and length index, as we have represented them in this chapter already. The index mapping function β maps four dimensional closed subforest pair indices to a two dimensional table index.

For this, an additional table *offset* is computed, which for each node index i stores the number of nonempty CSFs having a node index less than i. This trick enables us to define the mapping β:

$$\beta_F(i,j) = \begin{cases} 0 & \text{if } j = 0 \\ \mathit{offset}_F[i] + j & \text{otherwise.} \end{cases}$$

5.3.6 Calculation order

With the recurrences and the mapping function defined, we can almost write down the dynamic programming algorithm. One important aspect is still missing though: The order of calculation of the matrix entries. We have to make sure that all recursive subproblems are already computed, before we can compute a problem. Row- or column-wise calculation in the matrix does not work here, as the intermediate results are comparisons of closed subforests, which are placed in the matrix by the mapping function.

The easiest case is the entry $E_{score}(0,0) = 0$, which does not depend on other entries and can thus be computed first.

The rest of the first row also does not depend on any entries outside of this row, as it is just the result of aligning the empty forest with a nonempty CSF, that is inserting a whole forest. This forest must be nonempty because β maps all index pairs representing the empty CSFs to 0, and all nonempty ones to a positive value.

The same applies for the rest of the first column, which is the result of aligning a nonempty CSF with the empty forest, that means deleting this forest completely.

So we know now that we can fill the first entry directly, and the first row and the first column do not need any external entries. To get the computation order within the first row, within the first column and to get the dependencies and computation order in the rest of the matrix, we have to consider node index monotony as presented in [42].

Studying the recursive subproblems, we see that for each edit operation in the recursive call either the node index increases strictly, or the length index decreases. Following this observation, the matrix can be calculated in decreasing order of the node index and increasing order of the length index.

Altogether, we now have the following calculation order:

1. initialize entry $(0,0)$
2. compute first column, looping over decreasing node index and increasing length index
3. compute first row, looping over decreasing node index and increasing length index
4. compute remaining entries, looping over decreasing node index and increasing length index according to the relevant CSFs for the particular problem (see also pseudocode in 5.5, 5.10, 5.11).

With the calculation order, we can now formulate a DP algorithm for global forest alignment.

5.3.7 Dynamic programming algorithm for global forest alignment

We can now calculate the dynamic programming table

$$E_{score}(x,y) = \textit{forest alignment distance}_{score}((i,j),(k,l)).$$

First, we have to compute the first row and first column again, now in the correct order, so that the subproblems are solved first.

This algorithm (pseudocode in Figure 5.7) can be used to align general forests. For the alignment of RNA secondary structures in the extended forest representation, we have to add special edit operations for the P-nodes that represent the base pair bonds in this representation.

This is taken care of in the following algorithm.

5.4 Global alignment of RNA forests

We now construct a dynamic programming algorithm for the global alignment of RNA forests. This is done in a similar fashion as for the general forest alignment, starting with an analysis of the subproblems.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxCSFlen*
Output: Empty alignment, all deletions of complete CSFs, insertions of complete CSFs

function FIRSTCOLANDROW(F, G)
  $E_{score}(0,0) = 0$ ▷ align two empty CSFs
  for $i \leftarrow |F|, \ldots, 1$ **do** ▷ complete CSF deletion
    for $j \leftarrow 1, maxCSFlen_F[i]$ **do**
      $E_{score}(\beta_F(i,j),0) \leftarrow$DELETION($lb_F(i), E_{score}(F_\downarrow,0), E_{score}(F_\rightarrow,0)$)
    end for
  end for
  for $k \leftarrow |G|, \ldots, 1$ **do** ▷ complete CSF insertion
    for $l \leftarrow 1, maxCSFlen_G[k]$ **do**
      $E_{score}(0,\beta_G(k,l)) \leftarrow$INSERTION($lb_G(k), E_{score}(0,G_\downarrow), E_{score}(0,G_\rightarrow)$)
    end for
  end for
end function

Figure 5.6: Compute the table entries for the empty alignment, all deletions of complete CSFs and insertions of complete CSFs. The deletion and insertion functions have to be defined for either RNA or general forest alignment. The mapping function β from a pair of CSFs in index notation to the table index was discussed in Section 5.3.4. Note that the calculation order is now explicitly specified according to 5.3.6, such that all reused subproblem solutions are already computed when they are used.

Relevant CSF pairs for global alignment of RNA

The above Lemma 1 holds for a general global forest alignment algorithm, which is not specialized for RNA structure comparison. If we want to produce well-formed RNA alignments (Section 4.2.2), we have to treat the P-nodes in a special way. According to the definition of a well-formed RNA forest alignment, when matching two P-nodes, their leftmost and both rightmost children have to be matched or replaced. The table of possible transitions is updated:

	Children	Right Siblings
Replacement	$RC = ({}_1]F_\downarrow[{}_1, {}_1]G_\downarrow|_1)$	$RR = (F_\rightarrow, G_\rightarrow)$
Deletion	$DC = (F_\downarrow, r : G)$	$DR = (F_\rightarrow, G : r)$
Insertion	$IC = (F : r, G_\rightarrow)$	$IR = (r : F, G_\downarrow)$

Table 5.2: Transitions between the pairs of closed subforests that have to be compared recursively in the global forest alignment algorithm for RNA. Most transitions are the same as for the general global alignment algorithm in Table 5.1, only the recursion over the children forest in the replacement case is different, as the pairing bases of the P-nodes are not part of the recursion.

The new replacement operation requires a recursive comparison of closed subforests of additional new types for the alignment of the children forests.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: Alignment of CSF $(i, \mathit{maxCSFlen}_F[i])$ and CSF $(k, \mathit{maxCSFlen}_G[k])$ and relevant subproblems

```
1: function GLOBALFORESTALIGNMENT(F, G)
2:     FIRSTCOLANDROW(F, G)
3:     for i ← |F|, 1 do                                   ▷ remaining cells
4:         for k ← |G|, 1 do
5:             l ← maxCSFlen_G[k]
6:             for j ← 1, maxCSFlen_F[i] do
7:                 E_score(β(i,j), β(k,l)) ←ALIGNCSFPAIR((i,j),(k,l))
8:             end for
9:             j ← maxCSFlen_G[i]
10:            for l ← 1, maxCSFlen_G[k] do
11:                E_score(β(i,j), β(k,l)) ←ALIGNCSFPAIR((i,j),(k,l))
12:            end for
13:        end for
14:    end for
15: end function
```

Figure 5.7: Compute the global forest alignment matrix. Note that the calculation order is now explicitly specified according to 5.3.6, such that all reused subproblem solutions are already computed when they are used.

Lemma 2. *The relevant CSFs for the global alignment problem of RNA forests have the form* $({}_a]\overline{F}[_0, {}_c]\overline{G}[_d)$, $({}_a]\overline{F}[_b, {}_c]\overline{G}[_0)$, $({}_a]\overline{F}[_1, {}_c]\overline{G}[_d)$ *and* $({}_a]\overline{F}[_b, {}_c]\overline{G}[_1)$.

Similar to in the general case, we sketch a proof for the lemma in Figure 5.8. We apply all possible transitions to the goal of the global alignment problem of aligning the complete forests, which is an alignment of $(\mathit{CSF}_F(i, \mathit{maxCSFlen}_F[i]), \mathit{CSF}_G(k, \mathit{maxCSFlen}_G[k]))$. By applying all possible transitions from the above table, we always reach CSFs of the form indicated in Lemma 2.

Deriving a global RNA forest alignment algorithm idea Again, we can construct a control structure from the knowledge about the relevant CSF pairs. The algorithm sketch for the global forest alignment problem for RNA is shown in Figure 5.10.

The expression ${}_a]\overline{F}[_1$ is derived from the fact that we applied the ${}_1|\quad|_1$ operation to ${}_a]\overline{F}[_0$ where a is any start position in F already. That means the expression only has to be computed if $a > 0$, otherwise it is a closed subforest of length 0, and was already aligned to all other CSFs before.

The subword notation from Lemma 2, which describes a CSF as a continuous subsequence $a]\quad[b$ of a forest, is transferred to the node and length index pair notation of CSFs via the $maxCSFlen$ function, similar as done before for Lemma 1. The two additional forms of CSFs

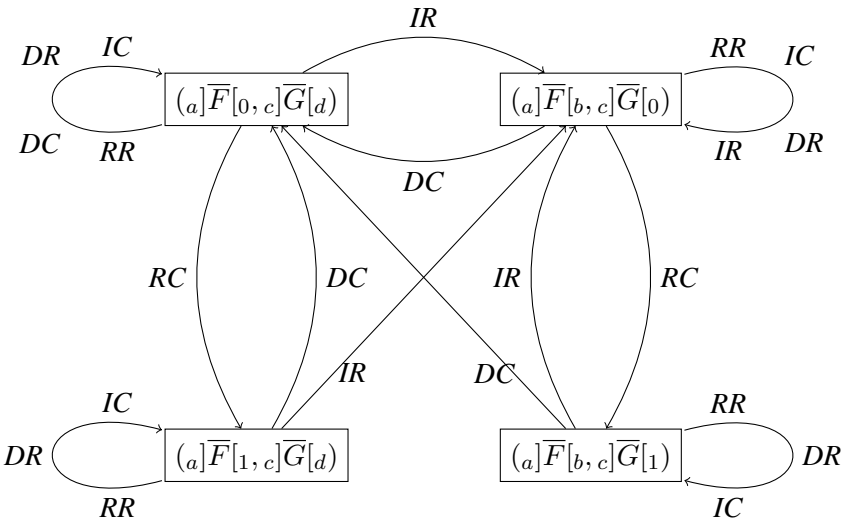


Figure 5.8: Relevant pairs of CSFs for the global forest alignment of RNA secondary structures, if well-formed RNA forest alignments are produced. The recursion over the child forest for the replacement case leads to two additional forms of CSF pairs. Compare also with Figure 5.3.

in Lemma 2 give rise to two extra cases of CSF comparisons for pairs $(i, j), (k, l-1)$ and $(i, j-1), (k, l)$ in the pseudocode in Figure 5.10.

Alignment of a pair of CSFs of RNA forests

For aligning a CSF pair of RNA structures, we use an additional edit operation for the replacement of P-nodes, called the pair replacement edit operation.

Let us assume that we have a given scoring scheme with scores for *base_match*, *pair_match*, *base_rep*, *base_indel*, *pair_indel*, and recall the definitions of the recursive subproblems *RC*, *RR*, *DC*, *DR*, *IC* and *IR* in Section 5.4. The edit operations and their scores for aligning the closed subforests (i, j) of F and (k, l) of G are the following:

- A *pair replacement*, which is applied only to two P-nodes, so $lb_F(i) = P$, and $lb_G(k) = P$. It produces the node $(lb_F(i), lb_G(k))$, its leftmost child node $(lb_F(i+1), lb_G(k+1))$, and its rightmost child node $(lb_F(rmbi_F(i+1)), lb_G(rmbi_G(k+1)))$ in the resulting alignment forest, and is scored as

$$\begin{aligned} pair\_replacement(i,j,k,l) &= pair\_match \\ &+ score\_l + score(RC) \\ &+ score\_r + score(RR), \end{aligned}$$

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: RNA structure alignment of CSF $(i, \mathit{maxCSFlen}_F[i])$ and CSF $(k, \mathit{maxCSFlen}_G[k])$ and relevant subproblems

```
1: function GLOBALFORESTALIGNMENT(F, G)
2:     FIRSTCOLANDROW(F, G)
3:     for all i ∈ {1, |F|} do                                   ▷ remaining cells
4:         for all k ∈ {1, |G|} do
5:             l ← maxCSFlen_G[k]
6:             for all j ∈ {1, maxCSFlen_F[i]} do
7:                 E_score(β(i,j), β(k,l)) ←ALIGNCSFPAIR((i,j),(k,l))
8:                 if l>1 then
9:                     E_score(β(i,j), β(k,l-1)) ←ALIGNCSFPAIR((i,j),(k,l-1))
10:                end if
11:            end for
12:            j ← maxCSFlen_G[i]
13:            for all l ∈ {1, maxCSFlen_G[k]} do
14:                E_score(β(i,j), β(k,l)) ←ALIGNCSFPAIR((i,j),(k,l))
15:                if j>1 then
16:                    E_score(β(i,j-1), β(k,l)) ←ALIGNCSFPAIR((i,j-1),(k,l))
17:                end if
18:            end for
19:        end for
20:    end for
21: end function
```

Figure 5.9: General loop structure for the global forest alignment of RNA forests to compute only the relevant subproblems.

where

$$\mathit{score\_l} = \begin{cases} \mathit{base\_match} & \text{if } \mathit{lb}_F(i+1) = \mathit{lb}_G(k+1) \\ \mathit{base\_rep} & \text{otherwise,} \end{cases}$$

$$\mathit{score\_r} = \begin{cases} \mathit{base\_match} & \text{if } \mathit{lb}_F(\mathit{rmbi}_F(i+1)) = \mathit{lb}_G(\mathit{rmbi}_G(k+1)) \\ \mathit{base\_rep} & \text{otherwise.} \end{cases}$$

- A *replacement*, which cannot be applied to P-nodes, so $\mathit{lb}_F(i) \neq P$, and $\mathit{lb}_G(k) \neq P$. It produces the node $(\mathit{lb}_F(i), \mathit{lb}_G(k))$ in the resulting alignment forest, and is scored as

$$\mathit{replacement}(i,j,k,l) = \begin{cases} \mathit{base\_match} + \mathit{score}(RR) & \text{if } \mathit{lb}_F(i) = \mathit{lb}_G(k) \\ \mathit{base\_rep} + \mathit{score}(RR) & \text{otherwise .} \end{cases}$$

  There is no recursive subproblem for the children forests, because the aligned nodes are always leaf nodes according to the definition of the extended forest representation.

- A *deletion*, which produces the node $(lb_F(i), -)$ in the resulting alignment forest, and is scored as

$$\mathit{deletion}(i,j,k,l) = \begin{cases} \mathit{pair\_indel} + \mathit{score}(DC) + \mathit{score}(DR) & \text{if } lb_F(i) = \text{P} \\ \mathit{base\_indel} + \mathit{score}(DC) + \mathit{score}(DR) & \text{otherwise .} \end{cases}$$

- An insertion, which produces the node $(-, lb_G(k))$ in the resulting alignment forest, and is scored as

$$\mathit{insertion}(i,j,k,l) = \begin{cases} \mathit{pair\_indel} + \mathit{score}(IC) + \mathit{score}(IR) & \text{if } lb_G(k) = \text{P} \\ \mathit{base\_indel} + \mathit{score}(IC) + \mathit{score}(IR) & \text{otherwise .} \end{cases}$$

The subproblems for the RNA forest alignment have been defined in Table 5.2. Tabulation order and calculation can be done in the same fashion as for the global alignment of general forests in Section 5.3. With this knowledge, we can already solve the presented alignment problems.

5.4.1 Dynamic programming algorithm for global alignment of RNA forests

For now, we can compute global alignments of general forests and RNA forests.

Our choice of computing a similarity score rather than a distance enables us to compute local forest alignments as well, transferring the local alignment concept from sequence alignment to forest alignment.

We will therefore now develop an algorithm for local alignment of general forests.

5.5 Local alignment of general forests

The local optimal forest alignment problem asks for a local alignment of the input forests with an optimal score. Again, we start to construct the new algorithm by analyzing the recursive subproblems.

5.5.1 Relevant CSFs for local alignment

In terms of closed subforests, a local forest alignment is any alignment of two closed subforests of the input forests F and G.

Therefore, the relevant CSF pairs for the local alignment problem are all possible combinations of CSFs. This includes both the general and the RNA local alignment problems.

The algorithm sketch for the local forest alignment problem therefore has to loop over all possible combinations of CSFs and is depicted in Figure 5.11.

The loop structure is different from the global alignment algorithms which were defined above, regarding the nesting of the loops. Nevertheless, the running indices have to run over the node and length indices of the relevant CSFs as before. With the same argument as for the global alignment, the computation order has to be by increasing the node index and decreasing the length index, as the transitions for the edit operations when aligning a CSF pair are exactly the same for global and local alignment.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: RNA structure alignment of CSF $(i, \mathit{maxCSFlen}_F[i])$ and CSF $(k, \mathit{maxCSFlen}_G[k])$ and relevant subproblems

```
1:  function GLOBALFORESTALIGNMENT(F, G)
2:      FIRSTCOLANDROW(F, G)
3:      for i ← |F|, 1 do                                          ▷ remaining cells
4:          for k ← |G|, 1 do
5:              l ← maxCSFlen_G[k]
6:              for j ← 1, maxCSFlen_F[i] do
7:                  E_score(β(i,j), β(k,l)) ←ALIGNCSFPAIR((i,j),(k,l))
8:                  if l>1 then
9:                      E_score(β(i,j), β(k,l−1)) ←ALIGNCSFPAIR((i,j),(k,l-1))
10:                 end if
11:             end for
12:             j ← maxCSFlen_G[i]
13:             for l ← 1, maxCSFlen_G[k] do
14:                 E_score(β(i,j), β(k,l)) ←ALIGNCSFPAIR((i,j),(k,l))
15:                 if j>1 then
16:                     E_score(β(i,j−1), β(k,l)) ←ALIGNCSFPAIR((i,j-1),(k,l))
17:                 end if
18:             end for
19:         end for
20:     end for
21: end function
```

Figure 5.10: Compute the global forest alignment matrix for RNA forests. Note that the calculation order is now explicitly specified according to 5.3.6, such that all reused subproblem solutions are already computed when they are used.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: All alignments of pairs of CSFs (F', G'), where F' is CSF of F and G' is CSF of G

```
1:  function LocalForestAlignment(F, G)
2:      FirstColAndRow(F, G)
3:      for all i ∈ {1, ..., |F|} do                          ▷ remaining cells
4:          for all k ∈ {1, ..., |G|} do
5:              for all j ∈ {1, ..., maxCSFlen_F[i]} do
6:                  for all l ∈ {1, ..., maxCSFlen_G[k]} do
7:                      E_score(β(i, j − 1), β(k, l)) ←alignCSFPair((i, j), (k, l))
8:                  end for
9:              end for
10:         end for
11:     end for
12: end function
```

Figure 5.11: Compute the local forest alignment subproblems for general or RNA forests.

5.5.2 Dynamic Programming algorithm for local alignment of general forests

With the knowledge about the loop structure, tabulation and calculation order for the local alignment algorithm, we can now formulate a dynamic programming algorithm in Figure 5.12.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: All alignments of pairs of CSFs (F', G'), where F' is CSF of F and G' is CSF of G

```
1:  function LocalForestAlignment(F, G)
2:      FirstColAndRow(F, G)
3:      for i ← |F|, 1 do                                          ▷ remaining cells
4:          for k ← |G|, 1 do
5:              for j ← 1, maxCSFlen_F[i] do
6:                  for l ← 1, maxCSFlen_G[k] do
7:                      E_score(β(i,j), β(k,l)) ←alignCSFPair((i,j),(k,l))
8:                  end for
9:              end for
10:         end for
11:     end for
12: end function
```

Figure 5.12: Compute the local forest alignment matrix for general forests. The calculation order is now explicitly specified, such that all reused subproblem solutions are already computed when they are used.

5.6 Local alignment of RNA forests

The local alignment algorithm for RNA forests can be constructed by a new combination of the dynamic programming algorithm ingredients we have already used in this Chapter.

The overall control structure, tabulation and order of calculation is the same as for the local alignment of general forests, because we also have to align all possible combinations of CSFs to get a local alignment.

To ensure the correct computation for RNA forests, a custom variant of alignCSFPair is used, which accounts for extra edit operations for the P-nodes. This is the same as already used in the global case to align a CSF pair of RNA forests.

5.6.1 Local suboptimal solutions

When computing the local similarity, more than one region might be similar, or we may search for a structural motif that appears multiple times in another structure. In this case, all solutions of the local alignment algorithm are worth examining, or at least the ones above a certain threshold. Our implementation is able to compute local suboptimal solutions in a special program mode. In the solution, it excludes intersecting suboptimal solutions, i.e. solutions that share one or more

nodes, to avoid local suboptimal results that are too similar. In the computation, *RNAforester* stores all these local similar CSFs, and reports the ones within a given amount of percent of the score, starting from the optimal score.

5.7 Small in large alignment of forests

Studying the nature of local and global forest alignment, the small-in-large alignment can be performed with a slight variation of the algorithm. If a smaller forest should be matched within a larger one, the smaller forest can be aligned in an arbitrary position within the larger one. So an arbitrary CSF of the larger forest (as in a local alignment) must be compared to the complete smaller forest (as in a global alignment). The optimal loop structure is thus a combination of the global and local forest alignment algorithms - the loop for the smaller forest is like for local alignment, and the loop for the larger forest is like for the global alignment.

5.8 Discussion: Role of relevant CSF pairs; edit operations for RNA

The relevant pairs of CSFs that have to be compared as subproblems of an alignment problem constitute the search space of the algorithm that solves the problem. The set of relevant forests depends on the particular alignment problem, e.g. aligning general forests or RNA forests, globally or locally.

The particular forest algorithm variant has to be tailored to solve the according relevant set of subproblems by adapting the recurrences.

Relevant CSFs for global alignment of general forests Hoechsmann states that the following CSF pairs are relevant for the global forest alignment problem.

> **Lemma 3.** *Let $\overline{F}$ and $\overline{G}$ be maximal CSFs of F and G, respectively. The pairs of subforests that are relevant for the calculation of $\sigma_{FA}(F, G)$ due to Figure 5.3 have the form $(\overline{F} : j, {}_l]\overline{G}[_k)$ and $({}_j]\overline{F}[_i, \overline{G} : l)$.*

As opposed to Hoechsmann, we use a different terminology for prefixes of length i of forests in Lemma 1 and the figure that sketches the proof (Figure 5.3), and write them down as subwords omitting i nodes at the front, and 0 nodes at the back. We also use different index variables a, b, c and d, so that the index scheme is not confused with the node and length indices for CSFs, and abbreviate the transitions.

Relevant CSFs for global alignment of RNA forests The statement in Lemma 2 differs from Hoechsmann, as in the *RC* operation the order of the suffix operation $F : j$ on a forest F and the removal of the outermost nodes $\circ|F|\circ$ (the same as ${}_1]F[_1)$ on a forest F was not performed in the correct order, and the ommitance of the leftmost node is subsumed in the suffix operation.

We therefore also suggest to remove the application of $rb_G[k]$ to get just k and $rb_F[i]$ to get just i in line 12 and 15 of the pseudocode on page 93 of [42], because otherwise, relevant subforests are missed. The backtracking that is used to retrieve the alignment from the tabulated intermediate results is not successful in these cases and no alignment can be constructed.

See Figure 5.8 for a new figure not given by Hoechsmann depicting the correspondence between the possible transitions to recursive subproblems and the forms of the pairs of closed subforests that have to be compared.

Hoechsmanns proof that the relevant closed subforest pairs for the well-formed RNA alignment algorithm always take one of four forms also has to be updated accordingly, but it can be reconstructed from our updated figure that our statement is true.

Edit operations for aligning RNA forests Note that the presented model of aligning RNA secondary structure forests along the lines of Hoechsmann, although handling P-node replacements, does not have designated edit operations for deleting or inserting P-nodes. No requirements about the deletion and insertion of P-nodes are made in the well-formed forest alignment definition according to Hoechsmann either. The implications of this model design decision, which of course also reflects in the resulting algorithms for RNA forests, will be discussed in Section 9.

5.9 Pairwise vs. multiple forest alignment

The presented algorithm variants have all been defined for pairwise alignment only. From a theoretical point of view, it is very easy to generalize the forest alignment model to multiple alignments by using tuples rather than pairs in the definition of the edit operations and the resulting alignment forest. Implementing the algorithms for tuples of arbitrary length would add a new dimension to the computation costs, but we can use the fact that the RNA alphabet is quite small to avoid this. Using a progressive profile approach, Hoechsmann generalizes the algorithms to work on multiple input structures.

The multiple forest alignment data structure With the usage of tuples rather than pairs, the forest alignment model generalizes to multiple alignments. The forests are defined on the tuple alphabet $\mathcal{A}^n = \{(a_1, \ldots, a_n) \,|a_i \in \mathcal{A} \cup -\} \backslash \{-, .., -\}$.

Defining a multiple forest alignment The definition of a multiple forest alignment is also a straightforward generalization from the alignment of two forests.

Given a function $proj$: $G_{Forest}(L) \rightarrow G_{Forest}(L \setminus \{-\})$. A forest $F_{ali} \in \mathcal{F}(L^n \setminus \{-\}^n)$ is an alignment of forests $F, ...F_n \in \mathcal{F}(L \setminus \{-\})$ if and only if the following holds: $F_i = \pi(proj(F_{ali}|i))$ for $i \in [1, n]$.

The contraction function π is defined according to the definition for trees in Section 3.5.2 which we have extended for forests in Section 4.2.2. The symbol "$-$" is the gap symbol, and the label $\{-\}^n$ excluded from the alignment tuple alphabet of length n is the tuple that consists of gaps only.

Profile approach A profile approach is used, where both the input and alignment forests are forests on vectors with length of the RNA alphabet plus one (for the representation of P-nodes). In each position of the forest, there is a vector with frequencies for bases – that means one for each nucleotide – plus p node labels and b node labels. In this way, the structure and sequence consensus of a multiple forest alignment can be easily deduced by collecting the entries with the highest frequencies from the vectors at the alignment forest's nodes.

Multiple profile alignments can be computed progressively as further explained in [44].

5.10 Time and space complexity

Let us now assess the time and space complexity of the presented algorithms.

Global alignment The time complexity for the global alignment of general forests is $\mathcal{O}(|F| \cdot |G| \cdot ((deg(F) + deg(G))^2)$, and the same applies for the global alignment of RNA forests in the well-formed RNA forest alignment model, as the overall loop structure and tabulation are the same. (see [42]).

The time complexity can be derived from the loop structure of the global alignment algorithms and the computation of an alignment of a closed subforest pair.

In the two outer nested loops of the global algorithm, the running indices are the node indices for the CSF alignment problem, iterating over all nodes of the input forests and thus contributing the factors $|F|$ and $|G|$.

The innermost loops contribute factors of $deg(F)$ and $deg(G)$, respectively, to the comparison of each CSF pair. The comparison of a CSF pair itself contributes $deg(F) + deg(G)$ in worst case, when the two CSFs (i, j) and (k, l) have $maxCSFlength_F(i) = deg(F)$ and $\mathit{maxCSFlen}_G(k) = deg(G)$. The two innermost are computed in sequence, so their contributions are summed up.

Overall, this results in a complexity of

$$\mathcal{O}\Big(|F| \cdot |G| \cdot \Big(\big(deg(F) \cdot (deg(F) + deg(G))\big) + \big(deg(F) \cdot (deg(F) + deg(G))\big)\Big)\Big),$$

which can be simplified to $\mathcal{O}\Big(|F| \cdot |G| \cdot \big(deg(F) + deg(G)\big)^2\Big)$.

Local alignment Computing local forest alignments of general or RNA forests F and G has a time complexity of $\mathcal{O}(|F| \cdot |G| \cdot deg(F) \cdot deg(G) \cdot (deg(F) + deg(G)))$, as explained in [43]. This is also visible from the loop structures of the local alignment algorithm and the comparison of a CSF pair.

The four nested loop structures compute the factors $|F|$, $|G|$, $deg(F)$ and $deg(G)$. And the comparison of the CSF pair inside the loops contributes $(deg(F) + deg(G))$ as explained in the global alignment case.

Space requirements Both the local and the global variant use a dense two dimensional dynamic programming table, and considerably reduce the space requirements compared to previous versions of the algorithm with sparse, four dimensional tables.

See Chapter 12 for measurements regarding both time and space requirements of the discussed algorithms.

6 Gaps and affine gap cost alignment

In this chapter, we introduce a new affine gap scoring model for the forest alignment model and its algorithms. This is a major contribution of this thesis and my PhD work.

6.1 Motivation and model

The forest alignment algorithm inserts gaps in the input forests to construct an alignment forest. Usually, we are interested in comparing homologous sequences, where the sequence has been altered over time. In nature, mutations alter the genomic sequence in organisms, and eventually affect proteins and RNA. They occur spontaneously as errors during mitosis and meiosis, or by other cellular mechanisms, such as transposons. Mutations can also be caused by mutagens such as radiation or chemicals. Different edit and gap models are used to model the real world mutation events on sequences. See [120] for an evaluation of the common gap models in sequence comparison.

The way a gap is represented in the alignment model also affects the way it can be scored. And the gap score function in turn affects the resulting alignments.

6.2 The scattered alignment problem

With the original alignment algorithm of Hoechsmann, edit model and gap score model, sometimes alignments are introduced, which intersperse many small gaps into a sequence. As the algorithm scores each site separately, it does not matter in terms of the score function whether one large gap of length say 10, or 10 small gaps of length one are introduced. An example for such a scattered alignment is shown in Figure 6.1.

As the placement of gaps in the resulting alignment is determined by the scoring scheme and the gap model, let us study the gap model in more detail.

6.3 Gaps

The simplest gap model is to view each gap symbol as an entity, regardless of its neighborhood. This is done in the original tree alignment algorithm, which we just presented in chapter 5.

Let us first address the basic question “What is a gap?”.

6.3.1 Gaps in sequences

On sequences, this is straightforward and there are two common ways how a gap is modeled.

A *singleton gap* is a single position in a sequence alignment that contains a gap symbol “−”. It models a single position insertion or deletion. Single position insertions or deletions may be caused in biology by base slips of the polymerase in DNA replication.

```
global optimal score: -9
Intron_7          GUCUGUUACACGCCGAGAUCGGACUCCGAGUGAUAUCCUC-GA-CGGAUCUG
Intron_8          GUCUGUUACACGAGAGAUCGGUCUCCCGGAUCGAGCCCUCGACGGAUCUGAU

Intron_7          UCCGAUCUUGUGUUUCUCUGUUACUUGAU-UCGAUUACUCUGUUACUAUUCU
Intron_8          GAUCUGUUUCUCUGU-UACUUGAUUCGAUUACUGUUACUAUGUU-C----UC

Intron_7          UCUUUGUUACUACUACUACUACUA
Intron_8          -CG-U--U-CU--U--UG-U--UA

Intron_7         ((..((.(((...((((((((((...(((((((((.....-((-((((.....
Intron_8         .........(((((((((.((((....(((((((((.....(((((((((.....

Intron_7        ))))).))...........)))))))))).)-))))..............)))))
Intron_8        )))))))).......-..)))))))))...))))..........-.----))))

Intron_7         ....))).)).))...........
Intron_8         -))-)--.-..--.--..-.--..
```

Figure 6.1: Example of a tree alignment, which is scattered due to the original, linear gap score model and the default score parameters of *RNAforester*. This alignment has 24 matched basepairs and 23 singleton gaps, which appear as 15 composite gaps. Scoring type: global similarity; Scoring parameters: pair *match*: 10; pair indel: -5; base *match*: 1; base replacement: 0; base indel: -10;

A C A G G G C C G G
A C A − − − C C G G

The additive (also: homogeneous) gap model that is used in basic sequence alignment algorithms such as the Needleman-Wunsch algorithm, only "sees" singleton gaps. Each position in the sequence is scored separately by the algorithm. It has no information about its neighborhood and whether there are neighboring positions with gap symbols or not. Each position with a gap symbol contributes equally to the gap score of the overall alignment according to the cost function. That means, for a gap, the score function is linear in the number of singleton gaps.

A *(composite) gap* is a stretch of positions labeled with the gap symbol seen as an entity.

A C A G G G C C G G
A C A − − − C C G G

Such a model could be used to define a general gap score which is contributed by each continuous stretch of gap symbols, called fixed gap costs. Or it could be used to define a score that depends on the length of the gap, called affine gap costs.

6.3.2 Gaps in forests

Forests as data structures are a generalization of sequences, and they are defined recursively in two directions. So gaps in forests can be extended in two directions as well.

Definition 6.3.1. A *singleton gap* in a forest is a single node labeled with the gap symbol "–".

From a biological point of view, it is much more unlikely to open a new gap than to extend an existing one, as opening a new gap would require another evolutionary event, whereas a longer gap can still be introduced by a single event which inserts or deletes multiple bases at once. Therefore, we need to consider a series of adjacent gaps as one large unit.

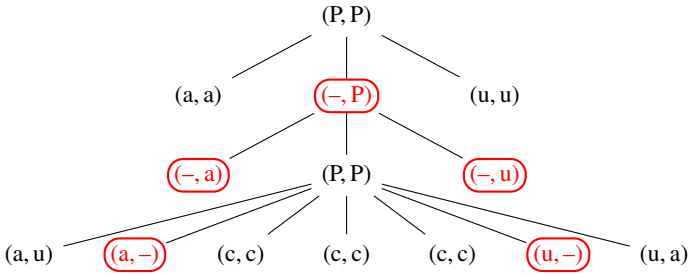


Figure 6.2: Singleton Gaps.

Definition 6.3.2. A *(composite) gap* in a forest is a set of singleton gaps, which is maximal and connected under the union of the parent-child and direct-sibling relations. A gap in an alignment A of F and G is a gap in either the left or right projection of A.

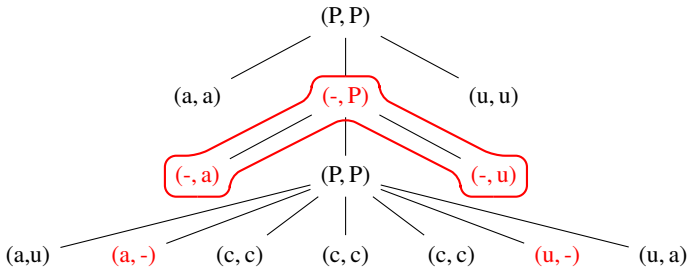


Figure 6.3: (Composite) gap.

Definition 6.3.3. An *oscillating gap* is a set of gaps from an arbitrary component of the alignment tree, which is maximal and connected under the union of the parent-child and direct-sibling relations. An oscillating gap in an alignment A of F and G may consist of multiple gaps in the left or right projection of A.

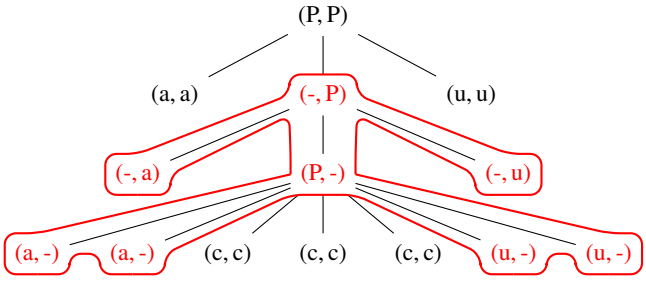


Figure 6.4: Oscillating gap.

In the algorithms that we presented in the previous chapter, we used a linear gap cost model and treated the gaps as singleton gaps. For composite gaps, we suggest an affine gap cost model.

Note that a composite gap in the forest alignment can appear as several gaps in the derived sequence alignment. For example, when several successive base pairs are deleted in F, this will be one gap in the tree alignment, but show as two gaps in the derived sequence alignment (see Figure 6.5). This is exactly what we want.

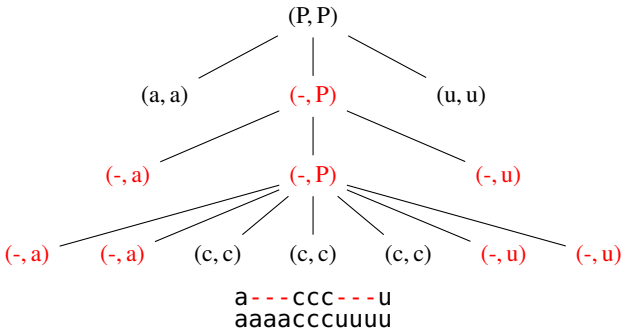


Figure 6.5: A composite gap in the forest alignment can appear as several gaps in the derived sequence alignment.

To see whether we can improve the performance in cases with scattered alignments as discussed above, we are now going to extend the gap cost model to account for an affine gap cost scheme. From this generalization of the forest alignment, new recurrences will arise.

On sequences, the use of an affine gap cost model is a well-established remedy for scattered alignment problems. Let us first recapitulate affine gap costs on sequences.

6.4 Affine gap costs on sequences

When aligning sequences rather than trees, an affine gap cost model leads to three gap modes, for each of which the three edit operations may be computed: Starting in normal mode (no gap mode), we stay in this mode if we begin the alignment with a replacement. If we start the alignment with an insertion, we open a gap in the first sequence, and the first sequence enters gap mode. Similarly, if we begin the alignment with a deletion, we open a gap in the second sequence, and the second sequence enters gap mode. All three cases, the original Case 1 (no

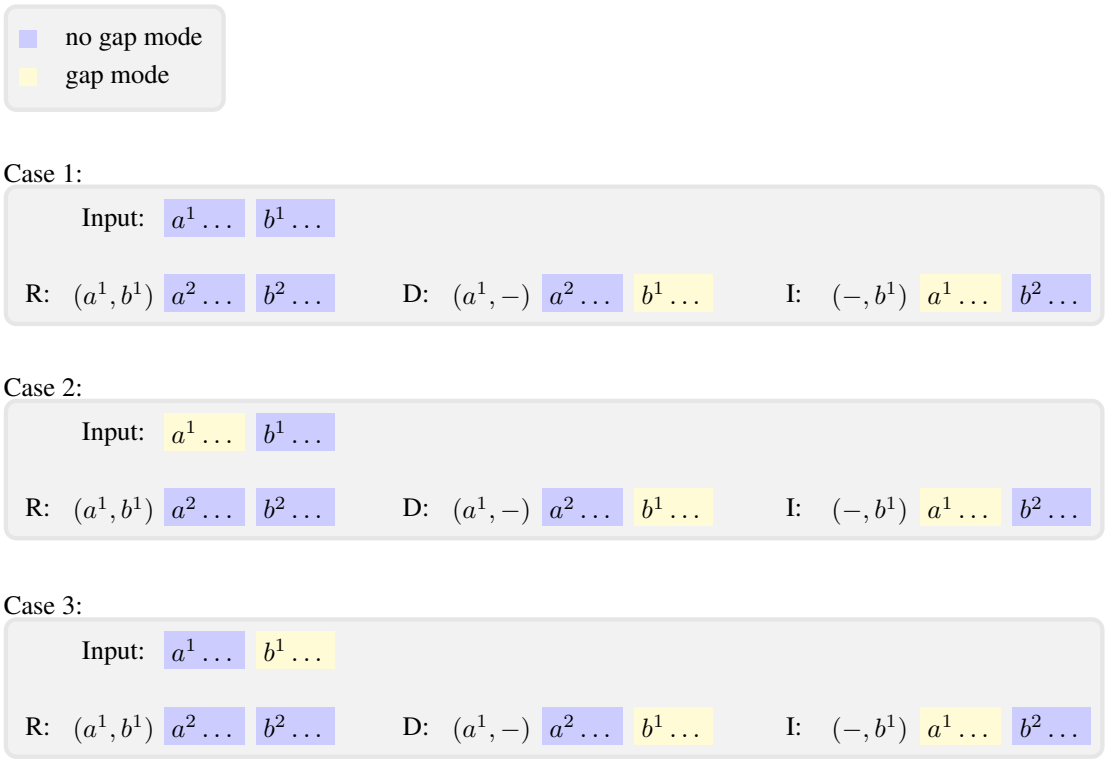


Figure 6.6: The recurrences of the Gotoh algorithm in our notation.

gap) and the two additional cases (Case 2: left sequence gap mode, and Case 3: right sequence gap mode) each contain the usual case distinction for the three edit operations.

For oscillating gaps, the two additional cases collapse to one (either sequence gap mode), giving two cases in total.

This algorithm, although usually presented in a slightly different form, is known from bioinformatics textbooks as the Gotoh algorithm (compare [26, 39, 35]).

6.5 Affine gap costs on forests

To solve the problem of scattered alignments, we suggest an *affine gap cost* model. In this model, we have higher gap opening costs w_{open} (i.e. the score decreases significantly), whereas the costs for gap extension w_{extend} are lower (i.e. the score decreases a bit). The cost function can be written as

$$w(l) = w_{open} + (l-1) * w_{extend},$$

where l is the length of the gap, i.e. the number of positions labeled with the gap symbol. If we now want to construct the score in a structurally recursive fashion, we do not know gap length l in advance and have to compute the score in multiple steps. To be able to do so, we have to keep track whether we already opened a gap and are in *gap mode* already. This is done via the recurrences.

In a forest alignment A of forests F and G, we do not only have to align the rest of the sequence of trees (*over*), but also, in each step, the forest of children trees of the aligned nodes (*down*). This two dimensional recursion causes on the one hand two directions in which we traverse the tree, and on the other hand two types of gap modes. A deletion at the start of the alignment, for example, introduces a gap in the first tree of G. For the rest of the forests of F and G, we may say that the latter has now entered *(left) sibling gap mode*, because its left sibling already opened a gap and paid the opening costs. For the forests consisting of children trees of F and G, we may say that the latter has now entered *parent gap mode*, because its parent has already opened a gap. In this way, we can score gap openings and gap extensions differently.

We will see in the following, that additional cases arise in the forest alignment algorithm due to the three gap modes. This happens in analogy to what we have seen in the above presentation of the Gotoh algorithm with its two gap modes, contributing a constant factor in space and time complexity. As more combinations of gap modes are possible in the forest alignment case, the constant factor due to additional cases is bigger than for the Gotoh algorithm. We discuss this in 6.7 in more detail.

6.6 The search space of forest alignments with affine gap cost model

Starting in normal mode for both input forests, we have a similar case distinction as for forest alignments with a linear gap cost model in the previous chapter, but we keep track of the gap mode:

- $A_\perp = (a, b)$
  - a is the label of $F_\perp$, and b is the label of $G_\perp$.
    - the forest of children of the alignment forest, $A_\downarrow$, is an alignment of the forest of children of F and the forest of children of G, which both stay in normal mode.
    - the forest of right sibling trees of the alignment forest, $A_\rightarrow$, is an alignment of the forest of right sibling trees of F, $F_\rightarrow$ and the forest of right sibling trees of G, $G_\rightarrow$, which both stay in normal mode.

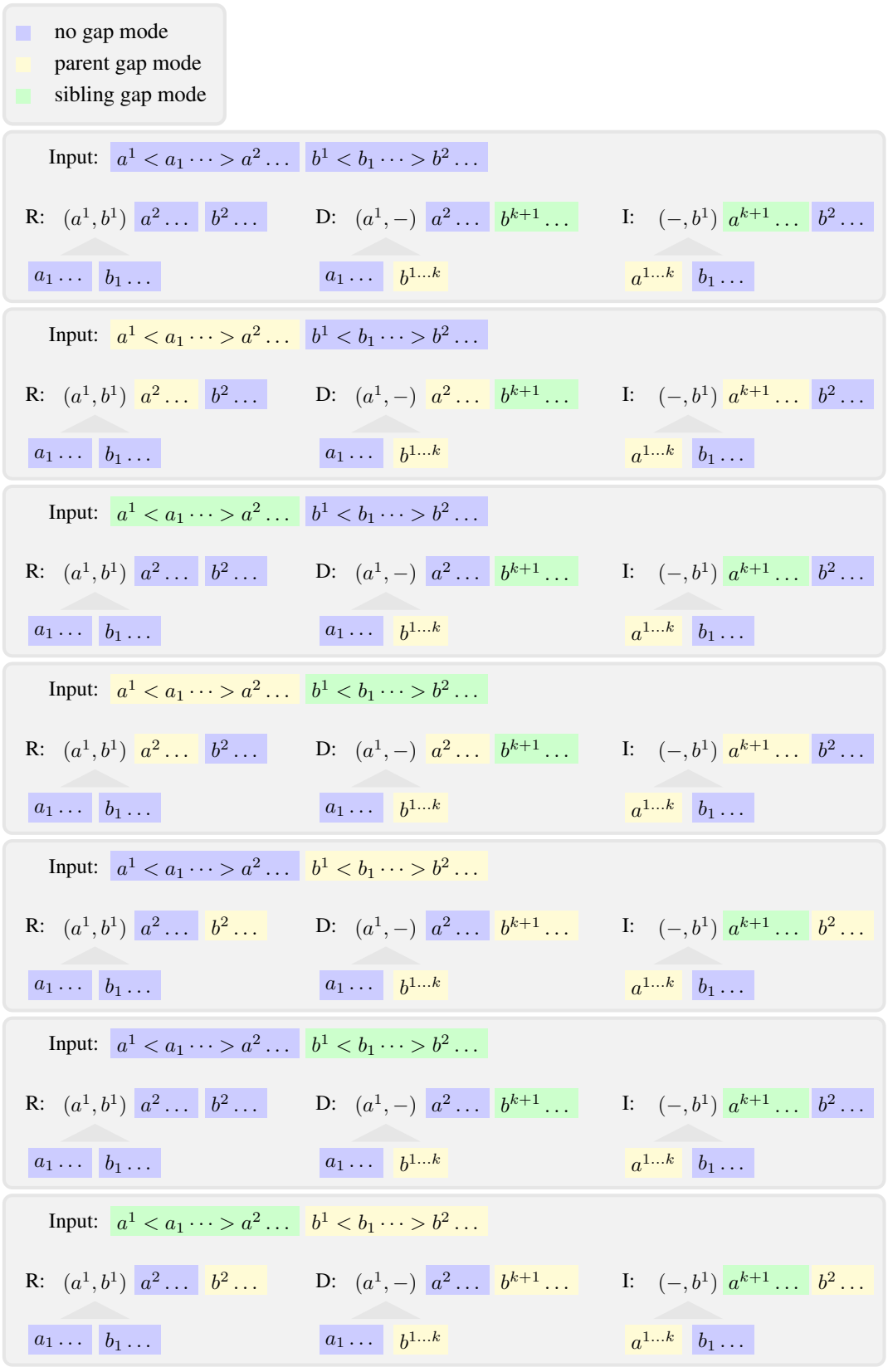
- $A_\perp = (a, -)$
  - a is the label of $F_\perp$.
  - for some $r \in [0, ..., |G|]$
    - the forest of children of the alignment forest, $A_\downarrow$, is an alignment of $F_\downarrow$ and the prefix of length r of G, the latter of which enters parent gap mode.
    - the forest $A_\rightarrow$, is an alignment of $F_\rightarrow$ and the rest of G (the suffix of length $|G| - r$), the latter of which enters sibling gap mode.
- $A_\perp = (-, b)$
  - b is the label of $G_\perp$.
  - for some $r \in [0, ..., |F|]$
    - the forest of children of the alignment forest, $A_\downarrow$, is an alignment of the prefix of length r of F and $G_\downarrow$, the first of which enters parent gap mode.
    - the forest $A_\rightarrow$, is an alignment of the rest of F (the suffix of length $|F| - r$) and $G_\rightarrow$, the first of which enters sibling gap mode.

Starting in other possible gap mode combinations, we get the according other cases of the recurrences, which we will see in the following section.

6.7 Recursive enumeration of the search space of forest alignments with affine gap costs

As for the linear gap cost model, we can derive abstract recurrences for the forest alignment with the affine gap cost model directly from the enumeration of the search space.

See Figure 6.7 for a graphical representation of the algorithm's dynamic programming recurrences.



98

Figure 6.7: The recurrences of the forest alignment algorithm with affine gap costs.

In the recursion, we get different combinations of *parent*, *left sibling* and *no gap mode* for the alignments to be solved. We get

- $A($*no gap mode*, *no gap mode*$)$,
- $A($*parent gap mode*, *no gap mode*$)$,
- $A($*sibling gap mode*, *no gap mode*$)$,
- $A($*parent gap mode*, *sibling gap mode*$)$,
- $A($*no gap mode*, *sibling gap mode*$)$,
- $A($*no gap mode*, *parent gap mode*$)$,
- $A($*sibling gap mode*, *parent gap mode*$)$,

thus, seven cases in total, which are illustrated in Figure 6.7. Note that, although all theoretical combinations of the three modes would be $3^2 = 9$, the cases $A($*parent gap mode*, *parent gap mode*$)$ and $A($*sibling gap mode*, *sibling gap mode*$)$ are logically impossible, as they would imply a node labeled $(-, -)$ in the alignment.

One may restrict the search space further by enforcing an insert-before-delete convention for adjacent indels.

For oscillating gaps, a switch from delete to insert mode or vice versa is scored as a gap extension rather than a new gap opening. The six last tables would collapse to two by merging cases $\{2, 4, 5, 7\}$ as well as $\{3, 6\}$, leaving three dynamic programming tables in total.

Each of the the seven possible cases has the above three subcases corresponding to the three edit operations.

6.8 Tabulation

The transition to the correct next recurrence is performed as implied by the recursive subcases in Figure 6.7. In the same way, the tabulation of intermediate results is written to the correct table.

We label the recursive subcases and tables like this, it is a bit more convenient than just talking about case numbers.

- Case 1 (no gap, no gap) $\rightarrow$ S
- Case 2 (parent gap, no gap) $\rightarrow$ V
- Case 3 (left sibling gap, no gap) $\rightarrow$ V'
- Case 4 (parent gap, left sibling gap) $\rightarrow$ VH'
- Case 5 (no gap, parent gap) $\rightarrow$ H
- Case 6 (no gap, left sibling gap) $\rightarrow$ H'
- Case 7 (left sibling gap, parent gap) $\rightarrow$ V'H

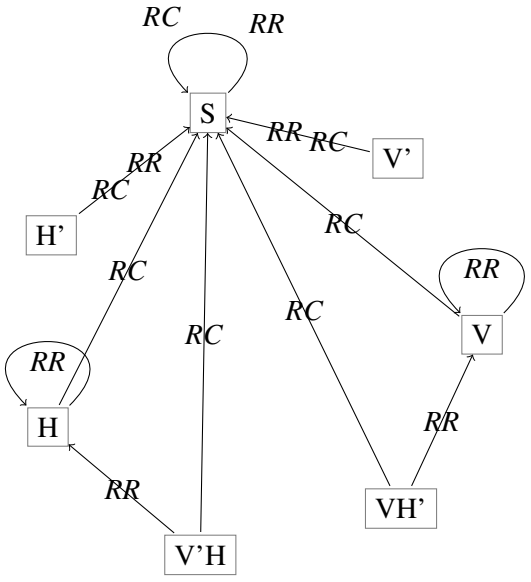
The names are chosen inspired by the usual naming of the tables for the Gotoh algorithm, which in common presentations has a general score matrix S, a matrix with horizontal transitions H and a matrix with vertical transitions V. However, this simple distinction does not apply for our seven tables, and we have chosen the other names to underline the duality of certain table pairs (parent vs. left sibling gap mode).

If we write down the table transitions for to the recursive subcases we know from Table 5.2, the recursive subcase transitions are as follows:

For replacements:

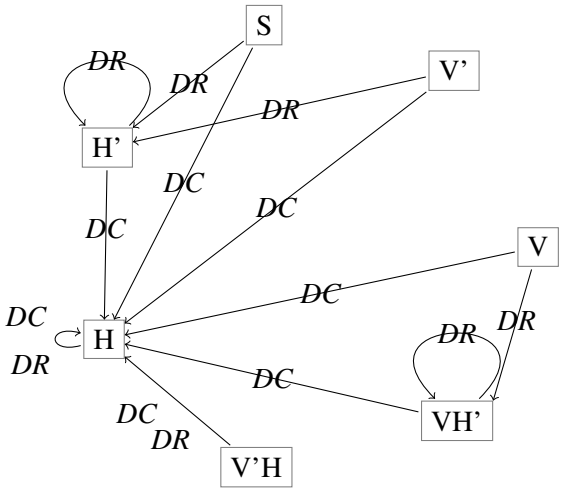
$$
\begin{aligned}
table_{RC} &= S \quad \text{the down case is always in table S} \\
table_{RR}(S) &= S \\
table_{RR}(V) &= V \\
table_{RR}(H) &= H \\
table_{RR}(V') &= S \\
table_{RR}(H') &= S \\
table_{RR}(V'H) &= H \\
table_{RR}(VH') &= V
\end{aligned}
$$

The table transition mechanism for replacements could also be described by the following state machine:



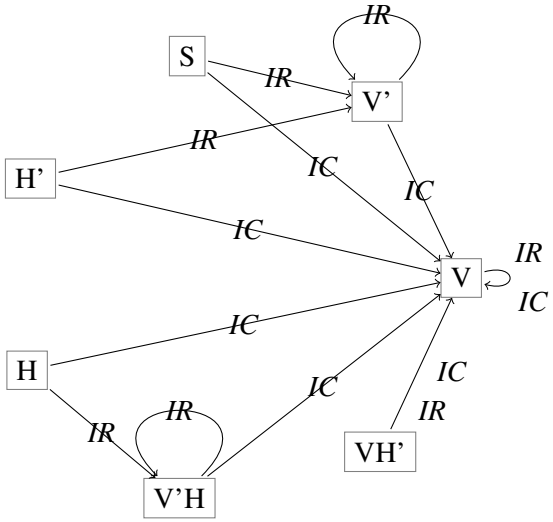
For deletions:

$$
\begin{aligned}
table_{DC} &= H \quad \text{the down case is always in table H} \\
table_{DR}(S) &= H' \\
table_{DR}(V) &= VH' \\
table_{DR}(H) &= H \\
table_{DR}(V') &= H' \\
table_{DR}(H') &= H' \\
table_{DR}(V'H) &= H \\
table_{DR}(VH') &= VH'
\end{aligned}
$$



For insertions:

$$
\begin{aligned}
table_{IC} &= V \quad \text{the down case is always in table V} \\
table_{IR}(S) &= V' \\
table_{IR}(V) &= V \\
table_{IR}(H) &= V'H \\
table_{IR}(V') &= V' \\
table_{IR}(H') &= V' \\
table_{IR}(V'H) &= V'H \\
table_{IR}(VH') &= V
\end{aligned}
$$



Apart from the procedure that selects the right table, the tabulation works as described in Section 5.3.4 within each table.

So the transitions between the tables for every subcase transition can directly be derived from the recurrences in Figure 6.7. But how do we decide whether to apply open or extend costs when inserting or deleting a node?

6.9 Scoring

To realize the concept of affine gap costs, we need to distinguish between opening and extending a gap.

Therefore, in addition of the five different scoring parameters explained in Section 4.3, we need two more parameters for pair deletion opening, and base deletion opening. The seven parameters account for *base* match, *base replacement*, *base deletion opening*, *base deletion (extension)*, *pair* match, *pair deletion opening*, and *pair deletion (extension)*. Recall that the insertion scores are the same as the deletion scores.

Depending on the mode the alignment is in, either gap opening score or gap extension score has to be applied when a deletion or insertion is performed. It can also be seen from Figure 6.7 how to perform this decision.

We have to see what the deletion or insertion do to the mode that the input sequences have. Do they add a new gap that fits to the gap mode of the input sequences? Then, a gap extension cost has to be applied. Do they not fit in, and open a gap when there was no suitable gap mode before? Then, a gap opening cost has to be applied.

For both deletion and insertion score, gap extension scores are applied in most cases.

For a deletion, a gap opening score is added in a transition

- from S. Both components of the alignment are in no gap mode , the deletion opens a gap in second component.

- from V. First component is in parent gap mode, second in no gap mode, the deletion closes the existing gap in the first component, and opens a new one in the second component.
- from V'. First component is in left sibling gap mode, second in no gap mode, the deletion closes the existing gap in the first component, and opens a new one in the second component.

For an insertion, a gap opening score is added in a transition

- from S. Both components of the alignment are in no gap mode, the insertion opens a gap in first component.
- from H. First component is in no gap mode, second in parent gap mode, the deletion closes the existing gap in the second component, and opens a new one in the first component.
- from H'. First component is in no gap mode, second in left sibling gap mode, the deletion closes the existing gap in the second component, and opens a new one in the first component.

The score is again maximized over all possible alignments.

6.10 Relevant CSF Pairs

The observations about the relevant closed subforest pairs for general global alignment, global alignment of RNA and local alignment which we discussed in Section 5.3.1 and 5.5.1, apply for the affine problem as well. They can directly be used to construct appropriate loop structures for general and RNA global and local alignment with affine gap costs.

Besides, the case distinction for the affine variant also works well with P-nodes, because in the current model, only insertions and deletions are affected by the distinction between opening and extension costs, and the pair replacement operation for P-nodes does not have to be changed. See 9 for a possible extension of the edit operations for P-nodes to insertions and deletions.

6.11 Aligning two CSFs

For the alignment of two closed subforests, we get additional edit operations that apply opening costs. Of course this can also be done by adding a switch to the previously defined insertion and deletion operations.

6.11.1 CSFs of general forests

Let us assume that we have a given scoring scheme with scores for match, *rep*, indel_open, *indel*, and recall the definitions of the recursive subproblems *RC*, *RR*, *DC*, *DR*, *IC* and *IR* in Section 5.3.1. The edit operations and their scores for aligning the closed subforests (i, j) of F and (k, l) of G with the affine gap cost model are:

- A *replacement*, which produces the node $(lb_F(i), lb_G(k))$ in the resulting alignment forest, and is scored as

$$\textit{replacement}(i,j,k,l) = \begin{cases} \textit{match} + \textit{score}(\textit{RC}) + \textit{score}(\textit{RR}) & \text{if } lb_F(i) = lb_G(k) \\ \textit{rep} + \textit{score}(\textit{RC}) + \textit{score}(\textit{RR}) & \text{otherwise,} \end{cases}$$

  as in the linear gap cost version.

- A *open deletion* operation, which produces the node $(lb_F(i), -)$ in the resulting alignment forest, and is scored as

$$\textit{deletion\_open}(i,j,k,l) = \textit{indel\_open} + \textit{score}(\textit{DC}) + \textit{score}(\textit{DR})$$

- A *deletion*, which produces the node $(lb_F(i), -)$ in the resulting alignment forest, and is scored as

$$\textit{deletion}(i,j,k,l) = indel + \textit{score}(\textit{DC}) + \textit{score}(\textit{DR})$$

- An *open insertion* operation, which produces the node $(-, lb_G(k))$ in the resulting alignment forest, and is scored as

$$\textit{insertion\_open}(i,j,k,l) = \textit{indel\_open} + \textit{score}(\textit{IC}) + \textit{score}(\textit{IR})$$

- An *insertion*, which produces the node $(-, lb_G(k))$ in the resulting alignment forest, and is scored as

$$\textit{insertion}(i,j,k,l) = \textit{indel} + \textit{score}(\textit{IC}) + \textit{score}(\textit{IR})$$

6.11.2 CSFs of RNA forests

Again, for aligning two forests in extended forest notation for RNA secondary structure, we get extra cases for the P-nodes. Let us assume that we have a given scoring scheme with scores for *base_match*, *pair_match*, base_rep, *base_indel_open*, *base_indel*, *pair_indel_open*, and recall the definitions of the recursive subproblems *RC*, *RR*, *DC*, *DR*, *IC* and *IR* in Section 5.4. The edit operations and their scores for aligning the closed subforests (i, j) of F and (k, l) of G with the affine gap cost model are:

- A *pair replacement*, which is applied only to two P-nodes, so $lb_F(i) = P$, and $lb_G(k) = P$. It produces the node $(lb_F(i), lb_G(k))$, its leftmost child node $(lb_F(i+1), lb_G(k+1))$, and its rightmost child node $(lb_F(rmbi_F(i+1)), lb_G(rmbi_G(k+1)))$ in the resulting alignment forest, and is scored as

$$\begin{aligned} \textit{pair\_replacement}(i,j,k,l) &= \textit{pair\_match} \\ &+ \textit{score\_l} + \textit{score}(\textit{RC}) \\ &+ \textit{score\_r} + \textit{score}(\textit{RR}), \end{aligned}$$

where

$$\begin{aligned} \textit{score\_l} &= \begin{cases} \textit{base\_match} & \text{if } \textit{lb}_F(i+1) = \textit{lb}_G(k+1) \\ \textit{base\_rep} & \text{otherwise,} \end{cases} \\ \textit{score\_r} &= \begin{cases} \textit{base\_match} & \text{if } \textit{lb}_F(rmbi_F(i+1)) = \textit{lb}_G(rmbi_G(k+1)) \\ \textit{base\_rep} & \text{otherwise,} \end{cases} \end{aligned}$$

as in the linear gap cost case.

- A *replacement*, which cannot be applied to P-nodes, so $\textit{lb}_F(i) \neq P$, and $\textit{lb}_G(k) \neq P$. It produces the node $(\textit{lb}_F(i), \textit{lb}_G(k))$ in the resulting alignment forest, and is scored as

$$\textit{replacement}(i,j,k,l) = \begin{cases} \textit{base\_match} + \textit{score}(\textit{RR}) & \text{if } \textit{lb}_F(i) = \textit{lb}_G(k) \\ \textit{base\_rep} + \textit{score}(\textit{RR}) & \text{otherwise,} \end{cases}$$

  also as in the linear gap cost case.

- A *open deletion* operation, which produces the node $(\textit{lb}_F(i), -)$ in the resulting alignment forest, and is scored as

$$\textit{open\_deletion}(i,j,k,l) = \begin{cases} \textit{pair\_indel\_open} + \textit{score}(\textit{DC}) + \textit{score}(\textit{DR}) & \text{if } \textit{lb}_F(i) = \text{P} \\ \textit{base\_indel\_open} + \textit{score}(\textit{DC}) + \textit{score}(\textit{DR}) + \textit{score}(\textit{RR}) & \text{otherwise .} \end{cases}$$

- A *deletion*, which produces the node $(\textit{lb}_F(i), -)$ in the resulting alignment forest, and is scored as

$$\textit{deletion}(i,j,k,l) = \begin{cases} \textit{pair\_indel} + \textit{score}(\textit{DC}) + \textit{score}(\textit{DR}) & \text{if } \textit{lb}_F(i) = \text{P} \\ \textit{base\_indel} + \textit{score}(\textit{DC}) + \textit{score}(\textit{DR}) + \textit{score}(\textit{RR}) & \text{otherwise .} \end{cases}$$

- An *open insertion* operation, which produces the node $(-, \textit{lb}_G(k))$ in the resulting alignment forest, and is scored as

$$\textit{open\_insertion}(i,j,k,l) = \begin{cases} \textit{pair\_indel\_open} + \textit{score}(\textit{IC}) + \textit{score}(\textit{IR}) & \text{if } \textit{lb}_G(k) = \text{P} \\ \textit{base\_indel\_open} + \textit{score}(\textit{IC}) + \textit{score}(\textit{IR}) + \textit{score}(\textit{RR}) & \text{otherwise .} \end{cases}$$

- An *insertion*, which produces the node $(-, \textit{lb}_G(k))$ in the resulting alignment forest, and is scored as

$$\textit{insertion}(i,j,k,l) = \begin{cases} \textit{pair\_indel} + \textit{score}(\textit{IC}) + \textit{score}(\textit{IR}) & \text{if } \textit{lb}_G(k) = \text{P} \\ \textit{base\_indel} + \textit{score}(\textit{IC}) + \textit{score}(\textit{IR}) + \textit{score}(\textit{RR}) & \text{otherwise .} \end{cases}$$

6.12 Prototyping in Haskell

A prototype of the algorithm has been implemented in the functional programming language Haskell. Haskell is in general well suited for rapid prototyping of algorithms.

Solving the forest alignment problem in the spirit of Algebraic Dynamic Programming (see e.g. [33]) made it possible to develop the solution on a very abstract level, while treating the search space construction and the scoring separately. Developing the algorithm in ADP works without explicit tabulation and table indices, and without the need to implement a backtracking procedure. However, some additional parser combinators for forests were required for this approach, as ADP was originally designed for sequence data.

As the Haskell implementation is slow due to the overhead of the functional programming environment, I developed an imperative implementation of the forest alignment with affine gap costs in the language C++.

6.13 Developing a dynamic programming algorithm

To be able to develop dynamic programming algorithms to solve the forest alignment problem with affine gap costs, we have to identify the recurring subproblems, to construct the search space, define the scoring, the tabulation and the order of calculation for the table entries.

The reasoning for the calculation order is directly transferred from the forest alignment with linear gap costs.

1. initialize entry $(0, 0)$ for all seven matrices
2. compute first column, looping over decreasing node index and increasing length index for all seven matrices
3. compute first row, looping over decreasing node index and increasing length index for all seven matrices
4. compute remaining entries, looping over decreasing node index and increasing length index according to the relevant CSFs for the particular problem for all seven matrices

6.13.1 Global forest alignment with affine gap costs

We calculate seven dynamic programming tables. The table S holds the optimal alignments, and the other tables are helping tables for the different combinations of gap modes. They contribute to the overall score in table S.

$$S(x, y) = \textit{forest alignment distance}_{\textit{score}}((i, j), (k, l))$$

First column and row Like in the linear gap cost case, we start with a procedure to compute the first row and column. As the first row and column already involve deletions and insertions, we have to assign an open or extend score according to the state, and also do the correct table transitions for the subproblem of aligning the children forest and the right sibling forest of the deleted, respectively inserted, node.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: Empty alignment, all deletions of complete CSFs, insertions of complete CSFs

1: **function** FIRSTCOLANDROWAFFINE(F, G)
2: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
3: $t(0,0) = 0$ ▷ align two empty CSFs
4: **end for**
5: $(t_c, t_rb) \leftarrow$GETSUBPROBLEMTABLESDEL(t)
6: $open \leftarrow$ISGAPOPENDEL(t)
7: **for** $i \leftarrow |F|, 1$ **do** ▷ complete CSF deletion
8: **for** $j \leftarrow 1, maxCSFlen_F[i]$ **do**
9: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
10: **if** open **then**
11: $t(\beta_F(i,j), 0) =$OPENDELETION($lb_F(i), t\_c(F_\downarrow, 0), t\_rb(F_\rightarrow, 0)$)
12: **else**
13: $t(\beta_F(i,j), 0) =$DELETION($lb_F(i), t\_c(F_\downarrow, 0), t\_rb(F_\rightarrow, 0)$)
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: $t\_c, t\_rb \leftarrow$ GETSUBPROBLEMTABLESINS(t)
19: $open \leftarrow$ ISGAPOPENINS(t)
20: **for** $k \leftarrow |G|, 1$ **do** ▷ complete CSF insertion
21: **for** $l \leftarrow 1, maxCSFlen_G[k]$ **do**
22: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
23: **if** open **then**
24: $t(0, \beta_G(k,l)) =$OPENINSERTION($lb_F(k), t\_c(0, G_\downarrow), t\_rb(0, G_\rightarrow)$)
25: **else**
26: $t(0, \beta_G(k,l)) =$INSERTION($lb_F(k), t\_c(0, G_\downarrow), t\_rb(0, G_\rightarrow)$)
27: **end if**
28: **end for**
29: **end for**
30: **end for**
31: **end function**

Figure 6.8: Compute the table entries for the empty alignment, all deletions of complete CSFs and insertions of complete CSFs under the affine gap cost model. The openDeletion, deletion, openInsertion and insertion functions have to be defined for either RNA or general forest alignment.

Table transitions and open vs. extend To have all the information about the table transitions in one place, we construct functions that look them up given the edit operation and the table the algorithm is currently in. We also have two functions to tell whether to transit into open or extend mode from the given table. This information is used to apply the appropriate cost.

$(t\_c, t\_rb) \leftarrow$ GETSUBPROBLEMTABLESREP(t)
$(t\_c, t\_rb) \leftarrow$ GETSUBPROBLEMTABLESINS(t)
$open \leftarrow$ ISGAPOPENINS(t)
$(t\_c, t\_rb) \leftarrow$ GETSUBPROBLEMTABLESDEL(t)
$open \leftarrow$ ISGAPOPENDEL(t)

The functions return the correct table transitions and open/extend information as explained in Section 6.8.

(General) global forest alignment Based on the previous functions, we construct a global forest alignment algorithm for general forests, which we outline in Figure 6.9 below.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: Alignment of CSF $(i, \mathit{maxCSFlen}_F[i])$ and CSF $(k, maxcsflen[k])$ and relevant subproblems

```
1:  function GLOBALFORESTALIGNMENTAFFINE(F, G)
2:      FIRSTCOLANDROW(F, G)
3:      for i ← |F|, 1 do                                          ▷ remaining cells
4:          for k ← |G|, 1 do
5:              for j ← 1, maxCSFlen_F[i] do
6:                  for t ∈ {S, V, H, V', H', VH', V'H} do          ▷ all tables
7:                      t(β(i,j), β(k,l)) = alignCSFPair_t((i,j),(k,l))
8:                  end for
9:              end for
10:             for l ← 1, maxCSFlen_G[k] do
11:                 for t ∈ {S, V, H, V', H', VH', V'H} do          ▷ all tables
12:                     t(β(i,j), β(k,l)) = alignCSFPair_t((i,j),(k,l))
13:                 end for
14:             end for
15:         end for
16:     end for
17: end function
```

Figure 6.9: Compute the seven global forest alignment matrices (for general forests) under the affine gap cost model.

The alignCSFPair procedure has to be adapted for the edit operations as explained in Section 6.11.1. For each edit operation, it first retrieves the tables for the subproblems, and the information whether opening or extension costs should be applied. It maximizes as usual over the scores after the edit operations' contribution.

The recursive subproblems occur as defined in Table 5.1, and have of course to be solved with the forest alignment algorithm with affine gap costs as well.

6.13.2 Global forest alignment of RNA with affine gap costs

The above algorithm can be refined for RNA structure alignment, as outlined in Figure 6.10.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: RNA structure alignment of CSF $(i, \mathit{maxCSFlen}_F[i])$ and CSF $(k, maxcsflen[k])$ and relevant subproblems

1: **function** GLOBALRNAFORESTALIGNMENTAFFINE(F, G)
2: FIRSTCOLANDROWAFFINE(F, G)
3: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
4: **for** $i \leftarrow |F|, 1$ **do** ▷ remaining cells
5: **for** $k \leftarrow |G|, 1$ **do**
6: **for** $j \leftarrow 1, \mathit{maxCSFlen}_F[i]$ **do**
7: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
8: $t(\beta(i,j), \beta(k,l)) = alignCSFPair_t$((i,j),(k,l))
9: **end for**
10: **if** l>1 **then**
11: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
12: $t(\beta(i,j), \beta(k,l)) = alignCSFPair_t$((i,j),(k,l-1))
13: **end for**
14: **end if**
15: **end for**
16: **for** $l \leftarrow 1, \mathit{maxCSFlen}_G[k]$ **do**
17: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
18: $t(\beta(i,j), \beta(k,l)) = alignCSFPair_t$((i,j),(k,l))
19: **end for**
20: **if** j>1 **then**
21: **for** $t \in \{S, V, H, V', H', VH', V'H\}$ **do** ▷ all tables
22: $t(\beta(i,j), \beta(k,l)) = alignCSFPair_t$((i,j-1),(k,l))
23: **end for**
24: **end if**
25: **end for**
26: **end for**
27: **end for**
28: **end for**
29: **end function**

Figure 6.10: Compute the seven global forest alignment matrices for RNA forests with the affine gap cost model.

6.13.3 Local forest alignment with affine gap costs

Figure 6.11 shows the local forest alignment with affine gap costs, which can be developed from the global alignment like in the linear cost model.

Input: Forests F and G, each given by tables *lb*, *noc*, *rb*, *offset*, *maxcfslen*
Output: All alignments of pairs of CSFs (F', G'), where F' is CSF of F and G' is CSF of G

```
1:  function LocalForestAlignmentAffine(F, G)
2:      FirstColAndRow(F, G)
3:      for i ← |F|, 1 do                                          ▷ remaining cells
4:          for k ← |G|, 1 do
5:              for j ← 1, maxCSFlen_F[i] do
6:                  for l ← 1, maxCSFlen_G[k] do
7:                      for t ∈ {S, V, H, V', H', VH', V'H} do      ▷ all tables
8:                          t(β(i,j), β(k,l)) = alignCSFPair_t((i,j),(k,l))
9:                      end for
10:                 end for
11:             end for
12:         end for
13:     end for
14: end function
```

Figure 6.11: Compute the seven local forest alignment tables for general or RNA forests under the affine gap cost model.

6.13.4 Multiple forest alignment with affine gap costs

The multiple forest alignment can be performed in the same way as for the linear gap cost case. Accordingly, variants were implemented that construct an alignment with affine gap costs every time an alignment is constructed in the progressive profile alignment process.

6.14 Time and space complexity

For $n = |F| = |G|$ and $d = deg(F) = deg(G)$, the time complexity for a local alignment of F and G is $\mathcal{O}(|F| \cdot |G| \cdot deg(F) \cdot deg(G) \cdot (deg(F) + deg(G)))$ as explained in 5.10 (see also [43]).

The time complexity for the global alignment of general and RNA forests is $\mathcal{O}(|F| \cdot |G| \cdot ((deg(F) + deg(G))^2)$ as explained in 5.10 (see also [42]).

This remains asymptotically the same with affine gap scoring, but a constant factor of 7 is expected due to the additional cases contributed by keeping track of the gap modes. In our current implementation, we measured an average runtime factor of $8,02$ for alignments of folded sequences of ≈ 100 nucleotides in length, and $7,79$ for those of ≈ 200 nucleotides in length.

Allowing gaps to oscillate between F and G without opening penalty, as explained above, would merge cases and reduce the constant factor further.

Detailed measurements will be discussed in Chapter 12.

7 Anchoring by shape abstraction

7.1 Motivation - shape anchoring

Often, structures to be aligned come from the same RNA family with a conserved abstract shape [34].

Simply said, abstract shapes record the (forest-like) arrangement of RNA helices, but, in the abstraction level about which we are going to talk, they abstract from their size and from unpaired regions. A shape like "[[][][]]" indicates a clover-leaf structure for sequences of any length. This should be reflected in the resulting alignment, and may also be exploited to speed up the algorithm.

The idea is to use the shape "brackets" to determine anchor points in the structures. The corresponding anchor nodes must be aligned with each other. This constrains the alignment algorithm, and only substructures between the anchor points have to be aligned by our usual algorithm.

See Figure 7.1 for an explanation of shape abstraction level 1–5.

The anchoring constrains the alignment by ensuring that all anchors are aligned to each other. This constraint makes the search space smaller. The reduced search space is restricted to anchored alignments, and it is possible that the optimal unanchored alignment is not found any more with the anchored algorithm if it contradicts to the anchoring. As we base our anchoring constraints on abstract shape analysis, the structure alignment is likely to match the constraints quite well (if the structures share a common shape), as the shape information is based on the structures themselves.

In contrast to the approach of [75] discussed earlier, our anchors do not imply that lowest common ancestors of anchors are matched – they are still candidates for deletion or insertion.

A similar approach on sequences is used to speed up a multiple sequence alignment algorithm by cut-points, which constrain the sequence alignment in a similar fashion to the anchors in our approach. The cut-points have to be aligned to each other as the constraint, and the sequences between the cut-points are then aligned as subproblems in a divide-and-conquer procedure [110].

7.2 Shape abstraction

Shape abstraction functions abstract from length and absolute position of structural elements. At the same time, they preserve nesting and adjacency of the input RNA 2D structure, thus, the arrangement of helices.

There are five shape levels, ordered by their degree of abstraction. Shape level 5 is the most abstract shape level and takes only multiloops and hairpin loops into account. It does not account for bulges and internal loops. Shape level 3, in contrast, reports helix interruptions, but ignores whether they are caused by internal loops, 5'-bulges or 3'-bulges.

According to [90], shape abstractions can formally be defined as mappings from the string representation of structures to the string representation of shapes. The following equations in Table 7.1 define shape abstraction function π_5.

```
CGUCUUAAACUCAUCACCGUGUGGAGCUGCGACCCUUCCCUAGAUUCGAAGACGAG
((((((...(((..(((...)))))))...(((..((.....))..)))))))))..

Shape Level 5: [[][]]
Shape Level 4: [[][[]]]
Shape Level 3: [[[]][[]]]
Shape Level 2: [[_[]][_[]_]]
Shape Level 1: [_[_[]]_[_[]_]]_
```

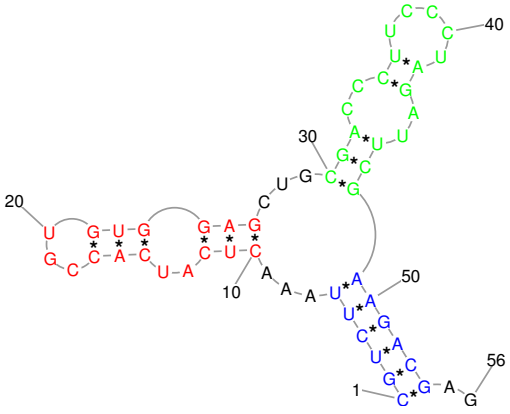


Figure 7.1: An example secondary structure and its dot-bracket representation. The five levels of shape abstractions of the secondary structure are shown below. Level five is the most abstract one, and accounts for hairpin loops and multiloops only. Shape level four records hairpin loops, multiloops, and also internal loops. Level three in addition also records helix interruptions by bulges. Shape level two distinguishes the interrupting bulges in left and right bulge. The least abstract shape level one represents all single stranded regions by underscores, and all pairing regions by a pair of brackets. Figure taken from [52].

Level 5 is the shape abstraction level for which the current version of *RNAforester* provides a speedup by shape anchoring, as for level π_5, parsing can be done in linear time. Parsing efficiency is important, because we want to speed up the forest alignment algorithm with it. Other levels of abstraction would also allow us to retrieve anchoring information, but they are not as easy to parse as level π_5.

The limiting factor is still the forest alignment step. Thus, anchoring by shape abstraction of level 3 (defined in [90]) would be a promising future addition, because although it requires a more complex parsing mechanism, it also may lead to a higher speedup due to the less abstract shape level, which helps to increase the number of anchor points derived from each shape.

$$
\begin{array}{rl}
\pi_5(.) = \varepsilon & \varrho_5(.) = \varepsilon \\
\pi_5(.s) = \pi_5(s) & \varrho_5(.s) = \varrho_5(s) \\
\pi_5(s.) = \pi_5(s) & \varrho_5(s.) = \varrho_5(s) \\
\pi_5((s)) = \mathbf{[}\varrho_5(s)\mathbf{]} & \varrho_5((s)) = \varrho_5(s) \\
\pi_5((s)s') = \mathbf{[}\varrho_5(s)\mathbf{]}\pi_5(s') & \varrho_5((s)s') = \pi_5((s)s')
\end{array}
$$

Table 7.1: Equations defining the shape abstraction for function π_5 for level 5 from the secondary structure in Vienna dot-bracket notation to the shape string. Here, s and s' are non-empty dot-bracket strings with correctly nested brackets, ε is the empty string. The function π_5 is the general shape abstraction function, ϱ_5 is called within a helical region to abstract from the lenght of the helix. Brackets in output string are printed in bold font. Figure based on [90].

7.3 Acquiring anchors for the input forest

We use the shape "brackets" to determine anchor points in the RNA secondary structure. Each pair of brackets represents a substructure that is closed by pairing bases in the secondary structure. We take the outermost basepair, and assign an anchor to it which we number in preorder traversal. An anchor is always a P-node, because each anchor is introduced by a shape bracket, and the anchor is mapped to the outermost P-node that gave rise to that shape bracket. Of course, the outermost basepair is just one possible choice of position to establish an anchor based on the shape abstraction.

In this way, only substructures between these anchor points have to be aligned. A parser is necessary to assign the anchors to the secondary structures that are the input of *RNAforester*. The parser uses the underlying tree grammar of the RNAshapes algorithm, in our case for abstraction level 5. Although this procedure seems straightforward, during the implementation, we faced two challenges, efficient parsing for the shape anchoring, and developing a top down implementation of the alignment algorithm, as the anchored alignment is described in a top down fashion.

To use abstract shapes as anchor points for the anchored alignment algorithm, we have to map them back on the original secondary structure input strings. Thus, we first need a way to parse the input strings which are in dot bracket format. The parse result should be the abstract shape of the structure represented by the input dot bracket string, along with position information for the shape brackets.

Implementation challenges Parsing of shapes in level 5 is a nontrivial parse problem and cannot be done out of the box with LALR(1) parsers (Lookahead LR parsers with 1 unconsumed lookahead symbol used for parsing decisions). This is the class of parsers which most parser generators generate, and the generated parsers operate in linear time related to the input length.

A trick may be used to circumvent this problem and use a fast LALR(1) parser nevertheless. We parse a string of dots and brackets with an LALR(1) parser generated by *flex/bison* (modern

implementations of *lexx* and *yacc*). Afterwards, we call a custom "*skipIt*" function on the stacks of basepairs (sequences of brackets), which skips all basepairs but one, as required for the level 5 abstract shape of the RNA structure.

Besides, we have to keep track of the exact position of the "shape bracket", as we need this position to use the shape as a substructure marker in the original dot bracket string. This can be done with the aid of custom position information in bison, which is usually used for detailed reports of parse errors. We extract this position information and pass it via the *Pair* constructor which performs the helix length abstraction via the *skipIt* function. Via this constructor, the position information of the shape bracket in the input structure is incorporated the final parse tree with the parsed abstract shape.

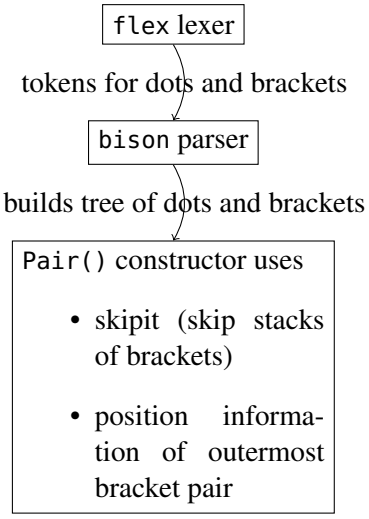


Figure 7.2: Parsing of RNA secondary structure with anchoring information. The function "skipIt" is used to skip consecutive stretches of basepairs, except for one basepair.

To add the anchors to the forest representation of RNA secondary structures, we need an additional table with the anchors numbered as they occur in preorder traversal of the forest. For the pairwise alignment this is all we need.

For multiple alignment, we equip the resulting alignment forest with the anchoring information table as well, as it will be reused in the progressive approach, and other alignments are added to it in each step. Therefore, the method for joining multiple alignments transfers the anchoring information to the newly created profile alignment as well.

7.4 The anchoring

It is not essential that the anchoring is derived from abstract shapes. It can be provided, for example, by expert annotation – if it satisfies the following definition.

Definition 7.4.1. An anchoring is a partial mapping function between the nodes of two forests, with the following constraints:

1. it is a bijection,

2. it preserves the ancestor relation,

3. it preserves the sibling ordering relation.

These properties are also the properties of a valid mapping in the context of the tree edit distance. A different method from the one we present to produce an anchoring can be found in [5].

Figure 7.3 gives an example of an anchoring. By the definition of a gap and of an anchoring, no gap can contain an anchor, which is why affine gaps and anchoring are orthogonal concepts that work well in cooperation.

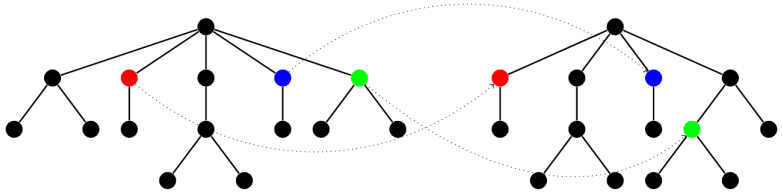


Figure 7.3: Anchored alignment input trees with constraints.

7.5 The anchored forest alignment problem

Definition 7.5.1. The anchored alignment of forest F with n anchors a_i, and G with n anchors b_i; $0 \leq i \leq n$ is the best alignment of F and G, with nodes (a_i, b_i) for $\forall i$.

The goal of the *anchored forest alignment problem* is to compute an anchored forest alignment of input forest F and G according to Definition 7.5.1, given an anchoring between F and G as described in Definition 7.5.1.

The question is now how to compute such an alignment. How do we restrict the "usual" recurrences in order to compute the anchored alignment?

7.5.1 Restricted recurrences for computing alignments with a given anchoring

In this step, we adapt the recurrences so that they reflect the anchored variant.

Adapted recurrences The anchoring has to fulfill the properties of Definition 7.4.1. The anchored alignment is less computationally expensive if the topology of the trees is very similar. If we use shape abstraction to generate anchors at the start of new nested substructures, all anchoring prerequisites are met.

Algorithm

The algorithms to compare anchored forests have the same overall loop structure as in the nonanchored case.

The difference lies in the way a pair of closed subforests is compared - the recursive calls for children and rightbrothers can partially be omitted, as in the speedup trick that Hoechsmann describes in his PhD thesis (Reducing the search space of forest alignments, pp. 73).

Under the objective that only all nodes that are mapped by the given anchoring have to be matched in the resulting alignment, let us observe the anchor combinations and all possible edit operations. For simplicity, let us start with the general forest alignment.

Let α_i be the anchor with index i. The symbol $\varnothing$ indicates that there is no anchor information. The following cases, regarding the anchoring information, arise for the pair ($F_\perp$, $G_\perp$) in the recurrences of the forest alignment algorithm

- $(\alpha_x, \alpha_x) \rightarrow \{match\}$
- $(\alpha_x, \alpha_y) \rightarrow$ stop / worst score
- $(\alpha_x, \varnothing) \rightarrow \{insertion\}$, insert until anchor match
- $(\varnothing, \alpha_x) \rightarrow \{deletion\}$, delete until anchor match
- $(\varnothing, \varnothing) \rightarrow \max\{match, insertion, deletion\}$

These observations lead to a new procedure for comparing a closed subforest pair:

If two matching anchors can be aligned, we directly take the match score without computing the insertion and deletion cases.

Semantically, it is forbidden to align two non-matching anchors. If two non matching anchors should be aligned, the worst score is returned and no further recursive comparison is performed. These steps ensure, that only the necessary parts of the subtrees between anchors are compared.

If we have an anchor at $F_\perp$, but no anchor in $G_\perp$, we can only perform insertions until we have to compare two anchors that match. The same applies when we have an anchor in $G_\perp$, but no anchor in $F_\perp$ – we perform only deletions, until two anchors match.

If both nodes do not contain any anchor information, we have to maximize over all three possible scores that are computed by the performance of the three edit operations and the solution of the recursive subproblems – that means in this case we have to do the same computation as in the non-restricted recurrences of the nonanchored forest alignment.

This new procedure to compare a CSF pair is presented in pseudocode in Figures 7.4, 7.5, 7.6 and 7.7.

Although we discussed the case distinction for general input forests, it applies to RNA forests in the extended forest representation as well, if the correct edit operations for the RNA alignment are used.

Input: CSFPair $(i,\ j,\ k,\ l)$
Output: Fill table for global alignment of CSF $(i,\ j)$ and CSF $(k,\ l)$ and relevant subproblems

1: **function** RECURSIVEFILLANCHORED($i,\ j,\ k,\ l$)
2: **if** $computed(\beta_{f_1}(i,j), \beta_{f_2}(k,l))$ **then** ▷ cell already computed
3: return
4: **end if**
5: **if** $j = 0\ \ \&\ \ l = 0$ **then** ▷ empty alignment - no recursion
6: SETMTRXVAL$(0, 0, 0)$
7: SETCOMPUTED$(0, 0)$
8: return
9: **end if**
10: **if** $a = b\ \ \&\ \ j > 0\ \ \&\ \ l > 0$ **then** ▷ replacement - a and b both anchors or not
11: $score \leftarrow$ REPLACEMENT($i,\ j,\ k,\ l$)
12: **end if**
13: **if** $a = 0\ \ \&\ \ j > 0$ **then** ▷ deletion - a must not be anchor
14: $score \leftarrow$ DELETION($i,\ j,\ k,\ l$)
15: **end if**
16: **if** $b = 0\ \ \&\ \ l > 0$ **then** ▷ insertion - b must not be anchor
17: $score \leftarrow$ INSERTION($i,\ j,\ k,\ l$)
18: **end if**
19: **end function**

Figure 7.4: Recursive call to align a pair of CSFs $(i,\ j,\ k,\ l)$ and compute their anchored global forest alignment. The table *computed* contains information about whether a table cell was already computed in the recursive process, which is used to avoid multiple computations of the same cell. The input forests, for which the alignment table is computed, are f_1 and f_2, and on them, the functions *rb*, *noc*, *label*, β, *down*, *over* are used as defined in the previous chapters. The anchors at i in f_1 and j in f_2 are named a and b, and are numbered in preorder traversal of the forests, $a = 0$ if i has no anchor and $b = 0$ if k has no anchor. The functions *replacement*, *deletion*, *insertion* can be found in the following Figures 7.5, 7.7 and 7.6.

```
1: function REPLACEMENT(i, j, k, l)
2:     if not computed(down_{f1}(i), down_{f2}(k)) then          ▷ recursive subproblems
3:         RECURSIVEFILLANCHORED(i + 1, noc_{f1}(i), k + 1, noc_{f2}(k))
4:     end if
5:     if not computed(over_{f1}(i, j), over_{f2}(k, l)) then
6:         RECURSIVEFILLANCHORED(rb_{f1}(i), j − 1, rb_{f2}(k), l − 1)
7:     end if
8:     ovr ← GETMTRXVAL(over_{f1}(i, j), over_{f2}(k, l))        ▷ computation of replacement
9:     dwn ← GETMTRXVAL(down_{f1}(i), down_{f2}(k))
10:    score ← alg_{replace}(label_{f1}(i), dwn, label_{f2}(k), ovr)
11:    if a > 0 & b > 0 then                                     ▷ if two anchors are matched - done
12:        SETMTRXVAL(β_{f1}(i, j), β_{f2}(k, l), score)         ▷ set matrix cell value
13:        SETCOMPUTED(β_{f1}(i, j), β_{f2}(k, l))
14:        return, from calling function recursiveFillAnchored, too
15:    end if
16:    return score
17: end function
```

Figure 7.5: Compute optimal replacement score for a CSF pair (i, j, k, l) for the anchored global forest alignment problem. The functions $alg_{replace}$, alg_{delete}, alg_{insert}, alg_{choice} are the functions of the scoring algebra. They add the costs for replacement, deletion and insertion operations as explained in 5.4, and the choice function maximizes (or minimizes if we compute distances) the score over the possible solutions.

```
1: function DELETION(i, j, k, l)
2:     lbl = label_{f1}(i);
3:     h ← k                                   ▷ h: node where suffix of split begins
4:     for r ← 0 ... l do                                  ▷ for all splits of f_2
5:         if not computed(down_{f1}(i), β_{f2}(k, r)) then      ▷ recursive subproblems
6:             RECURSIVEFILLANCHORED(i + 1, noc_{f1}(i), k, r)
7:         end if
8:         if not computed(over_{f1}(i, j), β_{f2}(h, l − r)) then
9:             RECURSIVEFILLANCHORED(rb_{f1}(i), j − 1, h, l − r)
10:        end if
11:        dwn ← GETMTRXVAL(down_{f1}(i), β_{f2}(k, r))          ▷ computation of delete
12:        ovr ← GETMTRXVAL(over_{f1}(i, j), β_{f2}(h, l − r))
13:        hscore ← alg_{delete}(lbl, dwn, ovr)
14:        score ← alg_{choice}(score, hscore)
15:        h ← rb_{f2}(h)
16:    end for
17:    return score;
18: end function
```

Figure 7.6: Compute the optimal deletion score over all splits of the second forest for a pair of CSFs (i, j, k, l).

```
1: function INSERTION(i, j, k, l)
2:     lbl ← label_{f2}(k)
3:     h ← i                                   ▷ h: node where suffix of split begins
4:     for r ← 0 ... j do                                  ▷ for all splits of f_1
5:         if not computed(β_{f1}(i, r), down_{f2}(k))) then     ▷ recursive subproblems
6:             RECURSIVEFILLANCHORED(CSFPAIR(i, r, k + 1, noc_{f2}(k)))
7:         end if
8:         if not computed(β_{f1}(h, j − r), over_{f2}(k, l))) then
9:             RECURSIVEFILLANCHORED(CSFPAIR(h, j − r, rb_{f2}(k), l − 1))
10:        end if
11:        dwn ← GETMTRXVAL(β_{f1}(i, r), down_{f2}(k))          ▷ computation of insert
12:        ovr ← GETMTRXVAL(β_{f1}(h, j − r), over_{f2}(k, l))
13:        hscore ← alg_{insert}(dwn, lbl, ovr)
14:        score ← alg_{choice}(score, hscore)
15:        h ← rb_{f1}(h)
16:    end for
17:    return score;
18: end function
```

Figure 7.7: Compute optimal insertion over all splits of the first forest for a pair of CSFs (i,j,k,l).

7.6 Time and space complexity

Recall the efficiency of the forest alignment and the forest alignment with affine gap costs that we discussed in 5.10 and 6.14.

Using $k-1$ evenly placed anchors, such that both F and G are split into k parts of about equal size, n is divided by k in the complexity expression. The complexity for the global is thus reduced from n^2 to $k(\frac{n}{k})^2$.

In general, the degree d may also be reduced, but with RNA structure trees, we tend to have $d \leq 30$ anyway. Efficiency with $k-1$ anchors is then $\mathcal{O}(\frac{n^2}{k}d^2)$.

Space is reduced in the same way as in [43].

Anchoring of subalignments leads to an average speedup of factor ≈ 3 compared to the usual computation, dependent on number and placement of the anchors. See Chapter 12 for measurements regarding time and space of the anchored alignment in practice.

7.7 Discussion

Anchored alignment is a well performing method to speed up the alignment of general and RNA forests. The method is especially useful for RNAs that share an abstract shape, as an anchoring for the alignment can be directly derived from the common shape.

Although we strive to keep the search space as clean as possible, so that it contains only semantically meaningful candidates, this is difficult for the anchored alignment algorithm. We have chosen to give an alignment of two non-matching anchors the worst possible score to disqualify it as a possible solution for an alignment subproblem. This decision has been made because it allows us to reuse the complex backtracing mechanism for the anchored alignment problem instead of having an additional case distinction to reject these meaningless alignments.

The method could be extended in the future by new parsing methods for the RNA structures, to make other levels of shape abstraction usable as well. Shape level 3 with its intermediate level of detail provides a good mixture between information conservation and abstraction, and is probably of high interest for this application, but it cannot be parsed with the SkipIt trick and a bison generated parser. Thanks to Georg Sauthoff for discussions about the SkipIt function.

The implementation of the anchoring also lead to interesting insights about top-down and bottom-up computation of the forest alignment problem. Future studies could reveal which one of them performs best under which circumstances.

8 Top down and bottom up alignment

The anchored variant of this algorithm was developed in a top down fashion, as the computation order of the entries does not exhibit an easy to see loop structure due to the branch-and-bound like abortion of recursive computations. The table entries can therefore not be computed in a bottom up fashion. It is not known in advance which ones will be needed for the overall solution, and to gain speedup we do not want to compute too many of them. In the common implementation of the forest alignment algorithm from *RNAforester 1.0*, the recurrences are translated into a bottom up computation method for filling the DP table. In a second step, the alignment is retrieved by backtracking.

Therefore, I developed an additional top down method of filling the tables in *RNAforester*. The top down and bottom up filling methods are analogues to the Unger and CYK approach in tabulated parsing. The choice of the approach highly depends on the given parsing problem. It is always a trade-off between the number of computed cells and computation time.

In top down style, due to the recursive implementation, just the cells which are needed for the result are computed. The order of computation of the cells comes naturally with the recursive program flow. During deep recursion, the stack of function calls may be very high. This overhead may lead to performance problems. To avoid multiple computations of one cell, an additional table is needed to keep track of those cells that are already computed. This table asymptotically requires the same space as the original table, although in practice it just holds bit values for computed/not computed cells.

In bottom up style, all table entries are computed in an iterative fashion. The programmer has to take care of computation order. For several DP problems, the order of computation of the cells follows a simple pattern with simple loops. In these cases, the bottom up computation is probably the better approach, as it requires no additional if-statements or function calls, just the loops.

For the anchored alignment, a top down approach is required, because the algorithm is described in a top down fashion only, and the required subproblems (i.e. DP table cells) and the order of their computation are not known in advance. All other forest alignment variants can be computed both top down and bottom up.

As an interesting insight, we may take a look at the filling percentage of the tables. When using the top down method, less cells are computed than with the bottom up method, which matches our expectation.

- example 1 (open structure, length 8)

```
> 1
GCAUGUUG
........
> 2
GCAUGUUG
........
```

- example 2 (two short hairpins, length about 20)

```
> 1
GCAGCGGAAUCCCCCACCUG
.(((.((......)).)))
> 2
GCAGCGGAAUCCCCCACCUG
.(((.((......)).)))
```

- example 3 (two medium sequences, each one has two hairpins, length about 70)

```
> 1
GCAGGCAGCGGAAAUCCCCACCUGGUAACAGGUGCCUCUGCGGCCAAAAGCCACGUGUAUAAGAUACACCUGC
...(((.(((((......((((((....))))))..))))).))).....((.(((((((...))))))).)).
> 2
GCAGCGGAAUCCCCCACCUGGUGACAGGUGCCUCUGCGGCCGAAAGCCACGUGUGUAAGACACA
((.(((((......((((((....))))))..))))).)).........((((.....)))).
```

- example 4 (6 real structures of length 78-83, will be aligned in pairs a, b, c)

```
> 1
CCUUUGCAGGCAGCGGAAAUCCCCACCUGGUAACAGGUGCCUCUGCGGCCAAAAGCCACGUGUAUAAGAUACACCUGCA
((((((((((..(((((......((((((....))))))..)))))))(.....))).((((.....)))).))))))
AAGG
))))
> 2
CCUUUGCAGGCAGCGGAAUCCCCCACCUGGUGACAGGUGCCUCUGCGGCCGAAAGCCACGUGUGUAAGACACACCUGCA
((((((((((..(((((......((((((....))))))..)))))))(.....))).((((.....)))).))))))
AAGG
))))
> 3
GCACGCAAGCCGCGGGAACUCCCCCUUGGUAACAAGGACCCGCGGGGCCGAAAGCCACGUUCUCUGAACCUUGCGUGU
(((((((((((((((.......(((((....))))).)))))))(.....))).((((...))))))))))))
> 4
GCAUGAUGGCUGUGGGAACUCCCCCUUGGUAACAAGGACCCACGGGGCCAAAAGCCACGUCCUCACGGACCCAUCAUGC
(((((((((((((((.......(((((....))))).)))))))(.....))).((((....))))))))))))
> 5
GCAUGACGGCCGUGGGAACUCCUCCUUGGUAACAAGGACCCACGGGGCCAAAAGCCACGCCCACACGGGCCCGUCAUGU
(((((((((((((((......((((((....))))))))))))))(.....))).((((....))))))))))))
> 6
GCAUGUUGGCCGUGGGAACACCUCCUUGGUAACAAGGACCCACGGGGCCGAAAGCCAUGUCCUAACGGACCCAACAUGU
(((((((((((((((......((((((....))))))))))))))(.....))).((((....))))))))))))
```

- example 5: s_meliloti (4 real structures of length 108-111)

```
>SmelB126
GTTCCTCATTTCAGGAGGTTGTTTTCTCCCTCAGAATTTCGACTAGCGGACAATCGCACGTGTCGGACAACTGCAGCGC
((((..(((....(((((......)))))..............(((......)))..)))..)))))....(((((((
TCCTGATCGCATCTGCCAGAGCGCTGTTTTTT
(.(((...((....)))))))))))))).....
>SmelA060
AGTTCCTATTCAGACGGAAGTTCTCCGCTGAAACGTTGGCTTAGCGGACAATCGCACAAGTCGACCAACTGCAGCGCTC
((((....(((((.((((.....))))))))).((((((((.(((......))).))))))))))))(((((((((((
CTTTTCAATCCGGGAGCGCTGCCCTTCTT
((.........)))))))))))).......
>SmelA072
AGTTCCTATTTCAGACGGAAGTTCTCCGCTGAAACGTTGGCTTAGCGGACAATCGCACAAGTCGACCAACTGCAGCGCT
((((....((((((.((((.....))))))))))).((((((((.(((......))).))))))))))))((((((((((
CCTTTTCAATCCGGGAGCGCTGCCCTTCT
(((.........)))))))))))......
>SmelB047
GTTCCTATTTCAGAAGGTTGTTTTCTCCTCTGGATTTCGACTCGCGAACAATCGCACATGTCGGACAACTGCAGCGCTT
(((.....(((((.(((.........)))))))).((((((..((((....))))....)))))).))).(((((((((
CGTATCGCATCTGCCGGAGCGCTGCTTTTCTTT
((....((....))))))))))))).........
```

The bottom up method of course always computes 100% of the cells. The top down method computes a fraction of the cells, usually about 50 percent, depending on the number of required intermediate results, and how often the same intermediate results occur.

So in the longer, non-toy cases (e.g. pairwise alignment of 6 sequences of length about 80), it would be faster to use the top down variant in the above example. It seems that the longer the sequences, the more speedup we get by using the top down variant instead. For the global alignment, the bottom up case gets an advantage, as it does not compute all of the entries, but just the ones for the relevant closed subforest pairs. Further studies have to be conducted to decide which method should be the default one. For now, this decision is left to the user.

Table 8.1: In the table we see the fraction of computed table entries in the top down and bottom up computation of local forest alignments with linear gap costs and no anchoring.

Input	top down		
	# cells	# calls	% filled cells
Example 1	1369	137	10.0073
Example 2	6241	3220	51.5943
Example 3	93900	41157	43.8307
Example 4a	114244	62716	54.8965
Example 4b	90592	52769	58.2491
Example 4c	83521	50761	60.7763

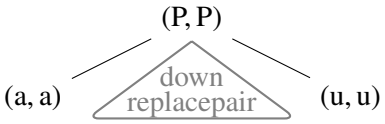
Table 8.2: CPU time (user+system time) to solve example alignment problems in a top-down and bottom-up fashion with linear gap costs and no anchoring.

Input	bottom up	top down
Example 1	0.008s	0.008s
Example 2	0.016s	0.016s
Example 3	0.088s	0.056s
Example 4a+b+c	0.212s	0.164s
s_meliloti	0.640s	0.576s

9 Case correction for pair nodes

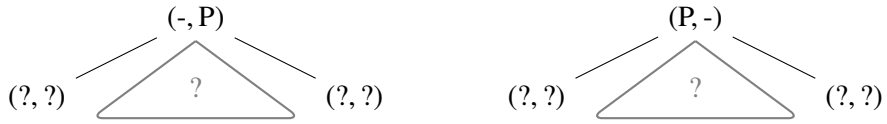
As explained in Section 4.2.2, the alignment algorithm of Jiang et al., and its version on forests, which was presented as a general forest alignment algorithm by Hoechsmann, is not necessarily producing well-formed RNA forest alignments. As a consequence, we need to adapt the edit operations to take care of the P-nodes in the forests we want to align.

To produce well-formed RNA forest alignments, according to [42], the replacement case has to be adapted for pair replacements. The way to make sure that well-formed RNA forest alignments are produced by the replacement edit operation follows directly from the definition of the well-formed RNA forest alignment. A basepair is replaced (in fact, matched, as it always has the label P) by also replacing its pairing bases, as described in [42]. An unpaired base can be replaced as in the general forest alignment.



The P-node replacement edit operation leads to an unwanted situation: Although possible in the general algorithm, a P-node and B-node may never be aligned in our model. We are able to rule out this situation via the scoring scheme.

What happens at the insertion or deletion of P-nodes? These operations are clearly intended in the algorithm, and even have their own scoring parameter, but is not clear what the leftmost and rightmost children nodes of the inserted (resp. deleted) P-node look like.



Quoting [42], the influence of the P-node to the edit operations is:

> Remember the observations concerning the search space of [general] forest alignments [..]. Clearly, the search space definitions of Case 2 (delete) and Case 2 (insert) are not affected by the constraints that make an alignment a well-formed alignment. Case 1 (replacement) treats the relabeling of nodes, and I refine this case to be consistent with the definition of well-formed [RNA] forest alignments. The following case analysis of Case 1 [for RNA forests] replaces Case 1 [..] resulting in a definition of the search space for well-formed forest alignments.

The statement claims that a custom P-node case is needed for the pair replacement edit operation only, and not for pair deletion or pair insertion. I object to this statement and will explain why additional cases are necessary for pair deletion and pair insertion.

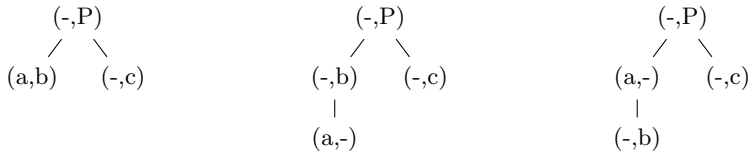


Figure 9.1: Three alignment trees of the trees F = a and G = P(b,c), where the middle and right "mean" the same. The right one is considered artefactual, as the b is removed from its P-node. A similar situation could happen after deleting a P-node.

Do we need more P-node edit operations? Although I follow this claim in the presentation of the algorithms in chapter 5 and 6, and programmed the new algorithms with the affine gap cost model in this way for consistency with the old program, I suggest that the deletion and insertion operations should be refined for P-node, because otherwise the search space contains malformed RNA forest alignments.

Pollution of the search space, be it with semantically meaningless or wrong solutions, should be avoided in general, because a clean search space is a prerequisite for a probabilistic solution of the DP problem.

My suggestion is justified by the observation that, with the common operations, unwanted cases are possible when deleting or inserting a P-node: For example, when we first insert a P-node, and then delete one or both children, the resulting alignment is illegal with respect to the well-formed RNA forest alignment model, because the P-node as an entity is violated.

The problem of P-node entity violation Semantically, a P-node and its two outmost children are one unit of RNA structure. The alignment model allows to break up this unit and recode it ambiguously in many different forms, because the pairing bases can be "removed" from their P-node by intervening gaps. See Figure 9.1. This semantic ambiguity thwarts the use of probabilistic scoring schemes, as the most likely alignment tree is not the most likely alignment. This problem does not arise in the arc annotated sequence model, because basepairs are modeled explicitely and in separate data structures from the sequence information.

To keep the P-node as an entity, we would have to take care of the pairing bases of the inserted P-node by inserting or replacing them in the same operation, so no other nodes can be deleted in between.

This problem might have been overlooked because these alignments never received a score good enough to show up in the (sub-)optimal results. Nevertheless, the search space of RNA structure alignments is contaminated by these malformed alignments.

9.1 Constructing well-formed RNA forest alignments

As a solution for this problem, I suggest to add case distinctions to the insertion and deletion cases as well.

On the deletion of a P-node, we may *not* insert a node at the leftmost and rightmost child position of that node. Thus, we may either

- delete the P-node and its two pairing bases as well (also known as base pair deletion in the literature), or
- delete the P-node and the left base, replacing the right base (base pair altering, right, in the literature), or
- delete the P-node and the right base, replacing the left base (base pair altering, left, in the literature), or
- delete the P-node only, and replace its two pairing bases by two unpaired bases (base pair breaking in the literature),

while still getting a well-formed RNA forest alignment.

The same case distinction can also be found in [26].

In the same way, we get four cases for the insertion of a P-node.

The second and third case are arguable cases and there are good reasons to allow and to forbid them. They could be forbidden, because their result can also be achieved by combining a base pair bond deletion and the deletion of an unpaired base. We allow them, because otherwise "wasting a step" in the recurrences would unnecessarily elongate the alignment.

That means that for comparing RNA forests, to make sure we get a well-formed RNA forest alignment, we have to refine the cases for pair replacement, deletion and insertion.

I developed these additional cases and implemented them in *RNAforester*, but also added a switch to enable/disable them for backwards compatibility and to avoid the slowdown caused by the additional cases.

9.2 The recursive subproblems of the edit operations

When constructing refined edit operations for the four (resp. eight) above cases, these new edit operations lead to subproblems of new forms for the down alignment. The usual $F \rightarrow F^{\downarrow}$ (down) and $F \rightarrow \circ|F^{\downarrow}|\circ$ (mdown) transitions can not adequately describe all four (resp. eight) of them, but only the first and the last one(s).

We describe the transitions to the new subproblems directly on closed subforests in our index notation, as it is versatile enough to describe the problem.

9.3 The recursive subproblems of the general forest edit operations

To familiarize ourselves with the notation, we first write down the transitions for the edit operations of the general forest alignment. All the transitions can directly be deduced from the abstract recurrences of the forest alignment problem (Figure 5.1), consulting the closed subforest subproblem notations from Lemma 1 and the forest index transition functions *rb* and *noc* from Section 5. Recall that $noc_F(i)$ is the number of children of a node i in forest F, $rb_F(i,k)$ is the k-th right brother of node i in forest F, and the tree nodes are indexed by their number in preorder traversal, so the leftmost child node of node i is just node $i+1$. All the alignment

transitions that align the remaining right sibling forests can be reused for the RNA forest edit operations later, as only the subproblem of aligning the children forests has to be adapted for RNA, and the subproblem of aligning the right sibling forests is unchanged.

For the general forest alignment's usual edit operations on the CSF pair (i,j),(k,l), the transitions to the subproblems are as follows:

- *downReplace*$((i,j),\,(k,l)) \quad \rightarrow \quad ((i+1, \mathit{noc}_F(i)),\,(k+1, \mathit{noc}_G(k)))$
- *rightReplace*$((i,j),\,(k,l)) \quad \rightarrow \quad ((\mathit{rb}_F(i), j-1),\,(\mathit{rb}_G(k), l-1))$

For deletion and insertion, the running index for the splits is required for the transition. Here, r is the length of the split, and h is the first node after the split.

- *downDelete*$((i,j),\,(k,l),\,r,\,h) \quad \rightarrow \quad ((i+1, \mathit{noc}_F(i)),\,((k,r))$
- *rightDelete*$((i,j),\,(k,l),\,r,\,h) \quad \rightarrow \quad ((\mathit{rb}_F(i), j-1),\,((h, l-r))$
- *downInsert*$((i,j),\,(k,l),\,r,\,h) \quad \rightarrow \quad ((i,r),\,(k+1, \mathit{noc}_G(k))$
- *rightInsert*$((i,j),\,(k,l),\,r,\,h) \quad \rightarrow \quad ((h, j-r),\,((\mathit{rb}_G(k), l-1))$

9.4 The recursive subproblems of the RNA forest alignment

To identify the recursive subproblems of the RNA forest alignment, a case distinction has to be made between base nodes and P-nodes.

For the base nodes, the recursive subproblems are the same as the ones we just postulated for the general forest alignment. In addition, there is no recursive down alignment problem, because b nodes are always leaves.

For P-nodes, new edit operations lead to new subproblems. These subproblems can be deduced by starting from the subproblems in the edit operations in the general forest alignment, and taking the edit operations on the pairing bases of the involved P-node into account.

9.4.1 The recursive subproblems of the pair replacement operation

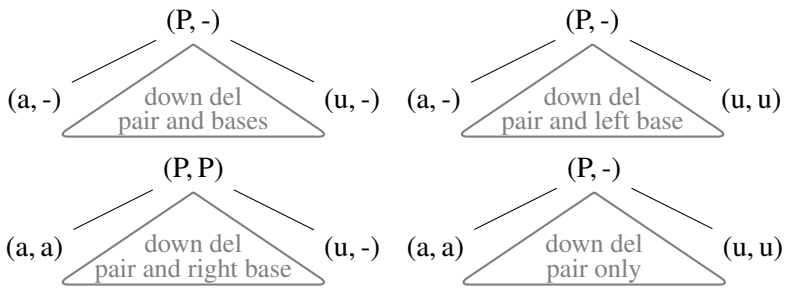
For the replacement of a P-node, we have to remove the leftmost and rightmost node from the usual *downReplace* transition, as they are matched by the edit operation and are not part of the recursive subproblem. Therefore, we increment the node indices not by one to start at the leftmost children (like in the general replacement), but by two. This increment correctly leads us to the right brother of the P-node's left pairing base, because a base is always a leaf. The length indices are decremented by two, because two nodes – the leftmost and the rightmost node – are removed from the children forests that we align in the subproblem. This is the *mdown* operation, which we already know as $F \rightarrow \circ|F^{\downarrow}|\circ$.

- *downReplacepair*$((i,j),\,(k,l)) \quad \rightarrow \quad (i+2, \mathit{noc}_F(i)-2, k+2, \mathit{noc}_G(k)-2)$

The transition for the alignment of the right sibling forests is *rightReplace*, as defined above.

9.4.2 The recursive subproblems of the pair deletion operations

To determine the involved subforests for the down alignment in the subcases of a pair deletion, consider the following examples for the four subcases as explained above:



When deleting a P-node and its bases, we have to remove these nodes from the *downDelete* operation of the general forest alignment. As the pairing bases of the P-node are involved in deletion operations, they are not part of the subproblem, and we only have to align from $i + 2$, with length $noc_F(i) - 2$ in the CSF of F (the argument is the same as for the pair replacement). In the second component of the CSF pair, the CSF of G, we transit to the same subproblem as in the general deletion case, because no node of G is consumed in the edit operations of the left- and rightmost children of the P-node we deleted.

If we delete the P-node and its left base, replacing the right base by an unpaired base, we have the same situation as above for the first component of the CSF pair, as the pairing bases of the P-node are involved in the deletion and replacement. For the second component, the CSF of G, we again get almost the same subproblem as in the general deletion case, but one node at the right is consumed in the replacement, and therefore the CSF of G to be aligned in the subproblem is shortened by one and has length $r - 1$.

On deleting the P-node and its right base, the left base is replaced by an unpaired base. In this case, the same reasons as in the previous case apply for the transition to the recursive subproblem in F. In G, we again have to shorten the closed subforest for the subproblem by one, yielding a length of $r - 1$, because one node is consumed in the replacement of the left pairing base of the P-node. This node is consumed, so we start the recursive subproblem at its right brother, from index $k + 1$.

The "deletion of the P-node only" means that its pairing leftmost and rightmost child nodes are both part of a replacement operation with an unpaired base. In this case, the transition for the CSF in F is again as explained above, and the transition for the CSF in G starts from the right brother of the left consumed node from F at node index $k + 1$, and has a length of $r - 2$ to account for the two nodes consumed from G in the replacement operations of the leftmost and rightmost child of the P-node.

Thus, the following CSF pair transitions arise for the four cases:

- *downDelPairAndBases*$((i,\, j),\, (k,\, l),\, r) \quad \rightarrow \quad ((i + 2,\, noc_F(i) - 2),\, (k,\, r))$
- *downDelPairAndLeftBase*$((i,\, j),\, (k,\, l),\, r) \quad \rightarrow \quad ((i + 2,\, noc_F(i) - 2),\, (k,\, r - 1))$
  (requires $r \geq 1$)

- *downDelPairAndRightBase*$((i,\, j),\, (k,\, l),\, r) \rightarrow ((i+2,\, \mathit{noc}_F(i)-2),\, (k+1,\, r-1))$ (requires $r \geq 1$)
- *downDelPairOnly*$((i,\, j),\, (k,\, l),\, r) \quad \rightarrow \quad ((i+2,\, \mathit{noc}_F(i)-2),\, (k+1,\, r-2))$ (requires $r \geq 2$)

The transition for the alignment of the right sibling forests is $rightDelete$, as defined above.

9.4.3 The recursive subproblems of the pair insertion operations

With a similar argument to the one in the previous section, the transitions for the insertion cases can be deduced:

- *downInsPairAndBases*$((i,j),\, (k,l),\, r) \quad \rightarrow \quad (i, r, k+2, \mathit{noc}_G(k)-2)$
- *downInsPairAndLeftBase*$((i,j),\, (k,l),\, r) \quad \rightarrow \quad (i, r-1, k+2, \mathit{noc}_G(k)-2)$ (requires $r \geq 1$)
- *downInsPairAndRightBase*$((i,j),\, (k,l),\, r) \quad \rightarrow \quad (i+1, r-1, k+2, \mathit{noc}_G(k)-2)$ (requires $r \geq 1$)
- *downInsPairOnly*$((i,j),\, (k,l),\, r) \quad \rightarrow \quad (i+1, r-2, k+2, \mathit{noc}_G(k)-2)$ (requires $r \geq 2$)

The transition for the alignment of the right sibling forests is $rightInsert$, as defined for the alignment of general forests.

9.5 A new well-formed RNA forest alignment algorithm

With the knowledge about the recursive subproblems, we can finally construct forest alignment algorithms that produces well-formed RNA forest alignments. We add subcases for the insertion and deletion operations with the required constraints for the running split index r. We compute their scores from the appropriate subproblems and the scores of the edit operations of P-node, left pairing base and right pairing base, and minimize over the scores of the four subcases.

Additional implementation effort was required besides adding the new deletion and insertion cases to the main algorithms for global and local alignment, linear and affine gap costs. More scoring functions for the evaluation algebras that are used to construct the score for each subcase were added. A much more intricate backtracking procedure that can handle the additional specialized insertion and deletion operations, and puts the correct P-nodes into the alignment forest, was implemented as well.

The new well-formed RNA forest alignment algorithm does not affect the anchoring, as it only treats the pairing bases of P-nodes in a new way, and does not affect other nodes. In our case of anchoring the alignment by abstract shape information, an anchor is always at a P-node, because each anchor is introduced by a shape bracket, and the anchor is mapped to the outermost P-node that gave rise to that shape bracket.

9.6 Discussion and Complexity

The case distinction introduces four completely new subcases for the insertion and deletion edit operations in the alignment. We have to recursively solve the subproblems for the four cases, and maximize over four scores for each insertion and deletion now, which slows down the algorithm, although it stays in the same asymptotic complexity class.

A possible speedup that still guarantees well-formed RNA alignments would be to forbid the cases that align a basepair to one single base. Then, the search space does not contain all possible well-formed RNA alignments any more, but only two subcases have to be computed per insertion and deletion.

The additional case distinction for pair insertions and deletions can be switched on via a command line parameter `--pIndels`, which is explained in Section 10.1.1.

9.7 Updated recurrences for aligning RNA forests

With the case distinction and the information about the involved down alignment subproblems, we developed updated abstract recurrences for the alignment of RNA forests, that take P-node into account for replacement, deletion and insertion. They are depicted in Figure 9.2.

9.8 Updated abstract recurrences and adjustment of the scoring functions

Transferring the new case distinction to the recurrences of the affine gap cost variant is straightforward, only one question remains: What do we do for open and and extend with the new P-node indel operations?

We can just apply them as in the affine gap cost previous algorithm, which is reasonable, because for example in a P-node deletion, the "parent gap mode" is passed on to the down alignment subproblem in the same way as if a deletion happened in the general algorithm. The "left sibling gap mode" is also passed on in the same way to the right aliment subproblem, as if a deletion happened in the general algorithm. The pairing bases of the P-node, which might end the "left sibling gap mode", are in the subtree and do not interfere with the extension of the gap via the P-node deletion into the right alignment.

Of course, another reasonable decision would be to end the gap mode based on the operations that are applied to the pairing bases of the deleted P-node, although the deleted P-node itself would extend the gap.

We depicted the new recurrences for aligning RNA forests with affine gap costs in Figures 9.3, 9.4, 9.5, 9.6, 9.7, 9.8 and 9.9.

9.9 Discussion of the new P-node edit operations

From the analysis of the P-node edit operation adjustment we performed above, a more general necessity can be derived.

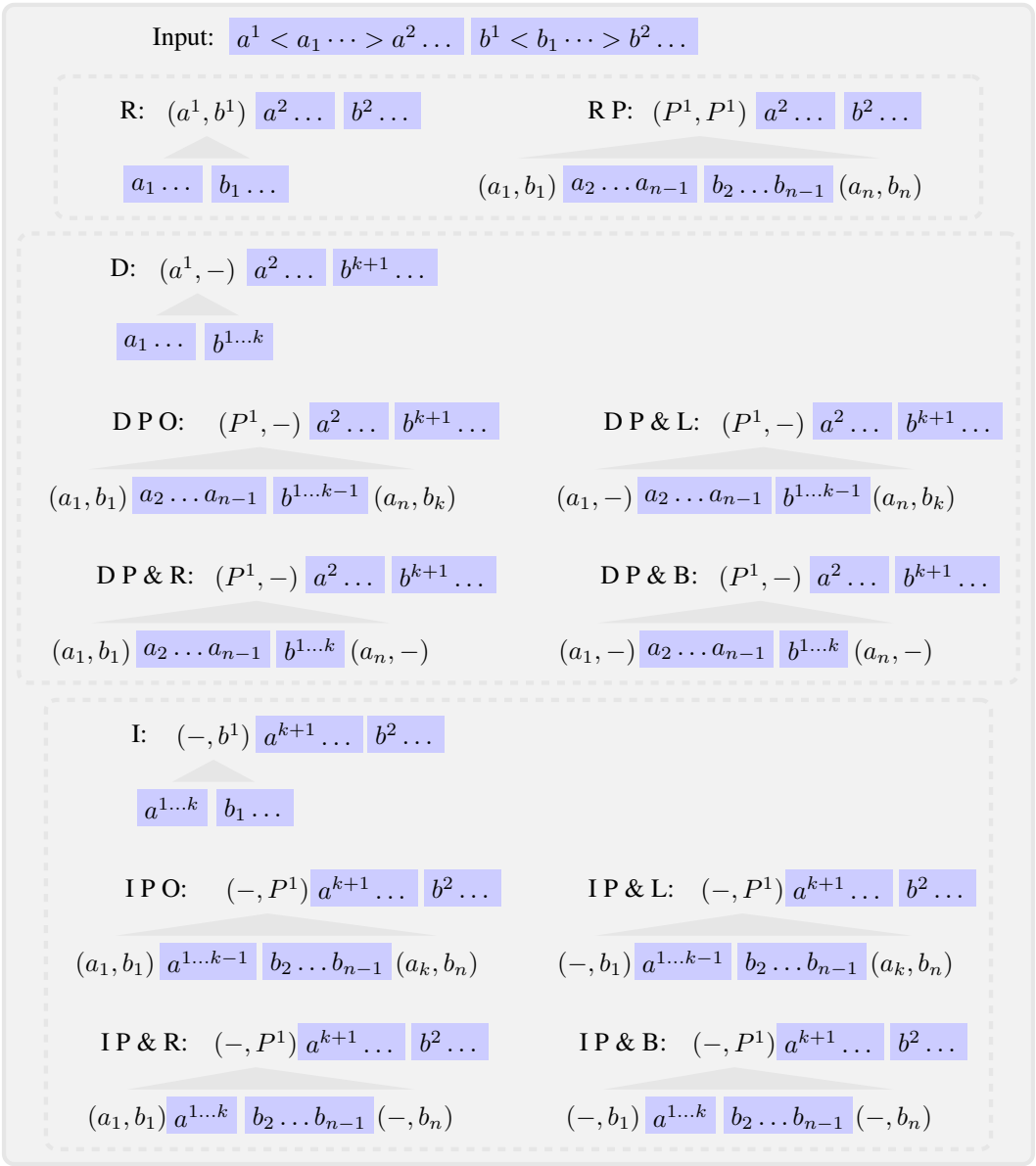


Figure 9.2: Case distinction for the tree alignment algorithm. Compare also Figure 5.1.

Case 1:

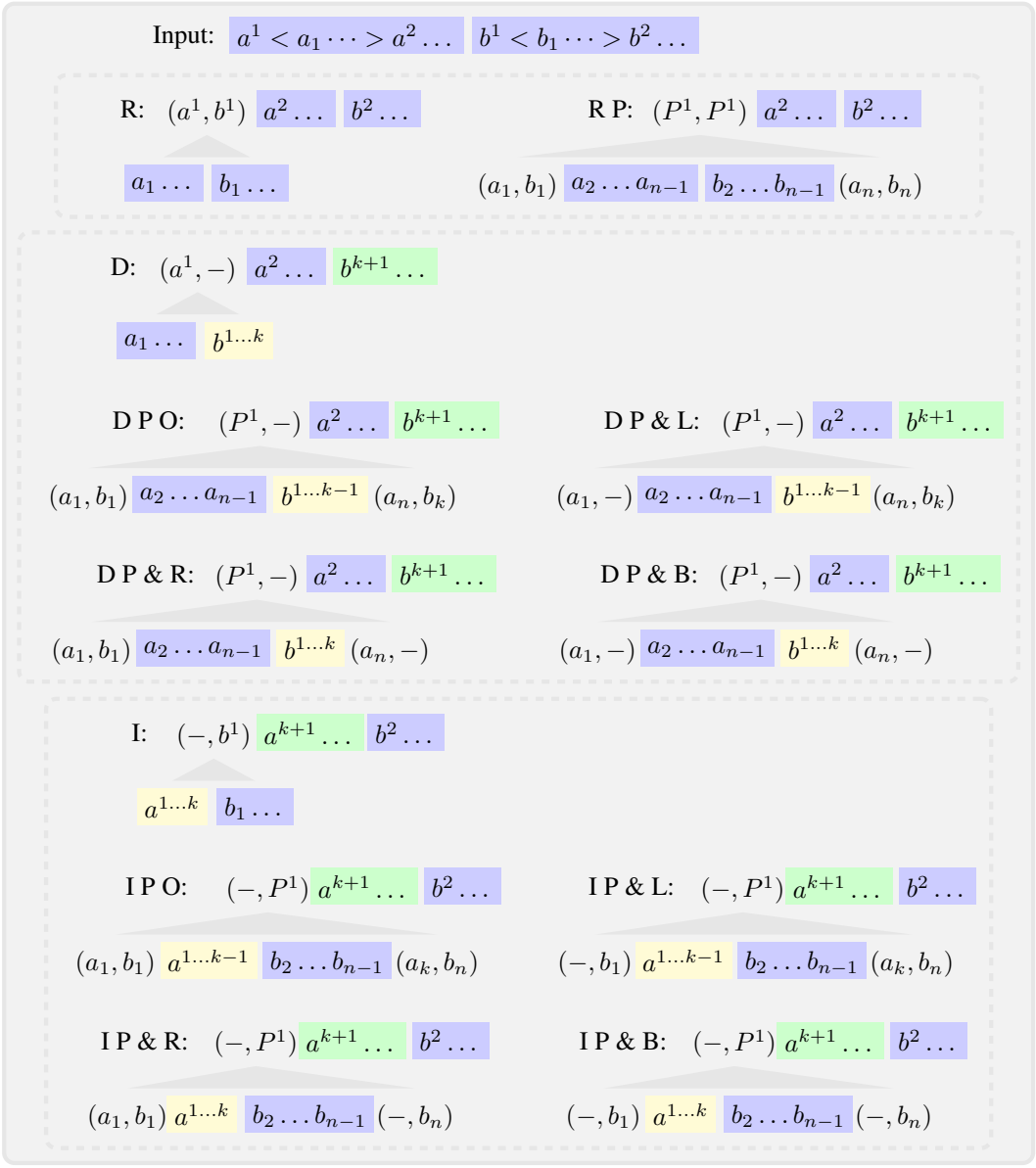


Figure 9.3: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-nodes. As in Figure 6.7, no-gap-mode is blue, parent gap mode is yellow and sibling gap mode green and the overall case distinction has subcases for the edit operations *replacement*, *deletion* and *insertion*. As before, a *replacement* can be a) a *base replacement* or b) a *pair replacement*. New subcases for a *deletion* are a) *delete pair only (DPO)*, b) *delete pair and left base (DP&L)*, c) *delete pair and right base (DP&R)*, d) *delete pair and bases (DP&B)*. Likewise, there are four new operations for an *insertion*. First case of the case distinction.

Case 2:

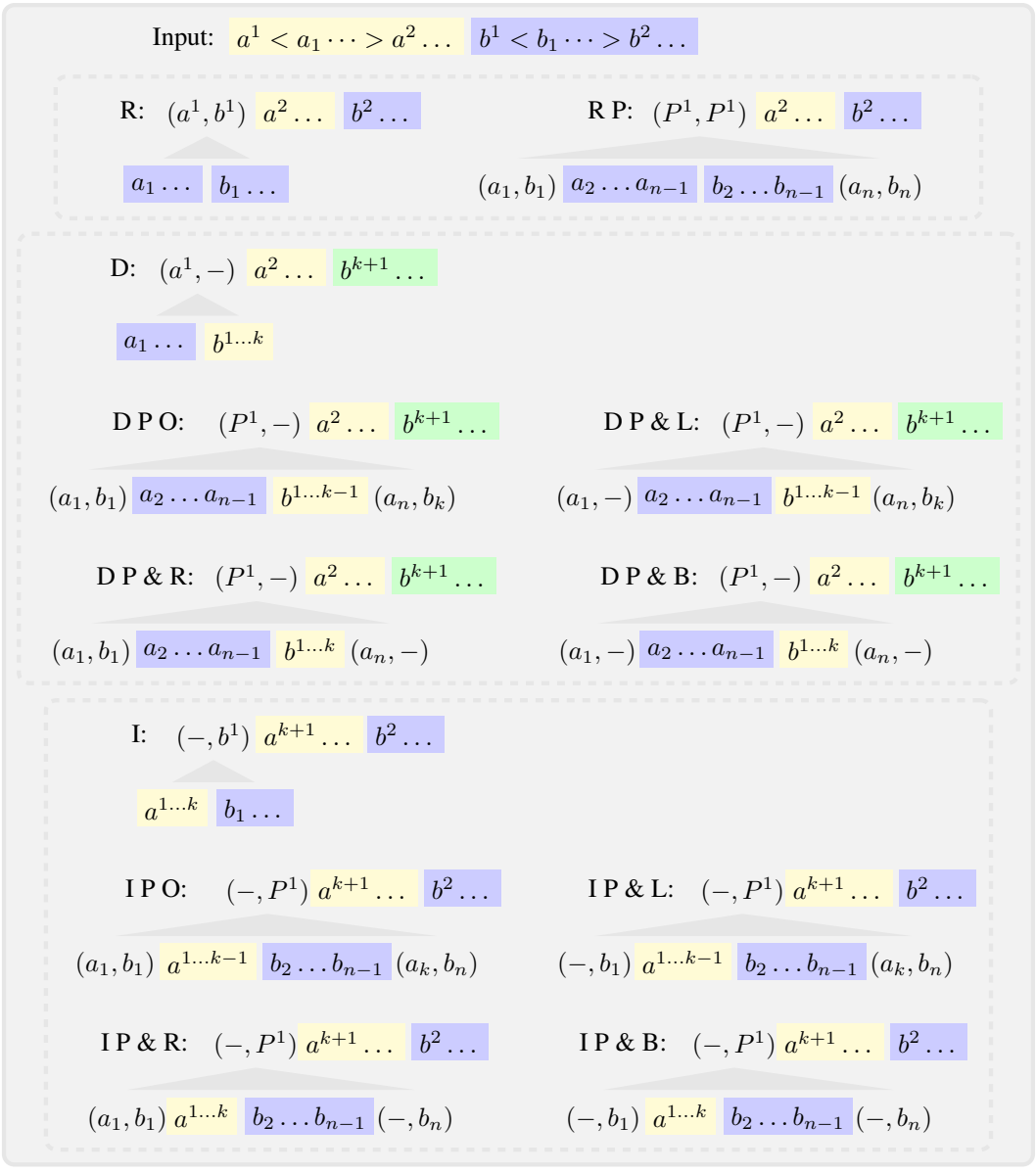


Figure 9.4: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-node. Second case of the case distinction.

Case 3:

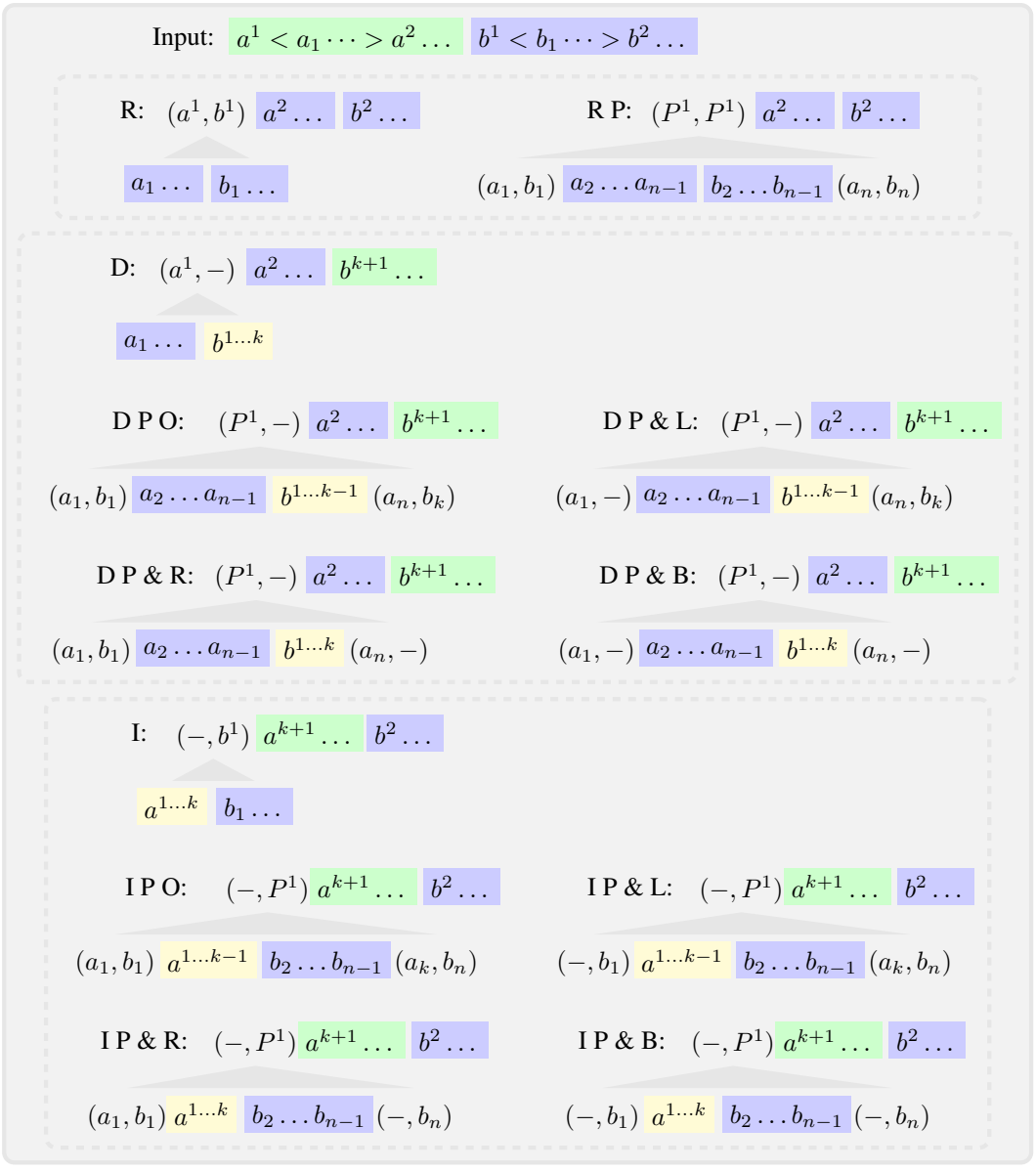


Figure 9.5: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-node. Third case of the case distinction.

Case 4:

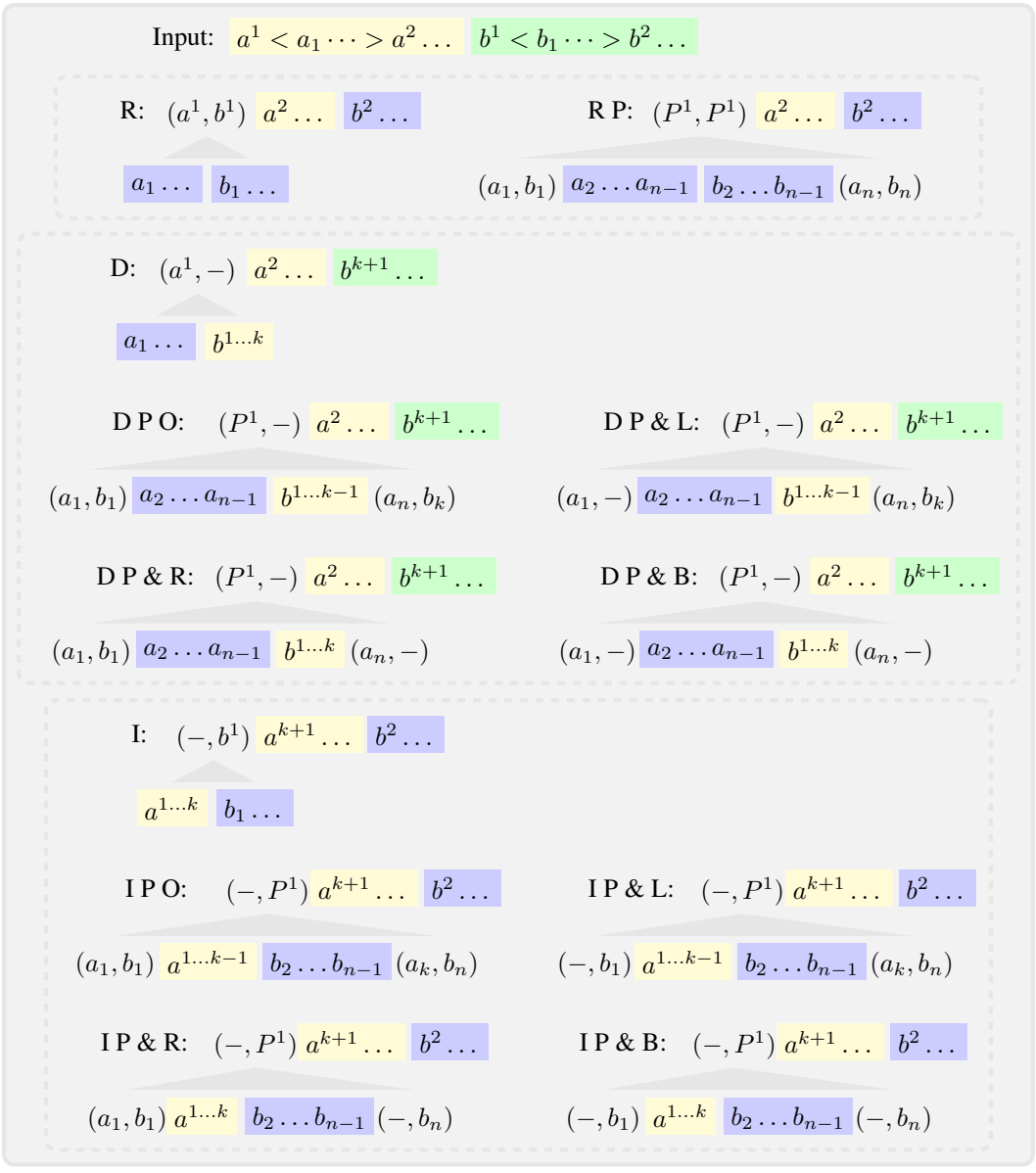


Figure 9.6: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-node. Fourth case of the case distinction.

Case 5:

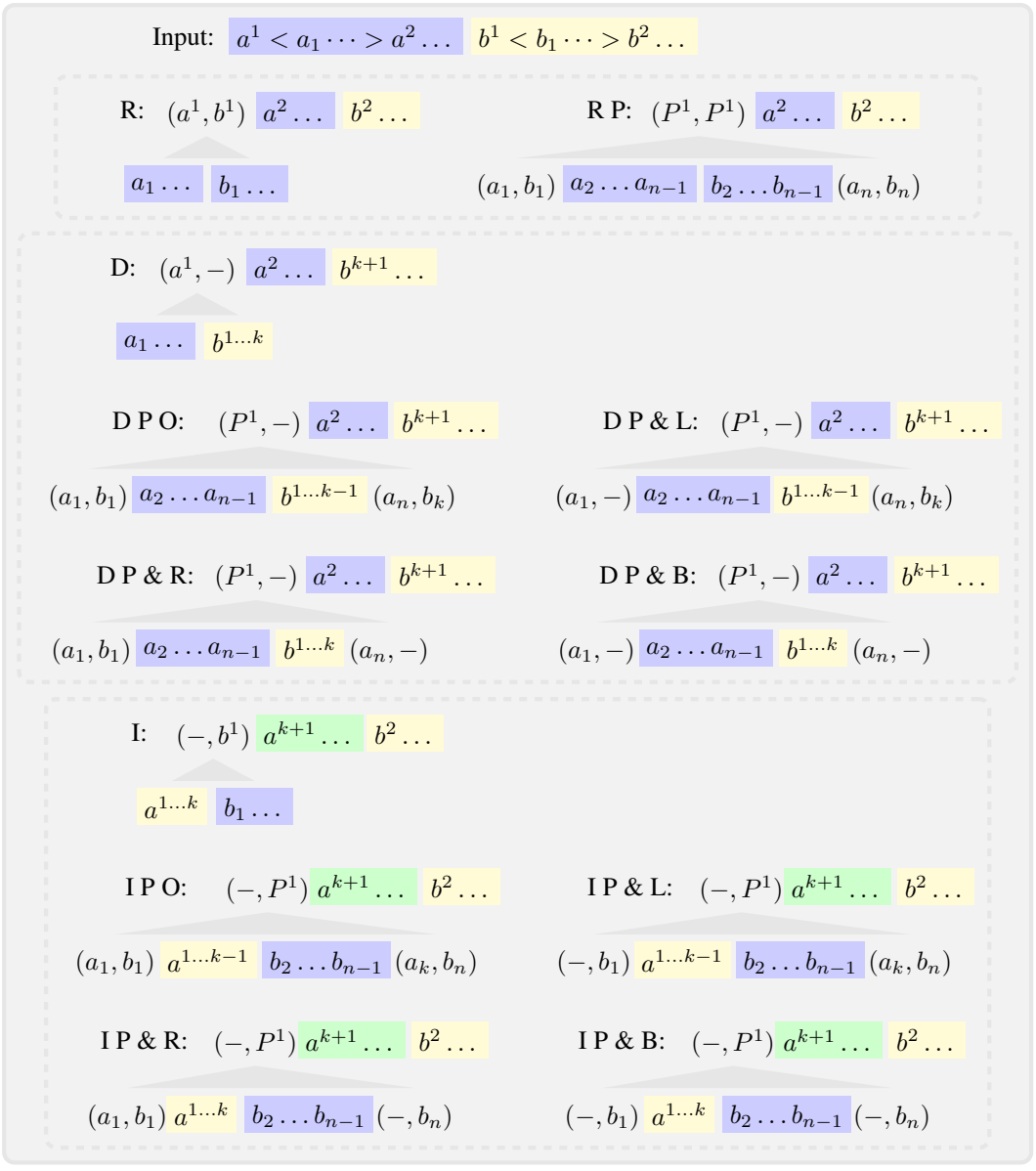


Figure 9.7: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-node. Fifth case of the case distinction.

Case 6:

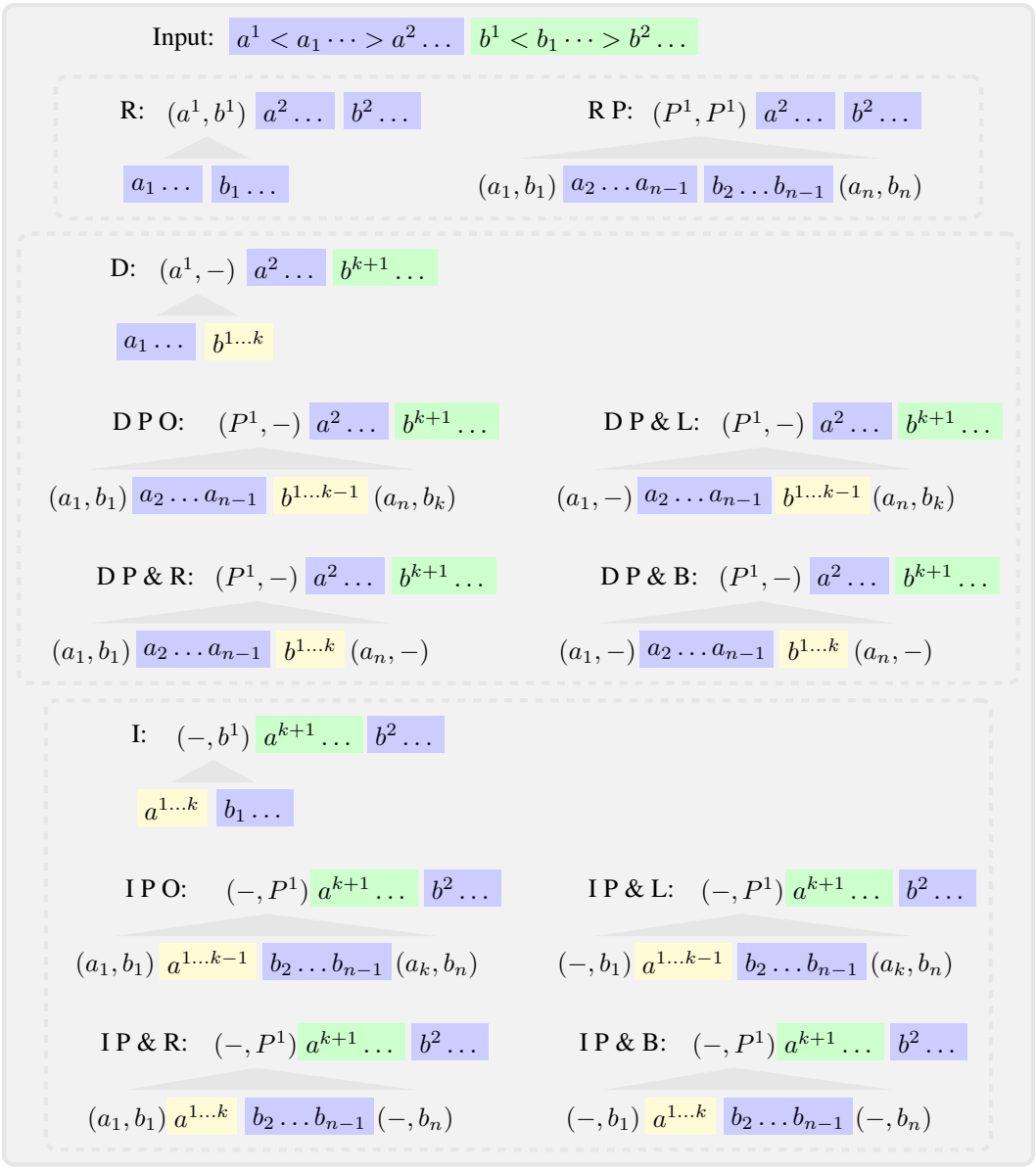


Figure 9.8: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-node. Sixth case of the case distinction.

Case 7:

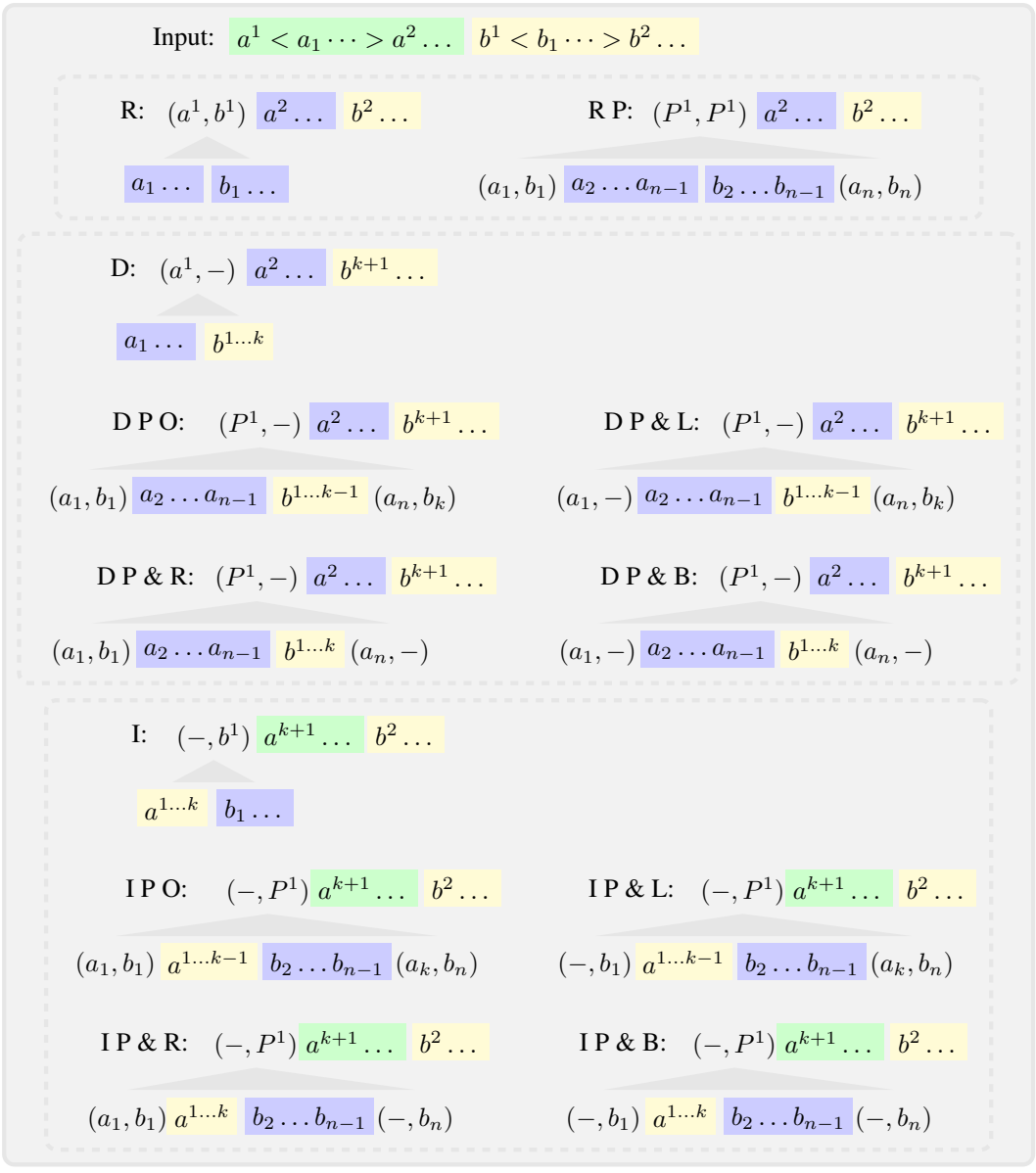


Figure 9.9: Case distinction for the tree alignment algorithm with affine gap costs, with special operations for P-node. Last part of multipart figure.

We want to be able to describe meaningful, non-atomic edit operations explicitly, depending on the meaning of the trees to be aligned. Then, a generalized tree alignment algorithm should be based on these edit operations.

In this way, we could avoid unwanted cases such as in Figure 9.1 by giving a rule stating that a and b are not matched, and the implied alignment is the middle one, where b stays with its P-node, while a becomes a $(a, -)$ (or (a, c), which also makes sense).

Explicit rules would also allow us to prohibit alignment nodes such as (a, P), which are legal in the general tree edit model, but do not have a meaning in our semantically richer application context of RNA secondary structure alignment.

Our present implementation avoids this situation in an ad-hoc manner via the score function. The pollution of the search space with meaningless candidates as explained above remains a drawback. As a general model, tree alignment parameterized by explicit matching rules appears a well-motivated challenge for future research in combinatorial pattern matching.

10 RNAforester 2.0

The program *RNAforester* implements the forest alignment algorithm with linear gap scoring [43, 1]. It is available via the *Vienna RNA Package* [45], see also Section 3.5.1. See [43] for a detailed analysis of the program.

The described new algorithms with affine gap scoring and speedup by anchoring are implemented in the program *RNAforester 2.0*. The program is available at

`http://bibiserv.cebitec.uni-bielefeld.de/rnaforester2/`.

Like the original *RNAforester*, it computes pairwise alignments for a global, local and small-in-large structural alignment problems. It computes multiple alignments in a progressive way, and uses a two stage index mapping for efficient tabulation of intermediate results.

10.1 Implementation

When solving the forest alignment problem, several different variants of the problem are possible. The parsing and alignment problem variants depends on the following decisions:

- Parse different types of input file formats (Fasta, RNAcast output, XML multiple structure format).
- Parse the input trees with or without P-nodes. Different parsing methods have to be used for character forests, e.g. in Newick-like notation, and for dot-bracket RNA structures.
- Use special cases when aligning the P-nodes, such as the replacepair-operation. These operations are meaningful for RNA forests only.
- Compute distance or similarity.
- Use an affine or a linear gap scoring scheme.
- Compute a global, local, or small-in-large alignment.
- Do a pairwise or multi structure comparison.

These decisions can be specified via command line parameters, which we will discuss in Section 10.1.1.

10.1.1 Options and command line parameters

Several options can be used to specify the alignment problem for *RNAforester* and to choose the correct variant of the forest alignment algorithm.

The number of available options depends on the availability of *libRNA*, *libgd*, *libg2*, *libXML2* and *libXML++*, which are needed to layout and draw the resulting RNA consensus structure. New options for new features in *RNAforester 2.0* are indicated by a different color. The option names have not been changed to ensure backwards compatibility to previous versions of *RNAforester*.

General Options The following options influence the general nature of the program. The scale is a ruler intended to help the user to keep track of the position in the sequence and structure while making his or her input, as known from other bioinformatics programs with interactive command line input.

`--help` produces a help message

`--version` shows program version

`-f=filename` sets the input file, and disables reading from standard input, which is the default

`--tables` shows dynamic programming tables

`--backtrace` shows backtrace call table cells

`--score` compute only scores, no alignment

`--noscale` suppress output of scale

Computation Properties The way of computation can be set by the following parameters.

`-t` *calculate alignment top down instead of bottom up*

`-m` multiple alignment

`--anchor` *use shape anchoring for speedup*

`--pIndels` *enable extra pair indel cases (see Chapter 9)*

Score Properties The score can be computed in several ways, depending on the posed alignment problem.

`-a` *compute affine score (default: non-affine)*

`-l` local similarity (default: global)

`-so=int` local suboptimal alignments within int%

`-s` small-in-large similarity (default: global)

`-d` calculate distance instead of similarity

`-r` calculate relative score

`--RIBOSUM` RIBOSUM85-60 scoring matrix (for base-pair substitutions)

Score Values The scoring scheme, as used by the alignment algorithms, can be provided by the user. The overall computation of the result depends on the score properties (distances are minimized and scores are maximized), and on the scores for the basic edit operations.

If an affine scoring scheme is used, the user can supply scores for the additional pair gap opening operation and base gap opening operation. The usual pair indel and base indel scores are used as gap extension costs in this case.

-pm base pair match

-pd base pair indel

-bm base match

-br base replacement

-bd base indel

-pdo *open basepair indel (only if affine scoring is used)*

-bdo *open base indel (only if affine scoring is used)*

Progressive Alignment In multiple alignment mode, *RNAforester* computes a progressive profile alignment. In this mode, the input structures are translated into profile forests, which have at each node a vector of frequencies for each base and frequencies whether we encountered a pair or a base in that position. These profiles are aligned in a progressive way by computing the scores between all input forests, and then joining the profiles with the highest score in the pairwise comparison in an iterative fashion to achieve a clustering of the input structures.

The following parameters influence the progressive alignment of multiple structures.

-p predict profile from sequences (only if compiled with libRNA)

-pmin=double minimum basepair frequency for prediction (only if compiled with libRNA)

-sp save profile

-ps profile search

-cbmin *minimum base frequency for consensus structure*

-cmin - minimum pair frequency for consensus structure

-mt Clustering threshold

-mc Clustering cutoff

Output format options *RNAforester* allows the user to choose between different output formats. By default, the output is suitable to be displayed on the system console, while at the same time readable for the human eye. For computational analysis, an output in fasta and XML format is available, and for expert analysis or publication of alignment results a graphical output is provided.

`--fasta` fasta output format with gaps

`-2d` generate alignment 2D plots in postscript format (if compiled with libg2)

`--xml` generate XML file in RNAStructAlignmentML format (if compiled with libRNA, libXML++, libXML2)

`--xml_output` name of XML output file (if compiled with libRNA, libXML++, libXML2)

Squiggle Plot Options To influence the outcome of the graphical output of *RNAforester*, the following options are provided.

`-2d_hidebasenum` hide base numbers in 2D plot (if compiled with libg2)

`-2d_basenuminterval=n` show every n-th base number (if compiled with libg2)

`-2d_grey` use only grey colors in 2D plots (if compiled with libg2)

`-2d_scale=double` scale factor for the 2d plots (if compiled with libg2)

`-2d_fig` generate additional fig file of 2d plot (if compiled with libg2)

`-2d_png` generate additional png file of 2d plot (if compiled with libg2 and libgd)

`-2d_jpg` generate additional jpg file of 2d plot (if compiled with libg2 and libgd)

Via these options, the correct alignment algorithm variant of the algorithms we introduced in Chapter 5, 6 and 7 is selected.

11 Applications

We have presented new algorithms for forest alignment with affine gap costs, as well as a method to speed them up by anchoring of substructures.

To show where the new algorithms can be applied in practice, we discuss two application scenarios for the new algorithms implemented in our software tool *RNAforester 2.0*.

Both applications try to solve the problem of constructing a structural consensus from a number of related input sequences.

The first application is an approach to improve structural alignments based on a new combination of existing methods, and is called *planACstar*. The method has been developed together with the Bachelor student Andreas Bremges, and is published in [16]. In this scenario, the affine algorithm variant can be used to improve the performance of the method by improving the structural alignment step that is involved in it.

The second application is an approach that uses an alternative concept of structural consensus, based on the idea of abstract shape analysis. The program RNAcast [109] can be used to "cast" a family of structures into a common shape (consensus shape). Combined with *RNAforester*, we may look for a consensus shape whose representative folded structures are most alignable with *RNAforester*, to get a reasonable consensus as a result. Again, we use *RNAforester* in this pipeline, and applying the affine gap cost model can improve the alignment step. In addition, all structures that have to be aligned in this step share by concept of RNAcast the same overall shape, which makes it possible to apply the anchored variant for speedup.

11.1 Algorithms for RNA 2D structure prediction

Many algorithms for RNA 2D structure prediction are based on a single input sequence. Comparative structure prediction predicts a consensus structure for a family of input sequences. In practice, comparative structure prediction tends to perform better than single sequence prediction [31], but an ensemble of multiple related structures is not always available.

An important approach that combines comparative structure prediction with thermodynamic folding is the Sankoff algorithm [99]. While it computes the exact optimal solution of both the alignment and the folding optimization problems at the same time, it has the drawback of high computational costs of $\mathcal{O}(n^6)$ time and $\mathcal{O}(n^4)$ space.

Thus, several algorithms have been proposed that either focus on part of the problem, or introduce pragmatic restrictions and solve the problem in a heuristic way. Two examples which attack parts of the problem are the sequence alignment folding program RNAalifold and our structure alignment program *RNAforester*. Other alternatives to the Sankoff algorithm impose constraints and restrictions, such as Dynalign, Foldalign and LocARNA [79, 40, 132].

11.2 Problem: from a family of sequences to a structural consensus

The following sections sum up the ideas from [16]. The structural consensus for a family of RNA molecules can be computed according to three different "plans"[31]: We can (A) align the sequences first, and then fold them, or (B) simultaneously align and fold the sequences, or (C) first fold the sequences individually, and afterwards align their structures.

Plan B gives a theoretically optimal solution, because it solves the two optimization problems of alignment and folding at the same time [99]. The downside are the high computational costs of $O(n^{3m})$ time and $O(n^{2m})$ space, where n is the sequence length and m is the number of sequences.

Plan C can be used for sequences that are too diverged to be meaningfully aligned based on sequence conservation. First, sequences are folded individually. Then, a representative subset of near-optimal structures must be obtained for each sequence, for example by abstract shape analysis [34]. Those structures which have a consensus abstract shape are aligned, for example via tree comparison (*RNAforester* [43, 44], *MARNA* [106]). *RNAcast* [91] was the first Plan C approach.

Plan A is suited for sequences that can be meaningfully aligned using a multiple sequence alignment tool (e.g. *ClustalW* [22], *T-Coffee* [89], *MAFFT* [57]). The sequence alignment is then folded (e.g. *PFOLD* [64], *RNAalifold* [47, 46], *ILM* [97], *ConStruct* [133]). It is the most common of the three approaches.

11.3 Plan ACstar: Improving structural alignments in the twilight zone

The method *planACstar* which we discuss here is a fine-tuning step that can be combined with Plan A methods. Plan A is not useful when sequence conservation is above 90%, because the sequence alignment does not add much extra information compared to individual folding of the sequences. It performs best at about 75% of sequence similarity. Measurements [32] show a decline of performance below 55%, which was confirmed in [126, Figure 4].

We called the new method to improve Plan A *planACstar*, because it iterates over combined steps from Plan A and Plan C.

Idea

With Plan A and sequences between 30-55% identity, the found consensus structure may often be correct in its overall shape while structural detail gets lost due to the obscured covariance signal in the sequence alignment step. As a remedy, we disassemble the alignment and refold the sequences separately, with respect to the preliminary consensus.

Additional base pairs compatible with the consensus may be introduced in this separate folding step. The resulting structures are aligned with a *structure* alignment program such as *RNAforester*. It aligns base pairs irrespective of the particular bases, detecting compensatory base changes and recovering the covariance signal. Any structure alignment obtained in this way, comprises a sequence alignment, which we extract. This sequence alignment may now fold into a better consensus than before.

Note that we may recover base pairs missed in the first alignment folding, but we never remove consensus base pairs which were formed in the initial step. These base pairs will always remain in the improved structures, although they may be aligned in a different way.

Based on the tools *ClustalW*, *RNAalifold*, and *RNAforester*, the *planACstar* pipeline was implemented in [16]. The quality of the consensus is assessed by *RNAalifold*'s structure conservation index (SCI), which according to Gruber et al. [37] is the best measure for structural conservation.

If alignment A is folded with *RNAalifold*, it results in the minimum free energy (MFE) E_A of the consensus as output. The MFE E_A includes pseudo-energy contributions from observed covariation. Folding the sequences $B_1, \ldots, B_n$ separately, we can compute their average MFE $\bar{E}$ from the separate MFEs. The SCI is the quotient of $E_A/\bar{E}$. Thus, the improvement is measured in terms of *RNAalifold*'s own quality criterion, the SCI.

Implementation

The method *planACstar* is a pipeline that combines existing tools. In [16], we used *ClustalW* for the initial alignment step, and *RNAalifold* for folding the sequences into the consensus structure, because they are most widely used tools for these tasks in practice.

In pseudocode, the procedure is as follows:
Given a set of RNA sequences $B_1, ..., B_n$,

1. $A \leftarrow$ *ClustalW*$(B_1, ..., B_n)$ – initial sequence alignment
2. $C \leftarrow$ *RNAalifold*(A); $X \leftarrow$ SCI-score(C) – preliminary consensus
3. $C_i \leftarrow$ Projection of basepairs in C to B_i – (see below)
4. $S_i \leftarrow$ *RNAfold* -C $C_i(B_i)$ for $i = 1, ..n$ – individual folding of each B_i relative to consensus
5. $Y \leftarrow$ *RNAforester*$(S_1, ..., S_n)$ – multiple structure alignment
6. $A^* \leftarrow$ sequence alignment implied by Y – extract improved sequence alignment
7. $C^* \leftarrow$ *RNAalifold*(A^*); $X^* \leftarrow$ SCI-score(C^*) – fold improved alignment
8. if $X^* > X$, set $A \leftarrow A^*$ and iterate from step 3, else exit with result A, C and X

Step 3, the projection of basepairs in C to B_i, takes the base pairs from the consensus, removes the gaps with respect to sequence B_i, and results in structure C_i for B_i. Step 4, the call to *RNAfold* with option $-C\ C_i$, may produce additional base pairs aside from those of C_i, which will in general be different for each B_i. See Figure 11.1 for a graphical outline of the method.

The pipeline combines the existing programs in a modular way, and each component in it can be exchanged for alternative programs that solve the same task.

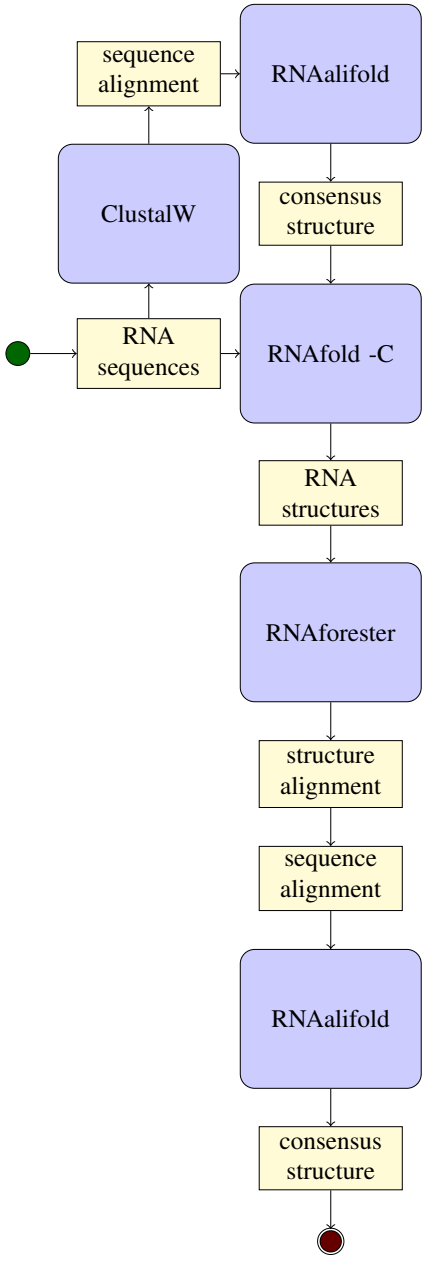


Figure 11.1: The *planACstar* pipeline design.

11.3.1 Results and discussion

If we substitute *RNAforester* by *RNAforester 2.0* with the affine gap scoring variant in the original pipeline, we are able to increase the SCI even further. See 11.2 for the overall results comparing the original *planACstar*, the affine variant of *planACstar*, *plan A*, and the *reference alignments*, and 11.3 for the original data.

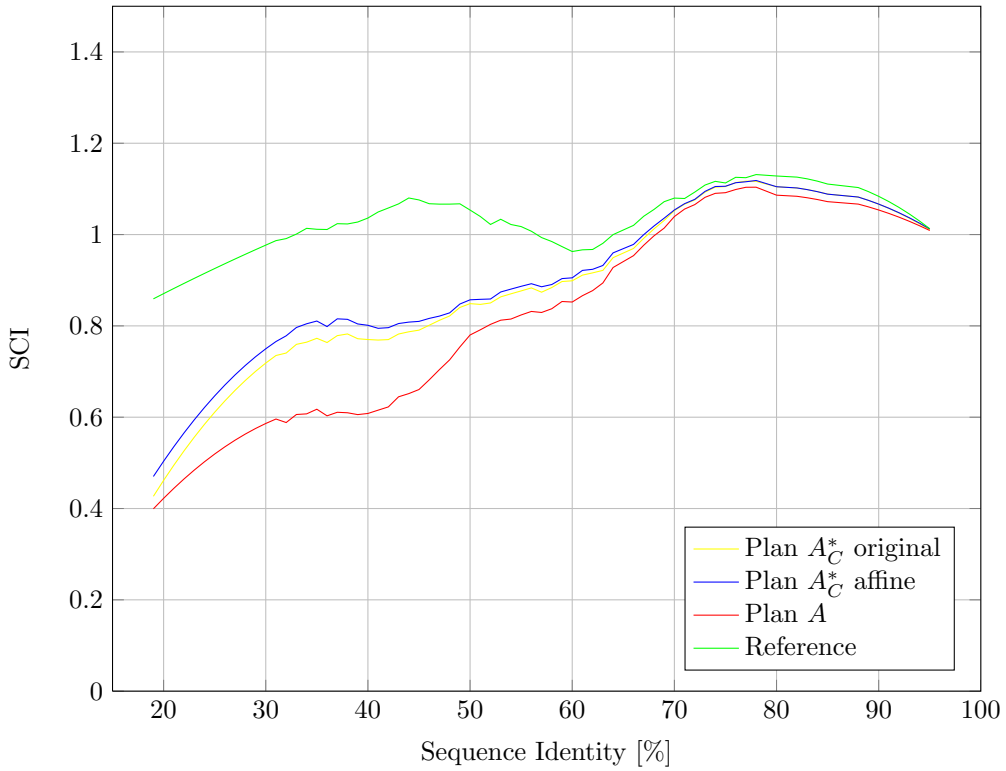


Figure 11.2: Structure conservation with the linear and affine variant of *RNAforester* in *planACstar*, in comparison to plan A and the reference alignments. The scores for the linear variant are the default scores of *RNAforester*, and the scores for the affine variant are Set 2.1: pair match: 10; pair indel open: -10; pair indel: -5; base match: 1; base replacement: 0; base indel open: -20; base indel: -10; The results are highly scattered, and a Savitzky-Golay filter was applied to get smooth functions. See Figure 11.3 for the original data.

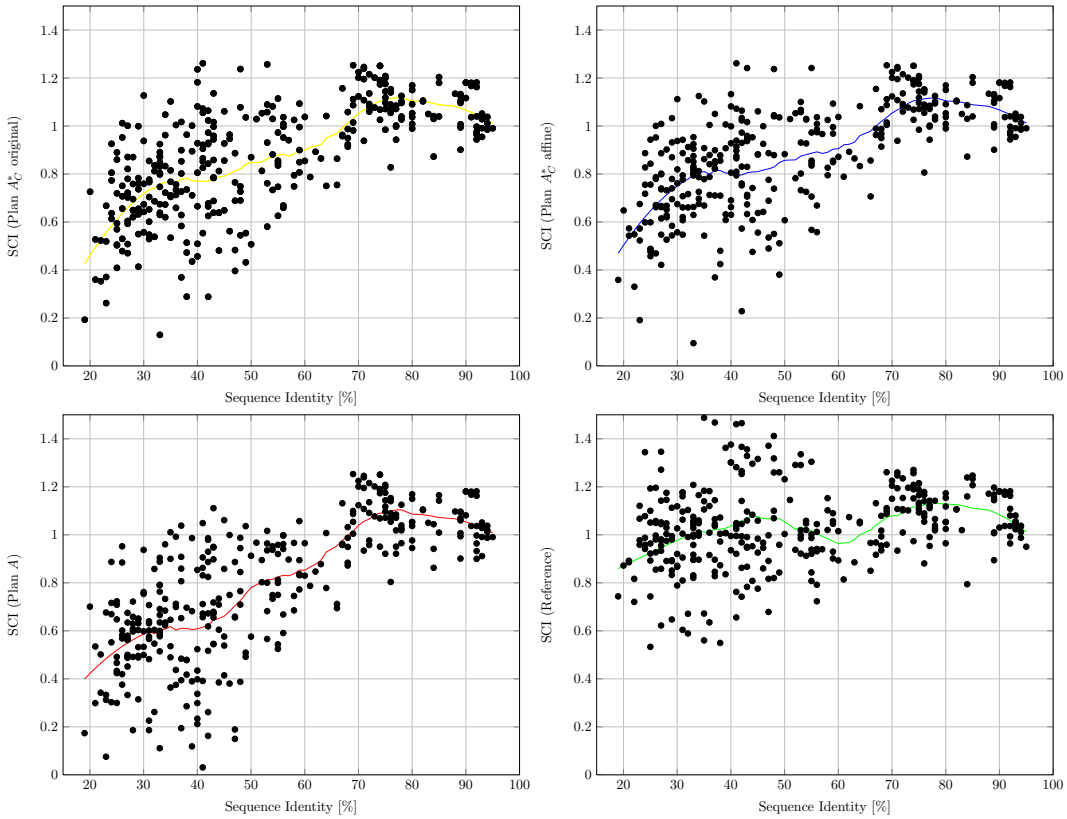


Figure 11.3: Structure conservation with the linear and affine variant of *RNAforester* in *planAC-star*, in comparison to plan A and the reference alignments. The scores for the linear variant are the default scores of *RNAforester*, and the scores for the affine variant are *Set 2.1:* pair match: 10; pair indel open: -10; pair indel: -5; base match: 1; base replacement: 0; base indel open: -20; base indel: -10; Some individual alignments have the same SCI under different parameter sets. See Figure 11.2 for the smoothened graphs in one plot.

For different parameter sets, we get a different amount of improvement in terms of the SCI as the pipeline result. We gradually change the gap opening parameters and leave the gap extension parameters fixed. See 11.4 for the overall result for the affine variant of *planACstar* with the different parameter sets, and 11.5 for the original data. Parameter set 2.1 performs best improving the SCI. Setting the gap opening costs even higher than in 2.1, like in parameter set 2.2, results in a lower SCI.

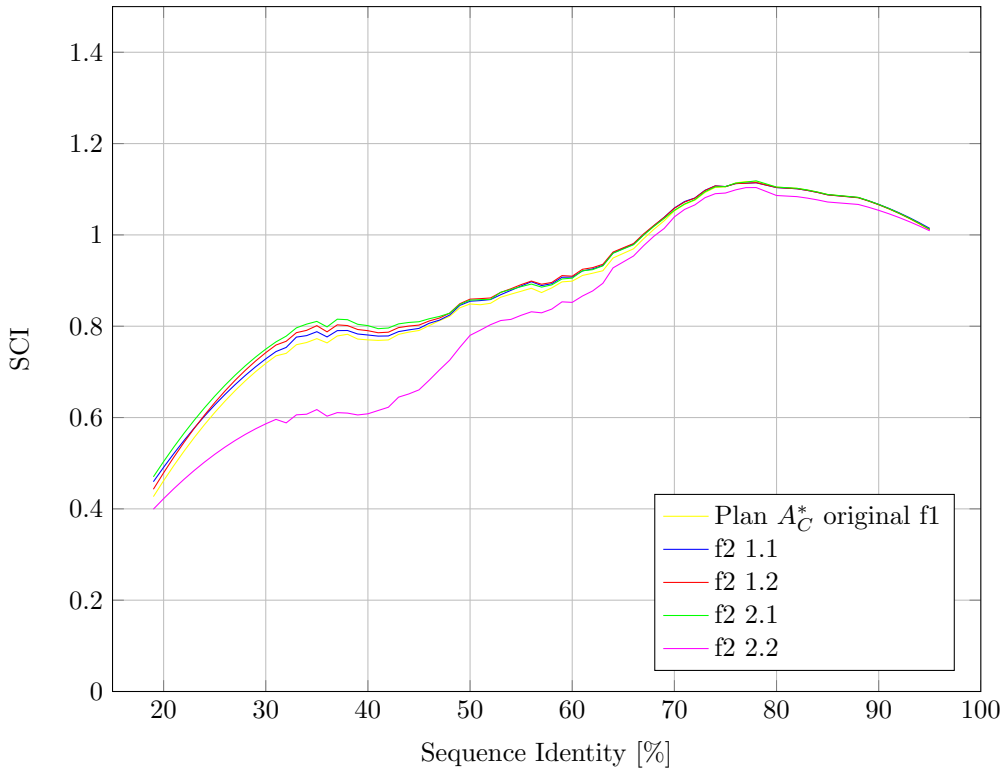


Figure 11.4: The affine variant of *RNAforester* in *planACstar*. The scores for the open=extend variant are the *default scores* of *RNAforester*. pair match: 10; *pair indel open: -5;* pair indel: -5; base match: 1; base replacement: 0; *base indel open: -10;* base indel: -10; The affine variant was tested with three more sets of scoring parameters: *Set 1.1:* pair indel open: -6; base indel open: -11; other parameters are default parameters; *Set 1.2:* pair indel open: -7; base indel open: -12; other parameters are default parameters; *Set 2.1:* pair indel open: -10; base indel open: -20; other parameters are default parameters; *Set 2.2:* pair indel open: -15; base indel open: -30; other parameters are default parameters; The results are highly scattered, and a Savitzky-Golay filter was applied to get smooth functions. See Figure 11.5 for the original data.

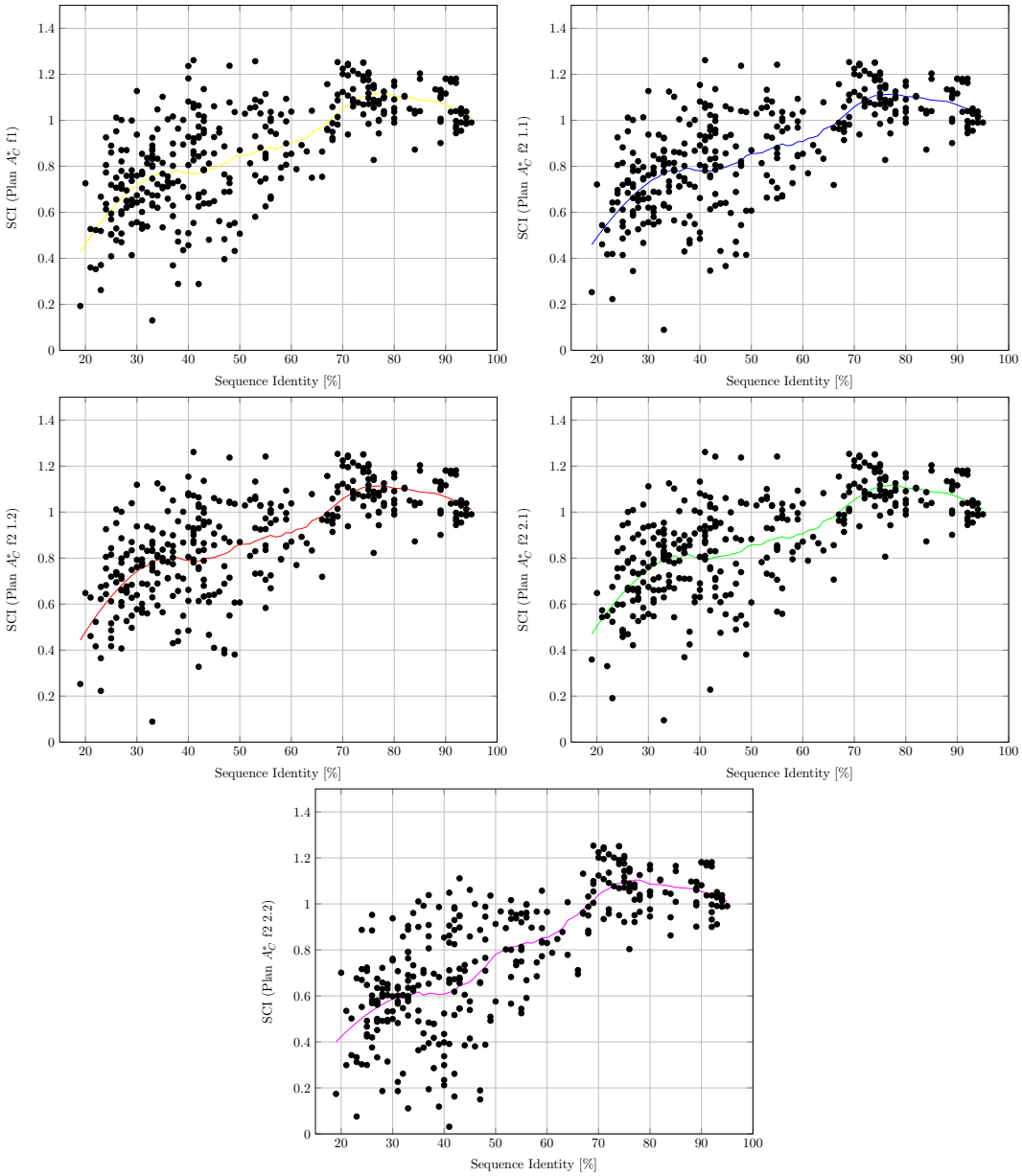


Figure 11.5: The affine variant of *RNAforester* in *planACstar*. The scores for the open=extend variant are the default scores of *RNAforester*. pair match: 10; pair indel open: -5; pair indel: -5; base match: 1; base replacement: 0; base indel open: -10; base indel: -10; Set 1.1: pair indel open: -6; base indel open: -11; other parameters are default parameters; Set 1.2: pair indel open: -7; base indel open: -12; other parameters are default parameters; Set 2.1: pair indel open: -10; base indel open: -20; other parameters are default parameters; Set 2.2: pair indel open: -15; base indel open: -30; other parameters are default parameters; See Figure 11.4 for the smoothened graphs in one plot.

11.4 RNAforecast as an alternative to the Sankoff algorithm

The idea of the RNAcast approach is that related structures share the same overall shape for similar function. RNAcast allows to "cast" RNA sequences into a number of common shapes, where the shape representative ("shrep") of each sequence and shape is the MFE structure folding into the given shape.

This abstract shape represents only some general building blocks of the structure, but abstracts from particular features such as the position or the length of a certain region.

The shape representatives for each shape are a number of unaligned secondary structures, which share the same abstract shape. The resulting shreps from the RNAcast approach can be aligned with an RNA structure alignment program such as *RNAforester*. From this structure alignment, a sequence alignment consistent to the consensus shape can be computed. In this way, the procedure can be used as an alternative to Sankoff algorithm, which aligns and folds a set of RNA sequences simultaneously.

The common structures can be ranked by the sum of their position numbers from the initial enumeration step. The top ranking pair is the consensus predicted by RNAcast.

This method has a major drawback: If the sequences differ in length, their structures cannot be identical. We need a more flexible method of comparing structures, and it would be useful to restrict the enumeration to a representative sample of the folding space.

These two problems may be solved with the idea of abstract RNA shape analysis. The method was first described in "Abstract Shapes of RNA" [34]. Using abstract shapes, the RNAcast approach enumerates one representative structure per shape for the input sequences. The highest ranking structure tuple, where all structures share the same shape, is the consensus.

The program RNAcast A *common shape* of $\{s_1, \ldots, s_k\}$ is a shape p for which $p \in \bigcap_{i=1}^{k} \mathcal{P}(s_i)$. For sequences $\{s_1, \ldots, s_k\}$, the common shape p that minimizes $rank(\hat{p}_1, \ldots, \hat{p}_k)$ is called the *consensus shape*. In this case, *rank* is a scoring function on the individual shrep scores.

The idea which has been discussed above is implemented in the tool RNAcast. RNAcast is an acronym for RNA consensus abstract shapes technique. Algorithmically, three steps become apparent.

- Input sequences are $s_1, \ldots s_k$, energy range R, and n is the average length of sequences. We run RNAshapes on each sequence with energy range R.

- We get k result lists or shape spaces. Which shapes are in all lists? We use hashing techniques for fast matching of shapes in time $k * n * |(P)(s_1)|$: iterate over first list $(P)(s_1)$, use hash table to look for same shape in other lists. Result is a list of the l common shapes and their shreps. $(p_1, [\hat{p}_1^1, \ldots, \hat{p}_k^1]), \ldots, (p_l, [\hat{p}_1^l, \ldots, \hat{p}_k^l])$.

- Score the common shapes with a scoring function, getting a sorted list of common shapes. The first shape of this list is the consensus shape with the shreps for the structures. The $r \leq l$ best common shapes may be returned on request.

The scoring function "rank" What about scoring function "rank"? Reeder proposes four definitions.

- Rank sum - each shape contributes with its original rank in the shape space of its sequences
  $rank_1(p_i, \hat{p}_1^i, \ldots, \hat{p}_k^i) = rank(\hat{p}_1^i) + \cdots + rank(\hat{p}_k^i)$
- Sum of energies
  $rank_2(p_i, \hat{p}_1^i, \ldots, \hat{p}_k^i) = E(\hat{p}_1^i) + \cdots + E(\hat{p}_k^i)$
- Normalized sum of energies
  $rank_3(p_i, \hat{p}_1^i, \ldots, \hat{p}_k^i) = \frac{E(\hat{p}_1^i)}{E(mfe(s_1))} + \cdots + \frac{E(\hat{p}_k^i)}{E(mfe(s_k))}$
- Sum of probabilities - structure probabilities from partition function [83]
  $rank_4(p_i, \hat{p}_1^i, \ldots, \hat{p}_k^i) = Prob(p_1^i) + \cdots + Prob(p_k^i)$,
  where the probability of a shape p_s^i is the sum of all foldings of sequence s that take shape i.

The function $rank_2$ is the default scoring function in RNAcast, as it proved to be most robust in practice.

Time and space complexity of RNAcast Reeder analyzes the asymptotic efficiency of RNAcast. The time complexity of enumeration phase is $\mathcal{O}(k \cdot n^3 + |\mathcal{P}(s_1)| + \cdots + |\mathcal{P}(s_k)|)$. The comparison phase takes $\mathcal{O}(k \cdot n \cdot |\mathcal{P}(s_1)|)$, as the algorithm iterates over first list and looks up co-occurrences with the help of a hash table. Scoring and sorting takes $\mathcal{O}(k \cdot l + l \cdot logl)$ time, where l is the number of common shapes. The enumeration dominates the computation time.

Runtime analysis on real world examples shows that the efficiency of RNAcast is quite good, as expected, and it depends on the runtime of RNAshapes.

Differences to Sankoff's consensus notion Note that *RNAcast* is no other Sankoff heuristic, but rather it changed the problem definition. In the Sankoff algorithm, a sequence alignment is sought which reflects common sets of basepairs. With Consensus Shapes, a consensus shape and a shrep for each sequence are computed, but no alignment. Thus, it is important to discuss the similarities and differences to the traditional and accepted notion.

In Reeder's RNAcast approach, the predictions are unaligned. But the predicted shreps may be aligned with tools like *RNAforester* as a multiple structure alignment. From this structure aliment, a sequence alignment can be derived. The structure alignment minimizes the number of gaps, but this is done *after* structure prediction with RNAcast. Due to this fact, one may expect that Sankoff fixes the positions of helices more strongly, whereas *RNAcast* lets them move more flexibly. This effect was not observed in Reeder's studies.

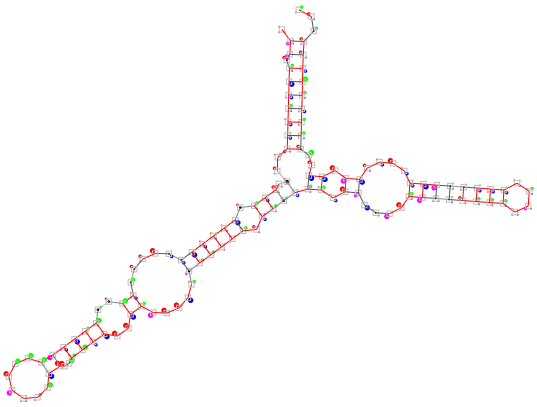
RNAforester comes into play: The RNAforecast pipeline

From Reeder's evaluations, it can be observed that the consensus shape approach predicts the majority of annotated basepairs. To mimic the usual Sankoff result, he suggests to take the unaligned consensus shreps, and use them as input for a structural alignment with *RNAforester*.

We pipe the output of *RNAcast* as input into *RNAforester*. The unaligned *RNAcast* output is a list of shreps for each common shape.

```
1) Shape: [[][]]  Score: -203.34  Ratio of MFE: 0.98
> Haloferax volcanii
        UUAAGGCGGCCAGAGCGGUGAGGUUCCACCCGUACCCAUCCCGAACACGGAAGUUAAGCUCACCUGCGUUCUGGUCAGUACUGGAGUGAGCGAUCCUCUGGGAAAUCCAGUUCGCCGCCCCU
-42.30  ....((((((..((((...((((.((((((((((....((.((((.(((.((.....)).)...))).)))).)).)...)))))).)).)))...)).)))))..((....)).))))))))))...  [[][]]  R = 2
> Arthrob.globiformis
        AUUACGGCGGUCAUAGCGUGGGGGAAACGCCCGGUCCCAUUCCGAACCCGGAAGCUAAGACCCACAGCGCCGAUGGUACUGCACCCGGGAGGGUGUGGGAGAGUAGGUCACCGCCGGACAC
-47.42  ....(((((((....(((((((......((((((.............))))))..))).....))))))).)).((.((....(((((((((....)))))))))....)).)).)).)))))).....  [[][]]  R = 4
> LT50091 Mycobacterium phlei
        GUUACGGCGGUCAAUAGCGGCAGGGAAACGCCCGGUCCCAUCCCGAACCCGGAAGCUAAGCCUGCCAGCGCCAAUGAUACUACCCUUCCGGGUGGAAAAGUAGGACACCGCCGAACAu
-45.22  (((.(((((((.....(((((((.....((((((.............))))))..))....)))))))).)).((.((....(((((((....)))))))....)).)).)))))))..  [[][]]  R = 1
> Lactobacillus plantarum (Firmicutes, Clostridiobacteria)
        UGUGGUGACGAUGGCGAGAAGGAUACACCUGUUCCCAUGUCGAACACAGAAGUUAAGCUUCUUAGCGCCGAGAGUAGUUGGGGGAUCGCUCCCUGCGAGGGUAGGACGUUGCCAUGC
-35.20  .((((..(((..((((..(((((.((..(((((((.......))..))))).))).....))))))..))))).......((.((((((....))))))..))..........)))..)))))..  [[][]]  R = 8
> Porphyromonas gingivalis (Cytophagales)
        CUCAGGUGGUUAUUACGUUGGGGAUCCACCUCUUCCCAUUCCGAACAGAGAAGUUAAGCCCAACGGUGCCGAUGGUACUGCGUUAUAGUGGGAGAGUAGGACGCCGCCGCC
-33.20  ....((((((..........((..(((..(((.(((((((...(((......))).......(((((((...)))))))......))))))))))..)))..))))))))  [[][]]  R = 2
```

From the unaligned output, we may compute a structural alignment of the shreps (here in graphical representation):



This idea was evaluated on the BRAliBase (Benchmark RNA Alignment dataBase) [32], the dataset of the first comprehensive RNA alignment benchmark, that is based on Rfam [36]. The database provides pairwise and multiple alignments with a mean pairwise sequence identity of $10 - 100\%$. As a performance measure, the authors of the Bralibase study used 1) the sum-of-pairs-score (SPS) to reflect the number of correctly predicted basepairs, and 2) the structure conservation index (SCI), to account for conserved structural information.

RNAforecast *RNAforecast* uses information from structural alignments to select better consensus shapes from the list of common shapes ("consensus shapes which are most alignable").

This assignment is formalized in another rank function.

- *RNAforecast* score: linear combination of RNAcast and relative *RNAforester* score, parameter α balances both scores $rank_5(p_i, \hat{p}_1^i, \cdots, \hat{p}_k^i) = \alpha \cdot \frac{rank_2(p_i, \hat{p}_1^i, \cdots, \hat{p}_k^i)}{\sum_{j=1}^{k} E(mfe(s_j))} + (1-\alpha) \cdot forester\_score(\hat{p}_1^i, \cdots, \hat{p}_k^i)$

Here, the first summand is weighted by α: $rank_2$, the function we already defined, and is normalized by the sum of MFEs to get a value from the interval $[0 \ldots 1]$. The function $rank_3$ with appropriate normalization would also be possible. The second summand is weighted by $1 - \alpha$ is the relative *RNAforester* score, which has an upper bound of 1, and no lower bound. For additional details on the relative score of *RNAforester*, see [42].

Computing a multiple structural alignment adds computation time. Recall that computing a single alignment of k structures with maximum sequence length n takes $\mathcal{O}(n^2 d^2 k^2)$ time, where d max degree of a tree node (compare Section 6.14). The runtime of the forest alignment algorithm does not scale linearly, but quadratically.

In practice, we thus have to restrict the alignment step to the best ten consensus shapes, according to the *RNAcast* score. With this restriction, *RNAforecast* is still much faster than *foldalignM*, and in the same runtime range as *LocARNA*.

One may now ask whether this extension of RNAcast by the forest alignment step is justified by improved predictions. Reeder shows in his study that this is the case. Both SPS and SCI are lifted to Sankoff-style alignment tools again, where the SCI is higher than the SPS. Note that the Bralibase Dataset II contains only pairwise alignments, whereas both *RNAcast* and *RNAforecast* can handle multiple alignments as well.

In terms of related work, Reeder mentions that *RNAforecast* is conceptually similar to MARNA [105]. Although the approaches are similar, there are also differences: MARNA proceeds as follows: In the first step, MARNA predicts a small set of structures for each sequence independently. These come from randomly sampled RNAs, and are generated by the probabilistic backtrace step of *RNAsubopt* or *RNAshapes*. In the next step, there is an all against all pairwise structural alignment. The scores of the pairwise alignment are used for a multiple alignment with t-coffee. For k sequences of length n, with E structures per sequence, MARNA has to compute $\mathcal{O}(E^2 \cdot k^2)$ structural alignments, each in $O(n^4)$ time. Thus, the number of structures per sequence E is set to a small constant, e.g. three, to keep the runtime practical.

In the extended consensus approach, the expensive structural alignment step is limited to structures with the same shape. For each sequence, we have only one structure for each shape (the shrep). That means the extended consensus approach has to compute by factor of E less structural comparisons. This reduction allows us to look deeper in the suboptimal shape (structure) space.

The underlying models for structural comparison also differ, as *RNAforecast*/*RNAforester* uses trees and MARNA uses arc annotated sequences to represent secondary structures.

The complexity class of the *planACstar* pipeline is determined by the highest complexity of all involved programs. In the discussed implementation, this is the complexity of *RNAforester*.

11.5 Performance of RNAforecast with RNAforester 2.0

The new RNA forest alignment algorithm with affine gap costs can be used to improve the comparison step in the *RNAforecast* approach.

The *RNAforecast* approach is in general another (presumably better) rank function for the consensus shapes of the RNAcast approach. To assess the quality of the *RNAcast* and *RNAforecast* approach with linear and affine gap costs, an evaluation as in [91] can be conducted.

Ideally, we compute the true shape from a known consensus structure of a set of related sequences.

1. Then, we compute the consensus shapes with RNAcast to find out the rank of the true consensus shape with the default ranking function of RNAcast, $rank_2$.

2. We then also align the shreps in a second step, to find out the consensus shapes that are most alignable (i.e. have a high alignment score). Sorted by a function of the alignment score, the true shape will get another rank.

3. The alignment step can also be carried out with the affine variant of *RNAforester*. We can assess the quality of the methods by analyzing the rank on which the true common consensus shape can be found in the result.

4. As all shreps that have to be aligned in the structural alignment step of the *RNAforecast* approach share the same overall shape, we can also compute the alignment with the shape anchoring speedup method.

In practice, this ideal evaluation is not feasible due to two reasons.

Within a family of related structures, the consensus structure is often well preserved (i.e. for Rfam family data). As the structures within one family are closely related (the families in the database are constructed due to common structural features), the top ranking consensus shape with RNAcast was the true consensus in all cases we observed.

Within structures from different families, it is not clear whether a meaningful common consensus shape exists. Although in theory there always is one, but this does not have to be meaningful if the structures are unrelated.

Still, two questions can be posed to shed further light on the performance of the *RNAforecast* method with the new alignment algorithm.

The first question is, what is the rank of the best alignment score? In what sense does it differ from the default RNAcast rank?

The second question is: can the *RNAforester* alignment in the structural alignment step of the *RNAforecast* pipeline be improved by the new alignment algorithm?

Rank of the best alignment score The rank that would be given by the best alignment score differs from the rank that is given by RNAcast. The best alignment score is not necessarily on rank one, but for example in the below case it is on rank seven.

```
RNAforester --score -m -f 11shapes.rnacast | grep Score

Parsing shape block 1) Shape: [[][]]  Score: -125.70  Ratio of MFE: 1.00
Score: 409.056
Parsing shape block 2) Shape: [[][[][]]]  Score: -121.16  Ratio of MFE: 0.96
Score: 349.25
Parsing shape block 3) Shape: [[][][]]  Score: -119.30  Ratio of MFE: 0.95
Score: 423.667
Parsing shape block 4) Shape: [[[][]][]]  Score: -118.40  Ratio of MFE: 0.94
Score: 419.75
Parsing shape block 5) Shape: [[][[][][]]]  Score: -112.56  Ratio of MFE: 0.90
Score: 396
```

```
Parsing shape block 6) Shape: [[][[[][]]][]]]  Score: -111.30  Ratio of MFE: 0.89
Score: 385.944
Parsing shape block 7) Shape: [[[][]][][]]  Score: -111.20  Ratio of MFE: 0.88
Score: 458.75
Parsing shape block 8) Shape: [[][][[][]]]  Score: -110.66  Ratio of MFE: 0.88
Score: 350.833
Parsing shape block 9) Shape: [[][[][]][]]  Score: -109.40  Ratio of MFE: 0.87
Score: 369.306
Parsing shape block 10) Shape: [[][][][]]  Score: -106.06  Ratio of MFE: 0.84
Score: 374.25
Parsing shape block 11) Shape: [[][[][][][]]]  Score: -102.80  Ratio of MFE: 0.82
Score: 372.472
```

The affine alignment changes the alignment scores, and also changes the order it imposes on the sequences. Different alignments get an advantage, which are supposedly more reasonable due to the affine gap cost model.

```
RNAforester --score -a -pdo=-10 -bdo=-20 -m -f 11shapes.rnacast | grep Score

Parsing shape block 1) Shape: [[][]]  Score: -125.70  Ratio of MFE: 1.00
Score: 409.778
Parsing shape block 2) Shape: [[][[][]]]  Score: -121.16  Ratio of MFE: 0.96
Score: 348.75
Parsing shape block 3) Shape: [[][][]]  Score: -119.30  Ratio of MFE: 0.95
Score: 420.778
Parsing shape block 4) Shape: [[[][]][]]  Score: -118.40  Ratio of MFE: 0.94
Score: 419.75
Parsing shape block 5) Shape: [[][[][][]]]  Score: -112.56  Ratio of MFE: 0.90
Score: 396
Parsing shape block 6) Shape: [[][[[][]][]]]  Score: -111.30  Ratio of MFE: 0.89
Score: 422.75
Parsing shape block 7) Shape: [[[][]][][]]  Score: -111.20  Ratio of MFE: 0.88
Score: 452.25
Parsing shape block 8) Shape: [[][][[][]]]  Score: -110.66  Ratio of MFE: 0.88
Score: 367.75
Parsing shape block 9) Shape: [[][[][]][]]  Score: -109.40  Ratio of MFE: 0.87
Score: 369.139
Parsing shape block 10) Shape: [[][][][]]  Score: -106.06  Ratio of MFE: 0.84
Score: 476
Parsing shape block 11) Shape: [[][[][][][]]]  Score: -102.80  Ratio of MFE: 0.82
Score: 395.25
```

A top down computation of the scores does not change them, it only affects the runtime of the algorithm as in most cases, less table entries have to be computed.

Using shape anchoring for speedup decreases the runtime significantly. See Section 12.4 for a large scale analysis of the speedup.

The anchored algorithm variant changes the scores a bit, as the algorithm computes less candidate solutions because the anchors are fixed on their counterparts in the other input forests. See Section 12.5 for an analysis of this score decline caused by the anchored alignment algorithm.

This should cause the score to get worse, but this effect is not directly visible when just looking at the scores. The score of the pairwise alignments in the progressive profile alignment approach does become worse due to the anchoring. If the pairwise distance is higher than the join cluster threshold value, an additional cluster is introduced, and if the score of a pairwise alignment is worse than a cutoff value, it is dropped altogether.

With the clustering of the multiple alignment approach, we tend to get worse pairwise alignment scores, and thus smaller clusters. The score of the first cluster is output in multiple alignment mode. It can get better with the anchored variant in comparison to the nonanchored variant, if the cluster is smaller.

```
RNAforester -r -t --anchor -m -f 11shapes.rnacast | grep Score

Parsing shape block 1) Shape: [[][]]  Score: -125.70  Ratio of MFE: 1.00
Score: 408.75
Parsing shape block 2) Shape: [[][[][]]]  Score: -121.16  Ratio of MFE: 0.96
Score: 366
Parsing shape block 3) Shape: [[][][]]  Score: -119.30  Ratio of MFE: 0.95
Score: 465.75
Parsing shape block 4) Shape: [[[][]][]]  Score: -118.40  Ratio of MFE: 0.94
Score: 436
Parsing shape block 5) Shape: [[][[][][]]]  Score: -112.56  Ratio of MFE: 0.90
Score: 396
Parsing shape block 6) Shape: [[][[[][]][]]]  Score: -111.30  Ratio of MFE: 0.89
Score: 424.75
Parsing shape block 7) Shape: [[[][]][][]]  Score: -111.20  Ratio of MFE: 0.88
Score: 457.75
Parsing shape block 8) Shape: [[][][[][]]]  Score: -110.66  Ratio of MFE: 0.88
Score: 386
Parsing shape block 9) Shape: [[][[][]][]]  Score: -109.40  Ratio of MFE: 0.87
Score: 402.25
Parsing shape block 10) Shape: [[][][][]]  Score: -106.06  Ratio of MFE: 0.84
Score: 476
Parsing shape block 11) Shape: [[][[][][][]]]  Score: -102.80  Ratio of MFE: 0.82
Score: 446
```

Another problem, which restricts the alignment score from being used in a straightforward way to rank the consensus shapes, is the following:

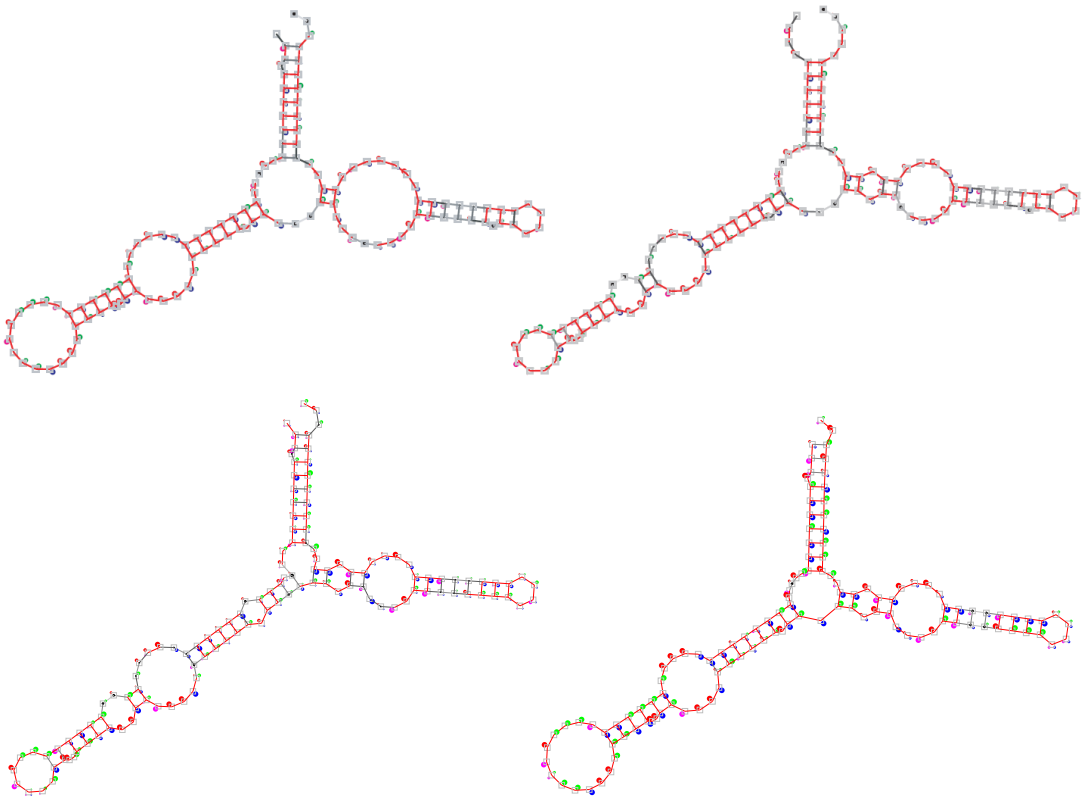
The length of the alignment influences the alignment score. In turn, the length of the shape, more precisely its shreps from the input sequences, influences the score as well, because brackets usually have a more influential score than bases.

With a relative score for the multiple alignment mode, the proposed rank function would be much more meaningful than the default absolute alignment score. The profile approach does not allow a straightforward computation of a relative score for a multiple structure alignment, as the profile alignment contains no explicit representation of its input forests, which would be needed to normalize the score.

Improvement of the alignment step Following Reeder and Giegerich in [91], we answer the first question and show that the alignment step can be improved.

We extend Figure 2 from [91], in which an alignment of known structures is compared to an alignment of shreps from the RNAcast application to the unfolded sequences. The resulting alignments are quite similar.

With our affine gap score variant of the alignment algorithm in *RNAforester*, it is possible to get a resulting alignment that is even more similar to the reference. The improved resulting alignment is shown in Figure 11.6.



```
RNAshapes -C -f 5SRNA.seqs | RNAforester -m -2d
RNAshapes -C -f 5SRNA.seqs | RNAforester -a -pdo=-10 -bdo=-25 -m -2d
```

Figure 11.6: This figure is extended from [91, Figure 2] and shows multiple structure alignments of five 5s rRNAs computed by *RNAforester*. On the upper left is the alignment of structures as found in the rRNA database [111], and on the right is an alignment of the shreps from *RNAcast*. As already explained in [91], the structures are quite similar and *RNAcast* predicts some compatible additional basepairs. We extended this figure by the two pictures below, the left one is an alignment of the same structures with *RNAforester 2.0* and default parameters (linear gap scoring), and the right one is an alignment with *RNAforester 2.0* and affine gap scoring, with *pair indel open=-10* and *base indel open=-25*. As we can see, with the affine variant of *RNAforester 2.0*, we get some more additional basepairs in the alignment, and the overall alignment gets more similar to the reference alignment, especially the predicted structure for the left hairpin.

12 Program evaluation

We designed a new algorithm for RNA structure comparison with affine gap costs, but does this algorithm improve the resulting alignments in practice? To assess the properties of the new algorithm, its performance can be evaluated with regard to the following criteria:

First of all, the algorithm is evaluated regarding basic features of technical correctness.

1. Is the new implementation of the original linear cost algorithm consistent with the old one, if the same scoring scheme is used?

2. Is the new algorithm with affine gap costs consistent with the algorithm with linear gap costs, if the same costs are used for gap opening and gap extension?

3. does the new algorithm produce purely structural or pure sequence alignments by setting the scores accordingly?

Then, the different algorithms are evaluated with regard to possible speedups and slowdowns.

1. The affine gap cost algorithm variants are expected to be slower than the original algorithm with the linear gap cost model, as they are computationally more expensive.

2. The anchored variant of the algorithm should be faster than the nonanchored variant. This is especially interesting for the affine gap cost model, which has higher computational costs to begin with.

The third very interesting evaluation part is the performance with regard to biological reality. The big question in this context is: How can we find out if our new alignment algorithm is better than the old one? Or to pose a follow up question: What does “better” or “good” mean? The idea applied here is that the resulting distance measure of RNAforester should reflect the relation between RNAs. Or in other words: are the RNAs from the same family?

Therefore, we evaluate the biological performance based on data from the Rfam database of noncoding RNA families. Each family consists of a seed alignment that is usually hand-curated, and a full alignment that was computed with a covariance model that was trained on the seed alignment. We focus on the seed alignments, because their quality is higher than that of the full alignments.

Our refined question now is: Does the affine cost model really advantage that a structure best aligns with its true family in Rfam?

Not many approaches have been made to benchmark alignment algorithms on structural RNAs. Sequence alignments have been benchmarked on structural RNAs in [32]. The WAR server [115] is a web server, where the user can choose one of several alignment programs to compute a consensus structure from a set of RNA sequences, also including *RNAcast* and *RNAforester*, but the authors do not focus on a comparison of the different methods. A benchmark of RNA secondary structure alignment algorithms has been presented in a short talk as described in [4] at Jobim 2008, in which several alignment methods were announced to be tested, including RNAforester 1.0. Following this benchmarking setup, we split up all families of Rfam in reference and test data, and try to find the true family by structure alignment.

```
global optimal score: -9
Intron_7          GUCUGUUACACGCCGAGAUCGGACUCCGAGUGAUAUCCUC-GA-CGGAUCUG
Intron_8          GUCUGUUACACGAGAGAUCGGUCUCCCGGAUCGAGCCCUCGACGGAUCUGAU

Intron_7          UCCGAUCUUGUGUUUCUCUGUUACUUGAU-UCGAUUACUCUGUUACUAUUCU
Intron_8          GAUCUGUUUCUCUGU-UACUUGAUUCGAUUACUGUUACUAUGUU-C----UC

Intron_7          UCUUUGUUACUACUACUACUACUA
Intron_8          -CG-U--U-CU--U--UG-U--UA

Intron_7         ((..((.(((...((((((((((...(((((((((.....-((-((((.....
Intron_8         .........(((((((.((((....(((((((((.....(((((((.....

Intron_7        ))))).))...........)))))))).)-))))..............)))))
Intron_8        ))))))))).......-..)))))))))...))))..........-.----))))

Intron_7         ....))).)).)).....
Intron_8         -))-)--.-..--.--..-.--..
```

Figure 12.1: Example of a tree alignment, which is scattered due to the original, linear gap score model and the default score parameters of *RNAforester*. This alignment has 24 matched basepairs and 23 singleton gaps, which appear as 15 composite gaps. Scoring type: global similarity; Scoring parameters: pair match: 10; pair indel: -5; base match: 1; base replacement: 0; base indel: -10;

12.1 Solving the scattered alignment problem with hand tuned scores

As intended, affine gap scoring helps to improve structure alignments. We show excerpts from three alignments of two introns of Arabidopsis thaliana. See Figures 12.1, 12.2, 12.3 and the explanation given there.

```
global optimal score: -125
Intron_7        GUCUGUUACACGCCGAGAUCGGACUCCGAGUGAUAUCCU-C-GACGGAUCUGU-
Intron_8        GUCUGUUACACGAGAGAUCGGUCUCCCGGAUCGAGCCCUCGACGGAUCUGAUUC

Intron_7        UCCGAUCUUGUGUUUCUCUGUUACUUGAU-UCGAUUACUCUGUUACUAUUCUCG
Intron_8        GAUC-UGUUUCUCUGUUACUUGAUUCGAUUACUGUUACUAUGUU-C----UCUC

Intron_7        UCUUUGUUACUACUACUACUACUA
Intron_8        CG--U--U-CU-------UUGUUA

Intron_7        ((..((.(((...((((((((((...(((((((((....-.-((((((.....-
Intron_8        .........(((((((((.((((....(((((((((.....(((((((.......

Intron_7        ))))).)).....)))))))))).)-))))..............)))))
Intron_8        ))))-))))........)))))))))...))))..........-.----))))

Intron_7        ....))).)).)).....
Intron_8        ))--)--.-..----------.....
```

Figure 12.2: Choosing the affine gap score model and a higher gap opening penalty for pair and base indels, the gaps are contracted to longer composite gaps. This alignment has also 24 matched basepairs, but only 11 composite gaps. Scoring type: affine global similarity; Scoring parameters: pair match: 10; pair indel open: -20; pair indel: -5; base match: 1; base replacement: 0; base indel: -10; base indel open: -20;

```
global optimal score: -161
Intron_7        GUCUGUU--ACACGCCGAGA-UCGGACUCCGAGUGAUAUCCU-C-GACGGAUCU
Intron_8        GUCUGUUACACG---AGAGAUCGGUCUCCCGGAUCGAGCCCUCGACGGAUCUGA

Intron_7        U--UCCGAUCUUGUGUUUCUCUGUUACUUGAU-UCGAUUACUCUGUUACUAUUC
Intron_8        UCCGAUC-UGUUUCUCUGUUACUUGAUUCGAUUACUGUUACUAUGUUC----UC

Intron_7        CGUUCUUUGUUACUACUACUACUACUA
Intron_8        CU-----CGUU-CU-------UUGUUA

Intron_7        ((..((.--(((...(((((-(((((...(((((((((....-.-((((((...
Intron_8        .........(((---(((((.((((....(((((((((.....(((((((...

Intron_7        .--)))).))..........)))))))).)-))))..............))
Intron_8        ...))))-))))........)))))))...))))...........----))

Intron_7        ))....))).)).))..........
Intron_8        ))-----))).-..----------.....
```

Figure 12.3: Choosing an even higher gap opening penalty for pair indels also contracts gaps between brackets/basepairs. This alignment now has 27 matched base pairs, and also 11 composite gaps, in positions different from those of the previous alignment. Scoring type: affine global similarity; Scoring parameters: pair match: 10; pair indel open: -30; pair indel: -5; base match: 1; base replacement: 0; base indel: -10; base indel open: -20;

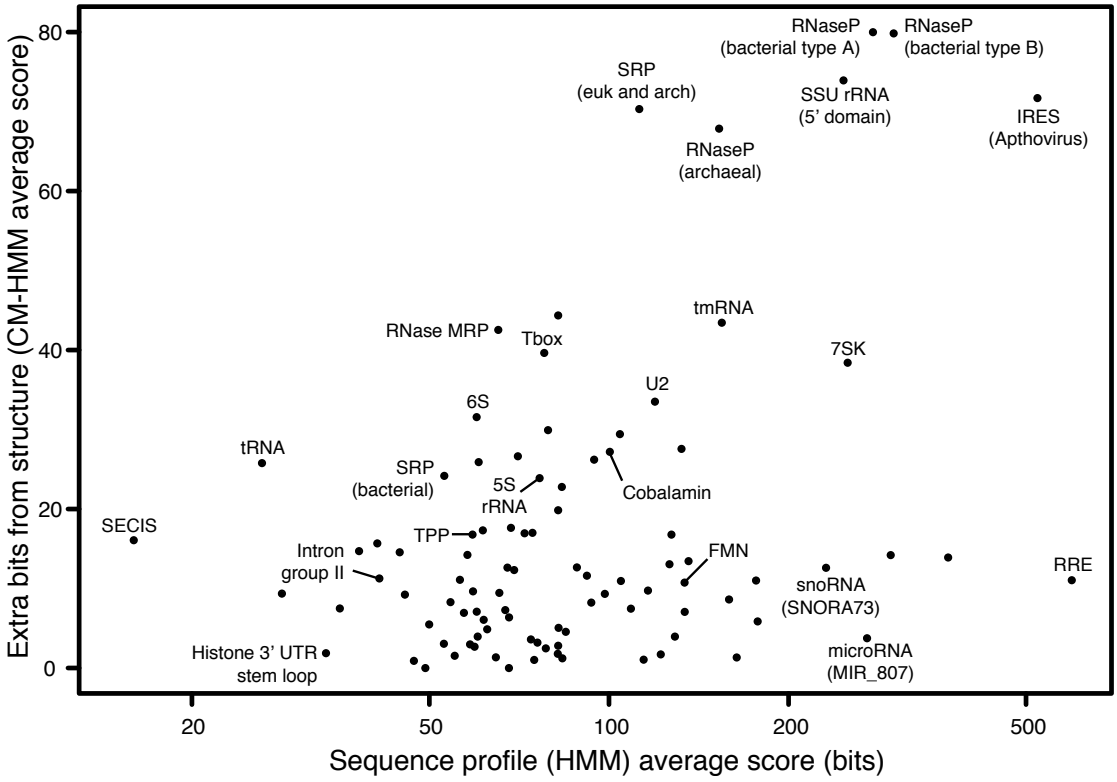


Figure 12.4: Figure from [86]

12.2 General measurements - on Rfam

To evaluate the structural alignment properties of RNAforester, the ideal data would be hand curated high quality structural alignments of RNAs that have a high influence of structural information. The Figure 12.4 from [86] sheds some light on the structure/sequence information ratio of the most important RNA families. For our pre-evaluation, we have chosen four well known RNA families with high structure/sequence information ratio, 5S rRNA (RF00001), Spot 42 (RF00021), Cobalamin (RF00174) and T-box (RF00230).

For the main evaluation, we use a subset of the Rfam database, with sequences of average length between 70 and 220 nucleotides. We exclude very short molecules that have not much structural information and can fold to hairpins only, and we exclude very long structures, which would make a large scale evaluation infeasible due to the complexity of the forest alignment algorithm (see Section 5.10). The majority of interesting families with a lot of structural information are within this length range.We use the Rfam 10 seed alignments, and with the restriction from 70 to 220 nucleotides average family length created a subset of 994 families.

To get to know the Rfam dataset a bit better, we first discuss some general measurements on our test data set. We inspect the testset of Rfam families with RNAforester 2.0. Compare also to [42], where the same properties were measured on artificial RNAs [42] for RNAforester 1.0.

Degree against number of nodes

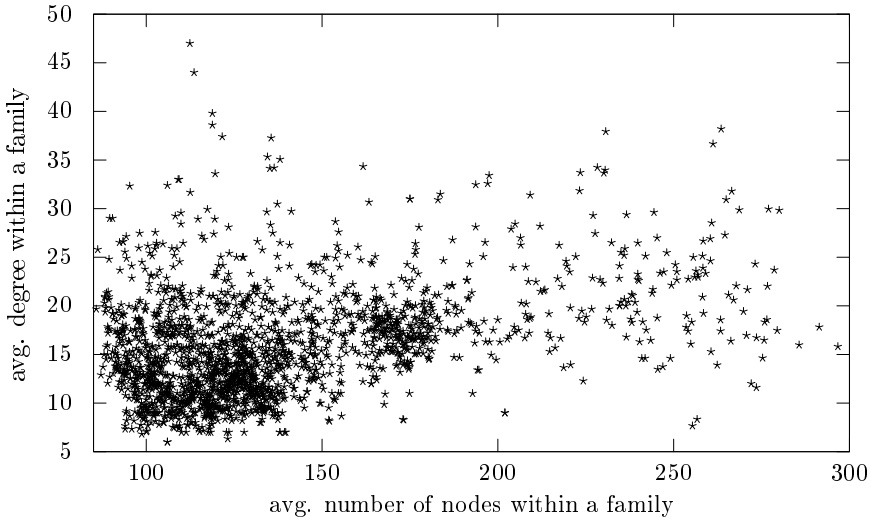


Figure 12.5: Degree of forest F against the size of the forest $|F|$ (i.e. the number of nodes). The degree is a constant function depending on the number of nodes.

If we plot the degree of each forest against the number of nodes $|F|$ in the forest for all input forests from the test set, we get a degree of about 30 even for larger forests. See Figure 12.5 for the plot result. That means, in the computational complexity terms, the degree can be treated as a constant.

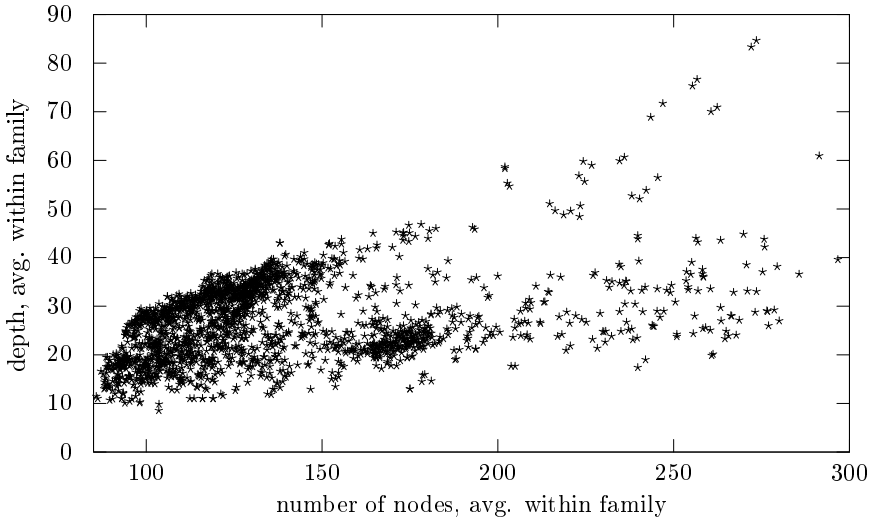


Figure 12.6: Depth of forest F against the size of the forest $|F|$. As expected, the depth grows in a linear fashion with the size of the forest.

Depth against number of nodes

The depth of the forest grows linearly with the number of nodes in the forest. This effect is depicted in Figure 12.6.

CSFs against number of nodes

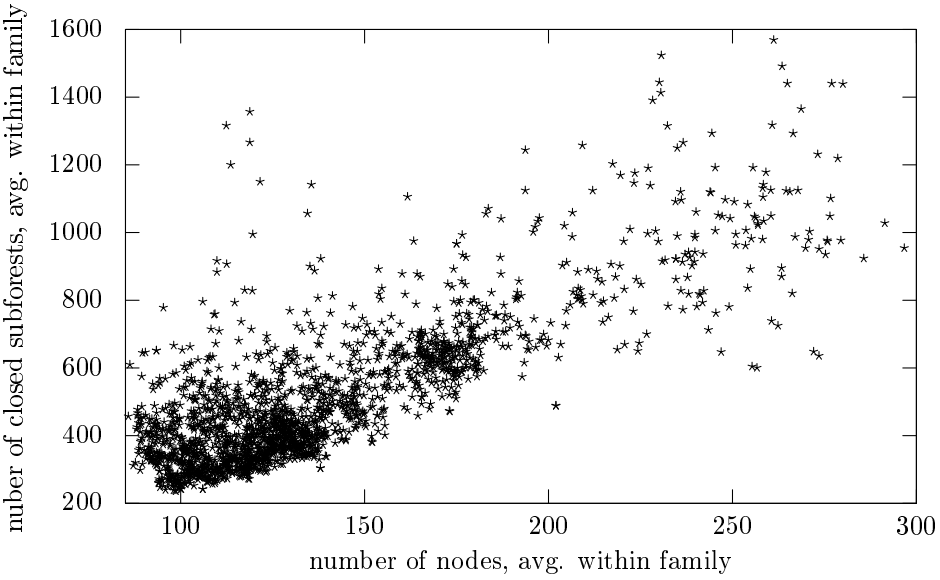


Figure 12.7: The number of closed subforests against the leaves of a forest.

The number of closed subforests determines the complexity of the global and local forest alignment, as the alignment of closed subforests is the subproblem in the recurrences. See Figure 12.7 for the number of CSFs against the nodes of the forests for all forests of the Rfam subset. We can see that the number of CSFs grows linearly with the number of nodes.

12.3 Consistency checks

RNAforester 1.0 versus RNAforester 2.0 The most obvious evaluation step is to check whether our implementation is coherent with the implementation of the original RNAforester program. To do this, we have to run the algorithm without the new features such as the affine gap cost model from Chapter 6, or the special computation that explicitly ensures intact P-nodes discussed in Section 9.2.

We also have to beware of some problems in the former program version, such as the silent death of the program on open (unfolded) structures as input, or fasta files of sequences instead of secondary structures as input.

If we filter out cases, in which RNAforester 1.0 produces no result, the programs are consistent.

The tests were performed on the subset of Rfam that was introduced in Section 12.2.

Linear gap scoring as special case of affine gap scoring The alignment with linear gap costs is a special case of the alignment with affine gap costs, in which the costs for opening and extending a gap are the same. This behavior should be measurable in practice, when we run the linear gap scoring variant and compare its result to that of the affine variant with *pair indel* = *pair indel open* and *base indel* = *base indel open*.

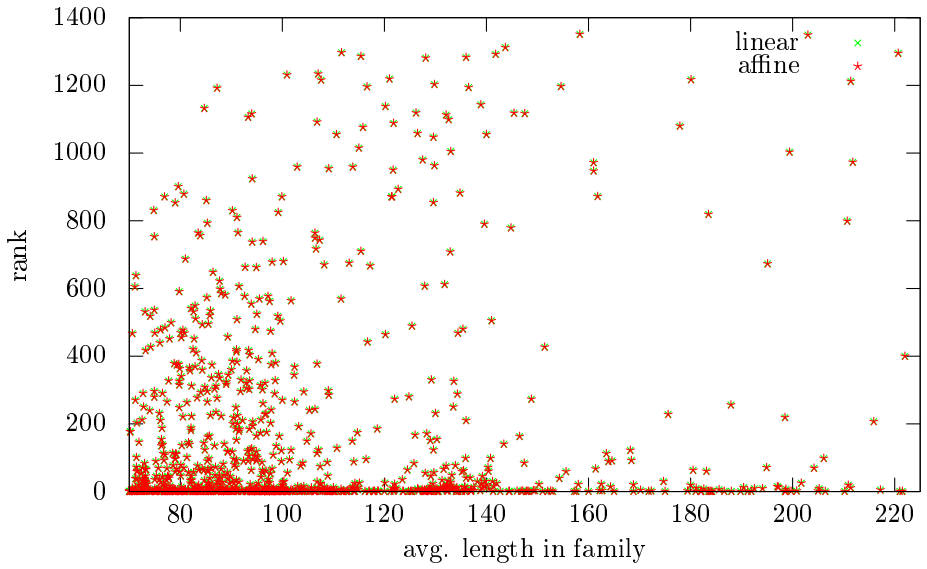


Figure 12.8: RNAforester 2 yields the same result for the linear gap cost model and the affine gap cost model when the provided open and extend scores are the same. The alignment with linear gap costs is a special case of the alignment with affine gap costs.

Figure 12.8 confirms this expectation. The rank evaluation procedure is explained in 12.6.

12.4 Speedup measurements

We now measure the speedup and slowdown for the two main new features, the affine gap scoring variant and the anchored variant of the algorithm. The measurements are again performed on the Rfam subset, with an all-against-all alignment of ten sequences from each family.

Comparison of affine and nonaffine variant As the affine gap cost model contributes a constant factor of seven to the original linear gap cost model algorithm, we expect to find this slowdown in the measurements as well. Figure 12.9 shows slowdowns by constant factors even for larger input forests. The factors depend on the algorithm variant. Global alignment is as expected less expensive than local alignment. Bottom up filling of the DP matrices tends to be less computationally expensive than the top down method, especially for larger problem sizes. Affine top down computation takes longest, but the time taken still is a constant factor compared to the linear variant.

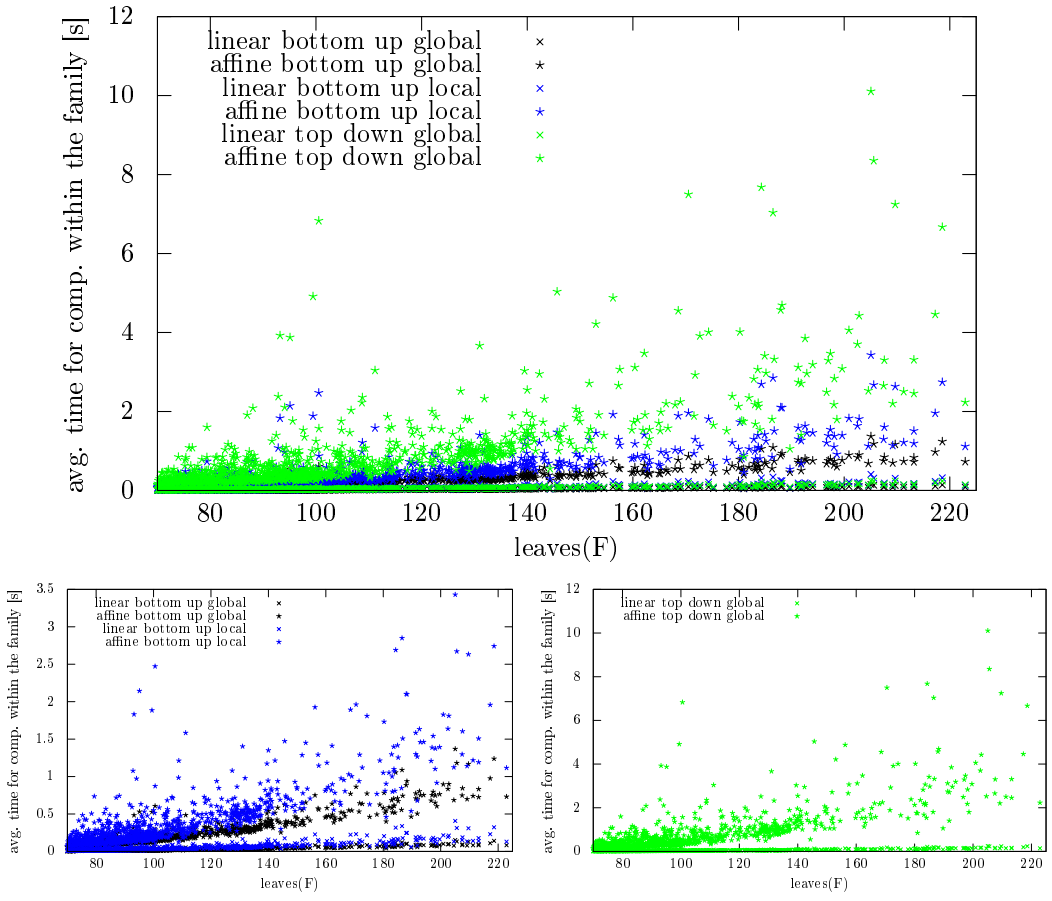


Figure 12.9: The slowdown when using the affine gap cost model in comparison to the linear one. The exact factors depend on the algorithm variant, if it is top down or bottom up, global or local. Anchoring is measured separately in the next subsection.

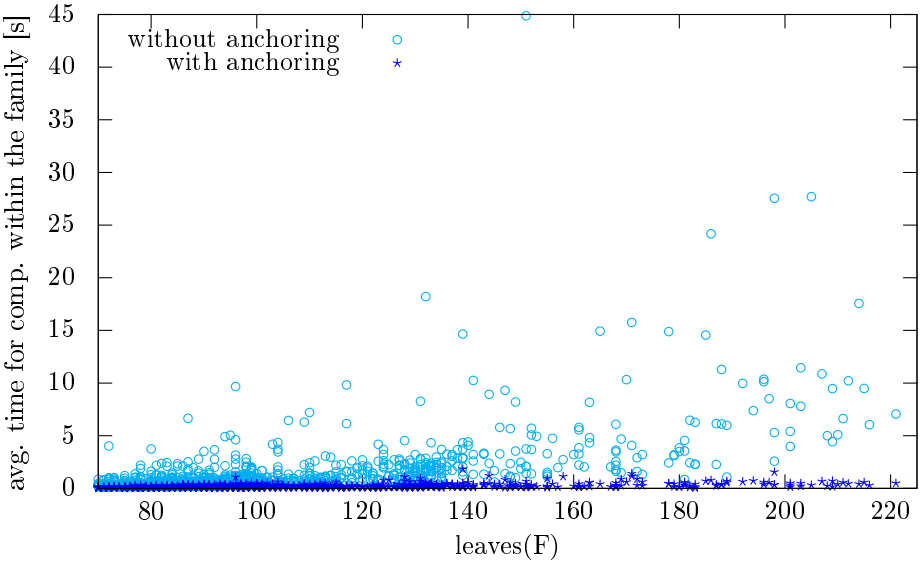


Figure 12.10: The speedup gained by using the anchoring, global, top down.

Comparison of anchored and nonanchored variant Figure 12.10 depicts the computation time of an average alignment within one family against the average length of the family's member sequences. The time is shown for the anchored computation and for the computation without anchoring. As expected, the anchored alignment is faster than the nonanchored variant. The improvement grows with the length of the input structure.

12.5 Score decline with the anchoring

The anchoring fixes certain P-nodes from moving around freely, and the search space is restricted. Thus, the score may be worse in the anchored case even if we use the same scoring parameters. This effect is visible in Figure 12.11.

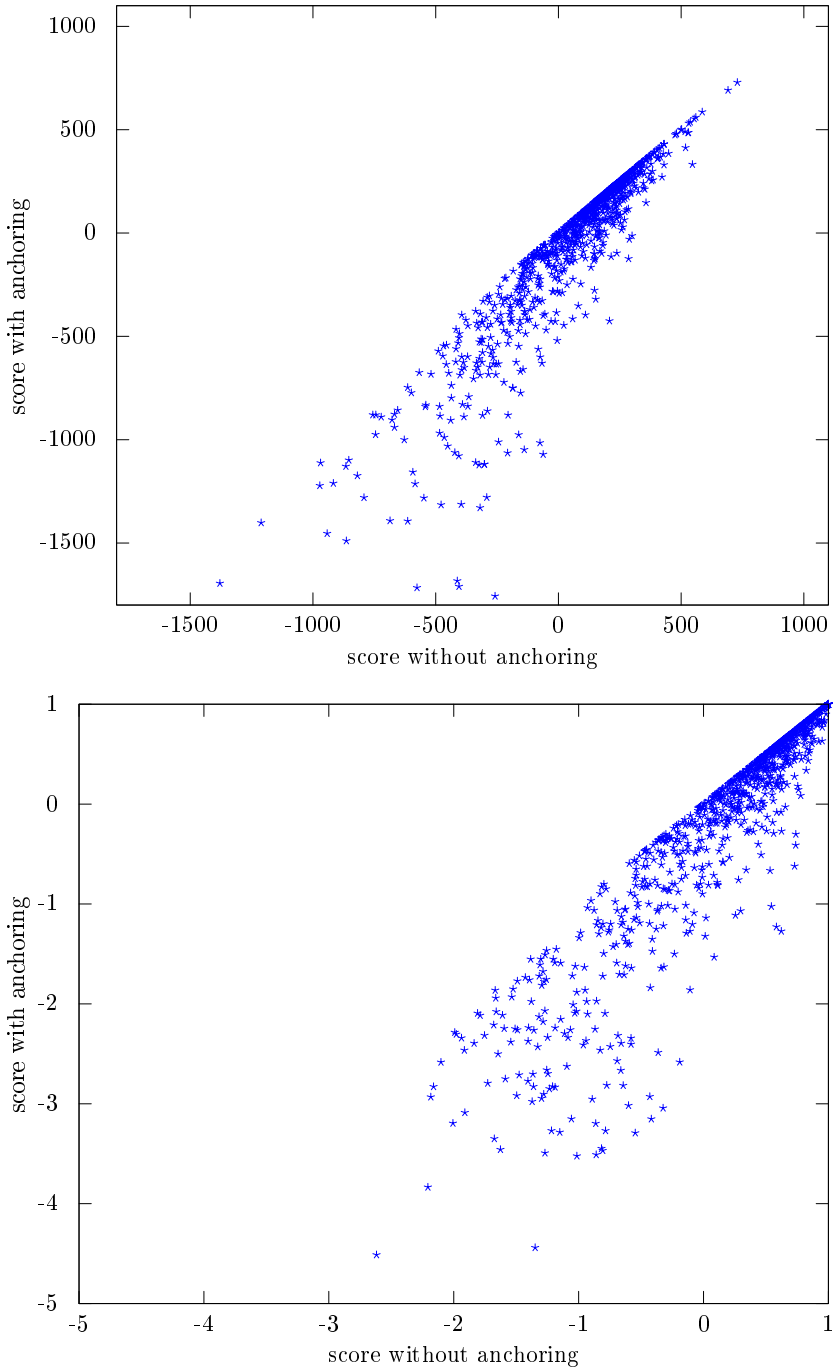


Figure 12.11: The score decline when using the anchoring - we use the affine variant with the default scoring scheme. On top we see the score decline of the absolute scores, and below is the score decline for the relative scores.

12.6 Benefits regarding the biological model

To evaluate the improvement of the affine gap score model with regard to biological reality, we have to find out whether the distance measure reflects the relation between RNA secondary structure correctly, for example for structures from the same family. We evaluate this property on the Rfam family data.

For this evaluation, we have taken the complete data set of Rfam 10.0 with 1412 families, to be able to compute the rank properly over all families.

The procedure is as follows:

- Take all Rfam families.
- Divide each family 50/50, put one part away as test data.
- For each family get the family consensus structure from Rfam.
- For each query seq from the test data, fold it into the consensus of its family with RNAfold -C.
- Search via tree comparison for the best match among all families by comparing each query against all families.
- For each query, rank the families according to the score.
- On which rank is the correct family for this structure? Is it higher than it was before, with RNAforester?

The overall design and information flow of the evaluation procedure is depicted as a diagram in Figure 12.12.

The procedure is straightforward, but problems might arise if the rank of the real family of the query is so low that it gets lost among other family. In this case it is not clear whether a rank improvement under these circumstances has a meaning or whether it is just noise. Thus, extra attention would be needed if a true family gets a bad rank in all cases. However, this problem was not prominent in our measurements after all, except for two data points for which the true family received a bad rank in all cases.

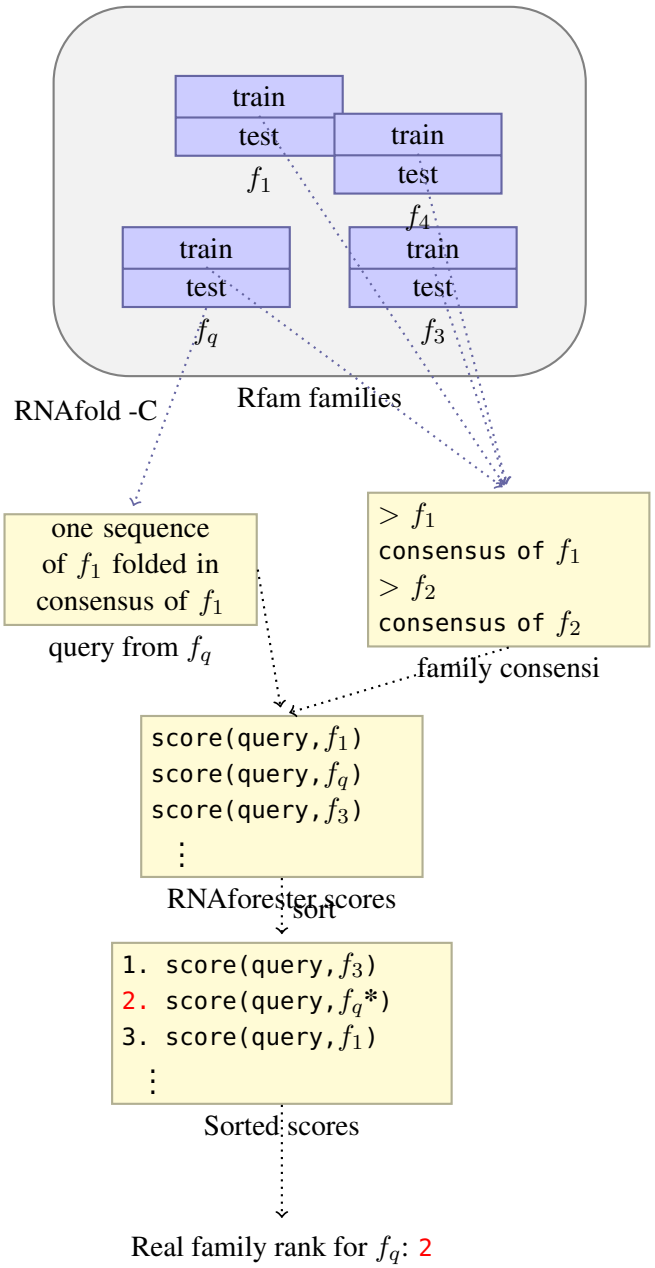


Figure 12.12: Rfam evaluation design for RNAforester 2.0.

Visualization of the rank analysis results

A good parameter when comparing the new affine algorithm to the original RNAforester 1.0 just slightly raises the cost for opening a gap in a sequence of bases.

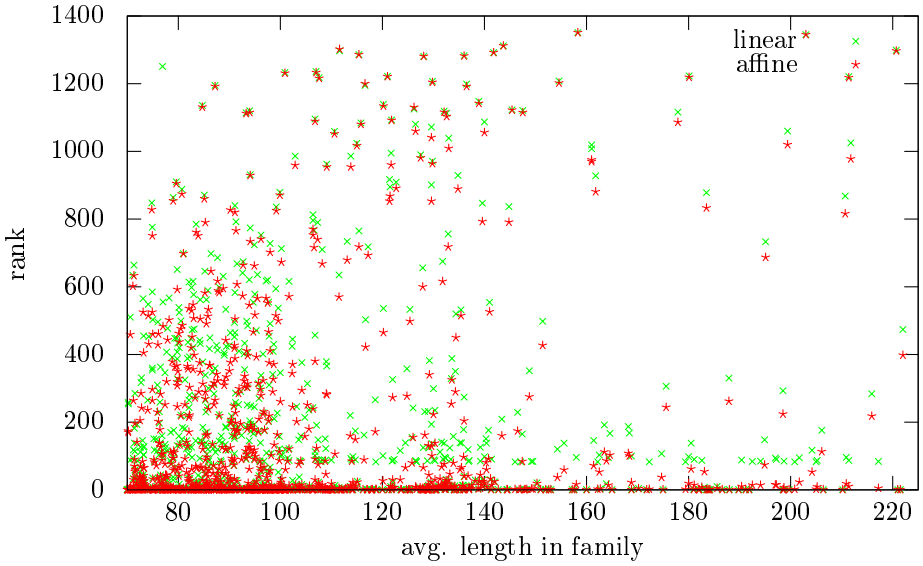


Figure 12.13: The rank of the true Rfam family for each query, measured with *RNAforester 1.0* and the linear gap scoring forest alignment algorithm, and measured with *RNAforester 2.0* and the affine gap scoring forest alignment algorithm. The query from an Rfam family is aligned better to its true family with the affine algorithm in almost all cases. In these cases, the rank of the true family score gets smaller - it gets a better rank. Scoring parameters RNAforester 1.0: pair match: 10; pair indel: -5; base match: 1; base replacement: 0; base indel: -10; Scoring parameters RNAforester 2.0 (Set 1.1): pair match: 10; *pair indel open: -6;* pair indel: -5; base match: 1; base replacement: 0; base indel: -10; *base indel open: -11;*

With this parameter set, and the affine cost model, almost all queries align better to their real family, which causes that the real family gets a lower rank than with the default set. To illustrate this result, we have drawn green arrows for an improvement with regard to the biological model, and red arrows for a change for the worse. As depicted, for most queries, the rank of the true family improved according to the score of the RNAforester 2.0 alignment, except for some outliers, for which the alignment received a lower score than before.

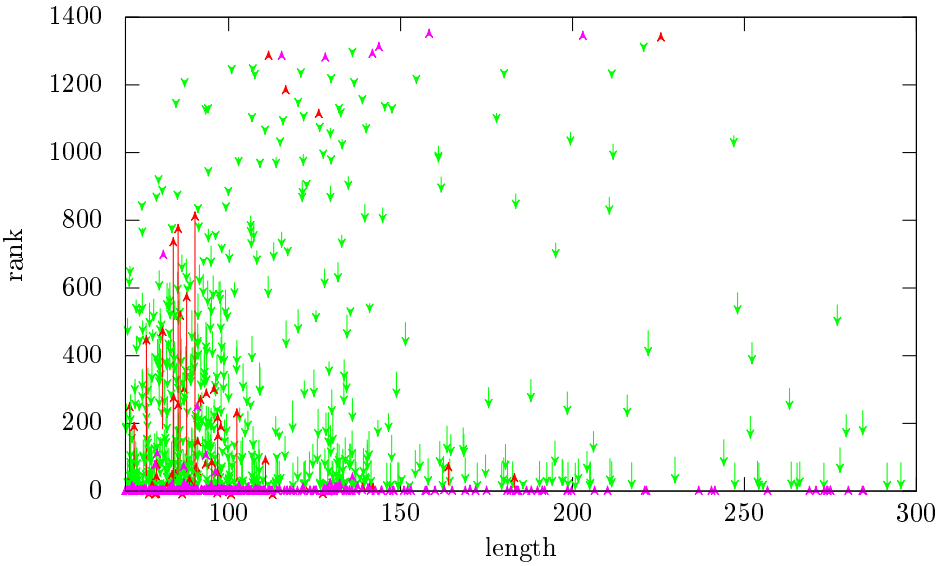


Figure 12.14: The green arrows depict a rank improvement, and the red ones a score that got worse under the affine scoring scheme with the given gap costs. The scoring parameters and the underlying data are the same as in figure 12.13.

Parameter stepping

We altered the gap opening parameters in several steps. As the evaluation over the complete Rfam database runs about a day even if parallelized, we were only able to try a few parameter combinations to get an impression. We alter the *base indel open* score and the *pair indel open* score, and leave the rest of the parameters fixed.

First, we increase the penalties for base indel open and basepair indel open in small steps of size one starting from *RNAforester 1.0's* default parameters.

The set of scores that performed best in our measurements is the one we have already encountered in the previous section. It is depicted in Figure 12.14. The rank of the true family improved in almost all cases with *RNAforester 2.0* and this parameter set.

Again increasing the gap opening penalty by one decreases the overall improvement of the rank of the true family. It leads to more and more alignments that get worse as can be seen in Figure 12.15 and Figure 12.16.

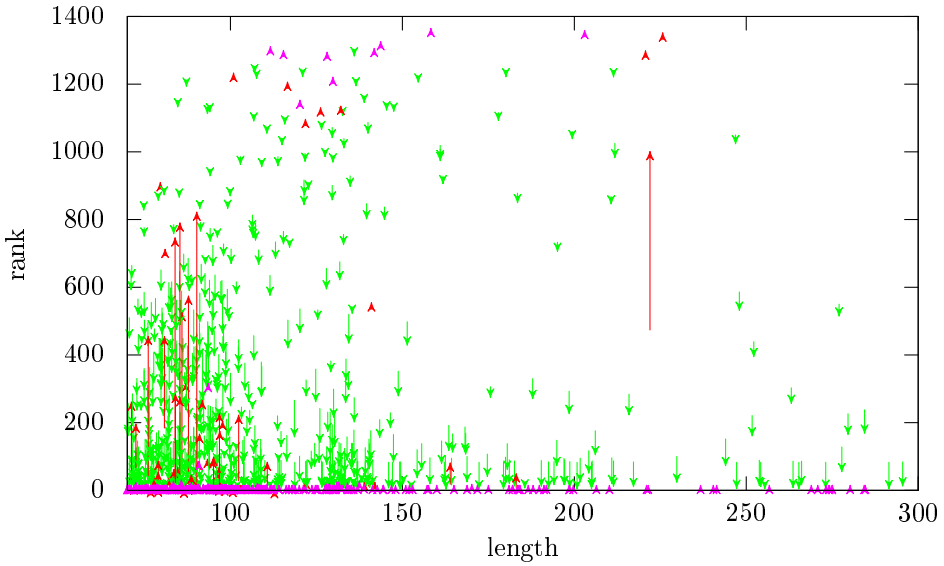


Figure 12.15: Increasing the gap opening penalty by two decreases the overall improvement gained by the affine variant. The query from Rfam is still aligned better to its true family in most cases. Scoring parameters RNAforester 1.0: all parameters are default parameters as in Figure 12.14; Scoring parameters RNAforester 2.0: *pair indel open: -7; base indel open: -12;* other parameters are default parameters as in Figure 12.14;

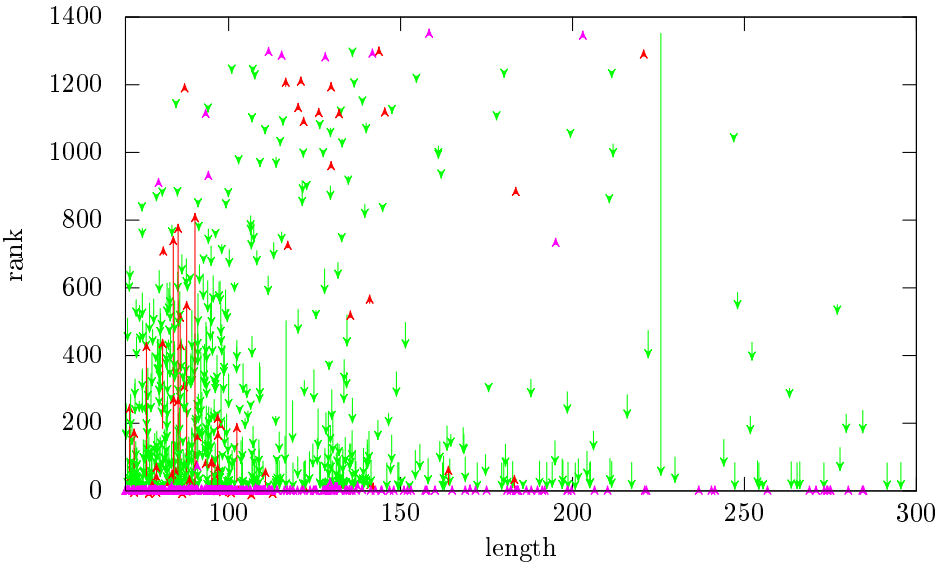


Figure 12.16: Increasing the gap opening penalty by three also decreases the overall improvement gained by the affine variant more. Scoring parameters RNAforester 1.0: all parameters are default parameters as in Figure 12.14; Scoring parameters RNAforester 2.0: *pair indel open: -8; base indel open: -13;* other parameters are default parameters as in Figure 12.14;

Increasing the gap opening penalty in bigger steps (by setting it to a multiplicity of the gap extension costs) decreases the improvement even more, and increases the number of alignments that get worse. This gradual effect can be seen in Figure 12.17 and 12.18.

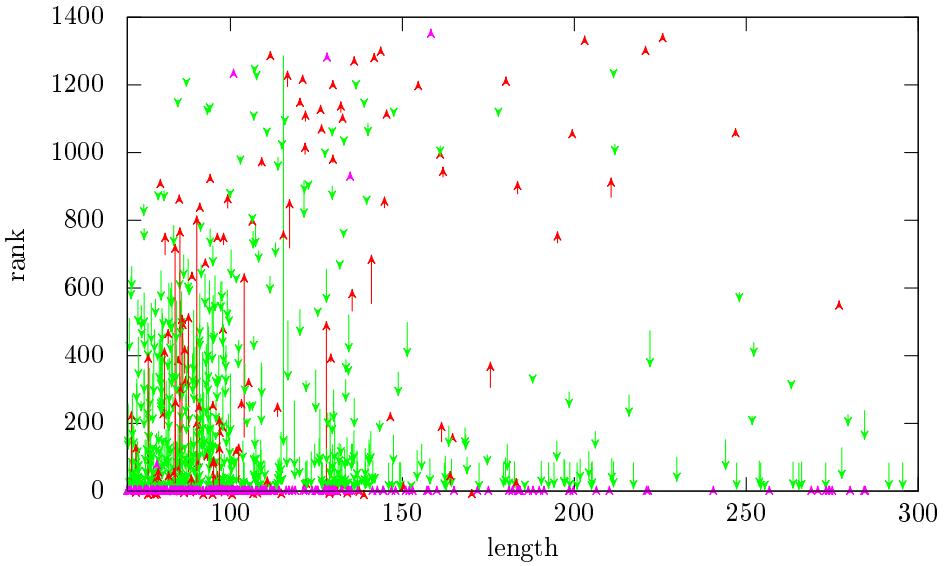


Figure 12.17: Increasing the gap opening penalty in bigger steps to set it to a multiplicity of the gap extension costs. Scoring parameters RNAforester 1.0: all parameters are default parameters as in Figure 12.14; Scoring parameters RNAforester 2.0: *pair indel open: -10; base indel open: -20;* other parameters are default parameters as in Figure 12.14;

Concluding from the evaluation, the affine and anchored alignment work as expected and may be used to replace the forest alignment with linear gap scoring and improve the resulting alignments.

There seems to be no general best choice for the gap opening parameters, as the best results are achieved for different parameter sets depending on the application. With the default parameters, a pair indel open score between -6 and -15, and a base indel open score between -11 and -20 is recommended based on the measurements.

Further analysis of the remaining negative outlier families reveals that they all have a secondary structure without any basepairings (an open structure) in the Rfam database.

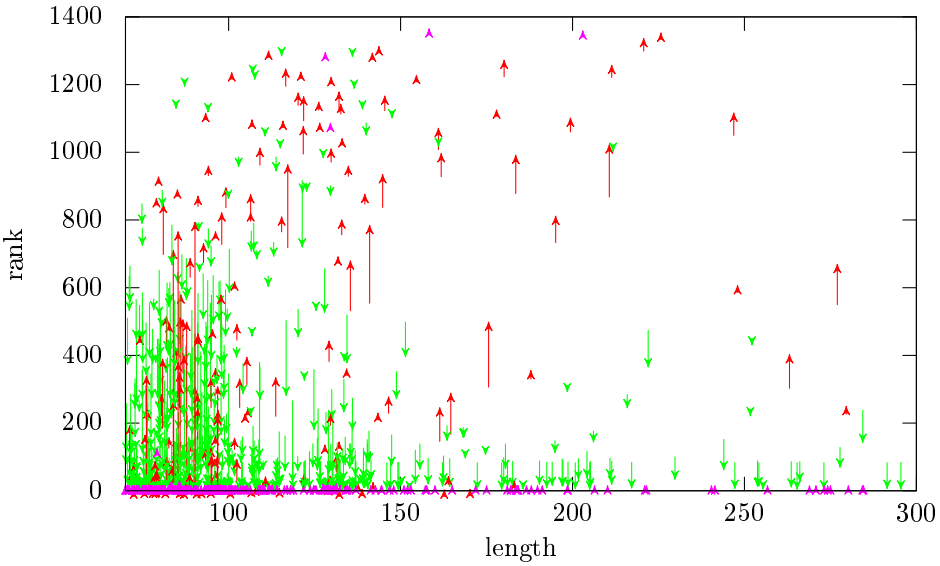


Figure 12.18: Increasing the gap opening penalty in bigger steps to set it to a multiplicity of the gap extension costs. Here, the opening costs are three times the extension costs. The set of opening costs is too aggressive compared to the other sets, because it most decreases the improvement that is gained by the affine gap model. Scoring parameters RNAforester 1.0: all parameters are default parameters as in Figure 12.14; Scoring parameters RNAforester 2.0: *pair indel open: -15; base indel open: -30;* other parameters are default parameters as in Figure 12.14;

13 Conclusion

In this thesis, we introduced two major additions to the forest alignment model and algorithm.

Firstly, the tree alignment algorithm was extended to work with an affine gap scoring model. Secondly, we also presented a way to increase the speed of the forest alignment algorithm by abstract shape analysis [109, 34] of the structures: the alignment could be anchored by an alignment of the abstract shapes.

Furthermore, the RNA forest alignment model has been equipped with additional edit operations for pair node insertions and deletions. Also, a top-down method to compute the alignment tables was developed.

Complexity The new algorithm with affine gap costs is in the same time complexity class as the original tree alignment algorithm [43, 55], as it just computes seven cases with the same recurrences instead of one. The time complexity for an alignment of two forests F and G with affine gap costs is thus $\mathcal{O}(|F| \times |G| \times (deg(F) + deg(G))^2)$, where $|F|$ is the number of nodes in F and $deg(F)$ is the degree of F.

The complexity expression simplifies to $\mathcal{O}(n^2 d^2)$, when $|F| = |G| = n$ and $deg(F) = deg(G) = d$. With the anchoring by shape abstraction, we are able to reduce the computation time to $\mathcal{O}(\frac{n^2}{k} d^2)$ with $k-1$ evenly placed anchors. In theory, d is in $\mathcal{O}(n)$, but in our application case of RNA secondary structures, $d \approx 30$.

Evaluation and Results Our affine gap cost algorithm works well on structural alignments where the original *RNAforester* introduced many small gaps.

Our measurements confirm that the implementation of the new algorithms performs as expected. *RNAforester 2.0* is overall consistent with *RNAforester 1.0*, except for cases where *RNAforester 1.0* produces no output. The linear gap score alignment algorithm implementation is consistent with the affine variant, and embedded in it, if we set the gap opening scores equal to the gap extension scores when using the affine gap score model.

The affine method incurs a constant factor compared to the linear gap scoring method. The size of the factor depends on the exact alignment algorithm (global, local, top down, bottom up).

The anchored alignment, which is only defined for the top down case, contributes a speedup to the computation. The speedup is expected to depend on the shape level, the length of the shape and the placement of the anchors. Further experiments have to be made to evaluate these dependencies.

Also, the anchored alignment procedure may lead to a score decline compared to the nonanchored alignment, due to restrictions of the search space. However, the majority of scores is close to the diagonal if we plot the nonanchored against the anchored scores, and therefore as good as in the nonanchored case. Our evaluation clearly demonstrates that the decline is least for high-scoring alignments. This justifies anchoring as a heuristic when the forests are compared because they are expected to be of high similarity because of a pre-selection.

To evaluate the performance of the affine algorithm in terms of biological reality, we retrieved one structure by random from each family from the Rfam database of noncoding RNAs and tried

to find their true family again via aligning to all family consensus structures from the complete Rfam database. Depending on the dataset, the rank of the true family improved in numerous cases with the affine gap scoring algorithm.

Replacing the common forest alignment algorithm with linear gap costs, we can use the affine gap score model to improve the alignment results.

This was shown for the *planACstar* pipeline, which includes a structural alignment step into the common align-and-fold procedure that is used to produce a consensus structure from a set of RNA sequences. For *planACstar*, the new alignment algorithm with affine gap costs is able to improve the structure conservation in the resulting alignments.

The *RNAforecast* pipeline is another way to get from a set of sequences to a consensus structure, in this case a shape-based consensus. As shown, the new forest alignment algorithms can be used to improve both the resulting alignment and the computation time in the *RNAforecast* pipeline.

Future prospects An interesting further extension of the forest alignment model and algorithms would be to integrate pseudoknot comparison in the analysis - this can for instance be done for simple, H-like pseudoknots [90] by opening one helix and comparing the remaining hairpin as usual, and then opening the other helix and comparing as usual, and taking the average (or: best) score of both cases.

Another promising addition would be the inclusion of a position dependent score, so the alignment program can be used for search of given structural motifs. A conserved basepair could e.g. be scored higher in a position where a paired base is expected by a known structural motif.

Also interesting would be an anchoring based on abstract shapes of shape level three, as would be more fine grained than the current anchoring which uses abstract shapes of level five, and could maybe lead to a bigger speedup. This requires a fast parser for level 3 shapes, which is probably not possible to develop with the help of a parser generator, as the level 3 shape parser is also not an LALR(1) parser and requires additional tricks.

Trees and forests are popular data structures throughout computer science. Wherever they grow, anchored alignment with affine gaps may help with their comparison.

Bibliography

[1] Bielefeld Bioinformatics Server. URL `http://bibiserv.techfak.uni-bielefeld.de`.

[2] J. Allali and M.-F. Sagot. A new distance for high level RNA secondary structure comparison. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1): 3–14, 2005.

[3] J. Allali and M.-F. Sagot. A multiple layer model to compare RNA secondary structures. *Software - Practice and Experience (SPE)*, 38(8):775–792, 2008.

[4] J. Allali, C. Chauve, A. Denise, C. Drevet, P. Ferraro, D. Gautheret, C. Herrbach, F. Leclerc, A. De Monte, M. Sagot, et al. Benchmarking RNA secondary structure comparison algorithms. *Actes des Journées Ouvertes de Biologie, Informatique et Mathématiques-JOBIM'08*, 2008.

[5] J. Allali, C. Chauve, P. Ferraro, and A.-L. Gaillard. Efficient chaining of seeds in ordered trees. In *Combinatorial Algorithms - 21st International Workshop*, pages 260–273, 2010.

[6] L. Alonso and R. Schott. On the tree inclusion problem. *Acta informatica*, 37(9):653–670, 2001.

[7] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM Journal of Computing*, 26(6):1656–1669, December 1997.

[8] A. Apostolico and Z. Galill. *Pattern matching algorithms*. Oxford Univ. Press, New York [u.a.], 1997. ISBN 0-19-511367-5.

[9] R. Backofen, G. Landau, M. Möhl, D. Tsur, and O. Weimann. Fast RNA structure alignment for crossing input structures. In *Combinatorial Pattern Matching*, pages 236–248. Springer, 2009.

[10] V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Combinatorial Pattern Matching*, pages 1–16. Springer, 1995.

[11] M. Barnsley. *Fractals everywhere*. Academic Press Professional, Inc., San Diego, CA, USA, 1988. ISBN 0-12-079062-9.

[12] T. Baumstark, A. Schröder, and D. Riesner. Viroid processing: switch from cleavage to ligation is driven by a change from a tetraloop to a loop e conformation. *The EMBO Journal*, 16(3):599–610, 1997.

[13] R. E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[14] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[15] G. Blin and H. Touzet. How to compare arc-annotated sequences: The alignment hierarchy. In *SPIRE*, pages 291–303, 2006.

[16] A. Bremges, S. Schirmer, and R. Giegerich. Fine-tuning structural RNA alignments in the twilight zone. *BMC bioinformatics*, 11(1):222, 2010.

[17] M. P. S. Brown. Small subunit ribosomal RNA modeling using stochastic context-free grammars. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 57–66. AAAI Press, 2000. ISBN 1-57735-115-0.

[18] T. R. Cech. The chemistry of self-splicing RNA and RNA enzymes. *Science*, 236(4808): 1532–1539, 1987.

[19] M. Charette and M. Gray. Pseudouridine in RNA: what, where, how, and why. *IUBMB life*, 49(5):341–351, 2000.

[20] W. Chen. More efficient algorithm for ordered tree inclusion. *Journal of Algorithms*, 26 (2):370 – 385, 1998.

[21] W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.

[22] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson. Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Research*, 31(13):3497–3500, 2003.

[23] F. Corpet and B. Michot. RNAlign program: alignment of RNA sequences using both primary and secondary structures. *Computer Applications in the Biosciences*, 10(4):389–399, Jul 1994.

[24] S. Dulucq and H. Touzet. Analysis of tree edit distance algorithms. In *CPM '03: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, pages 83–95, 2003.

[25] S. Dulucq and H. Touzet. Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms*, 3(2-4):448–471, 2005.

[26] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, July 1999. ISBN 0521629713.

[27] S. Eddy. Hidden markov models. *Current Opinion in Structural Biology*, 6(3):361–365, 1996.

[28] S. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079, 1994.

[29] P. Evans. *Algorithms and complexity for annotated sequence analysis*. PhD thesis, Citeseer, 1999.

[30] W. Fontana, D. Konings, P. Stadler, and P. Schuster. Statistics of RNA secondary structures. *Biopolymers*, 33(9):1389–1404, 1993.

[31] P. Gardner and R. Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5(1):140, 2004.

[32] P. P. Gardner, A. Wilm, and S. Washietl. A benchmark of multiple sequence alignment programs upon structuralRNAs. *Nucleic Acids Research*, 33(8):2433–2439, 2005.

[33] R. Giegerich, C. Meyer, and P. Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215–263, 2004.

[34] R. Giegerich, B. Voss, and M. Rehmsmeier. Abstract shapes of RNA. *Nucleic Acids Research*, 32(16):4843–4851, 2004.

[35] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, Dec 1982.

[36] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. Eddy. Rfam: an RNA family database. *Nucleic Acids Research*, 31:439–441, 2003.

[37] A. R. Gruber, S. H. Bernhart, I. L. Hofacker, and S. Washietl. Strategies for measuring evolutionary conservation of RNA secondary structures. *BMC Bioinformatics*, 9(1):122–122 [Epub ahead of print], Feb 2008.

[38] A. Gupta and N. Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210, 1998.

[39] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, January 1997. ISBN 0521585198.

[40] J. Havgaard, R. Lyngsø, and J. Gorodkin. The foldalign web server for pairwise structural RNA alignment and mutual motif search. *Nucleic Acids Research*, 33(2):W650, 2005.

[41] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977.

[42] M. Hoechsmann. *The tree alignment model: algorithms, implementations and applications for the analysis of RNA secondary structures*. PhD thesis, Bielefeld University, 2005.

[43] M. Hoechsmann, T. Toeller, R. Giegerich, and S. Kurtz. Local similarity in RNA secondary structures. *Proceedings of the IEEE Computational Systems Bioinformatics Conference (CSB 2003)*, 2:159–168, 2003.

[44] M. Hoechsmann, B. Voss, and R. Giegerich. Pure multiple RNA secondary structure alignments: A progressive profile approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1:53–62, 2004.

[45] I. Hofacker, W. Fontana, P. Stadler, L. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie*, 125: 167–188, 1994.

[46] I. L. Hofacker. RNA consensus structure prediction with RNAalifold. *Methods in Molecular Biology*, 395:527–544, 2007.

[47] I. L. Hofacker, M. Fekete, and P. F. Stadler. Secondary structure prediction for aligned RNA sequences. *Journal of Molecular Biology*, 319(5):1059–1066, June 2002.

[48] P. Hogeweg and B. Hesper. Energy directed folding of RNA sequences. *Nucleic Acids Research*, 12:67–74, 1984.

[49] P. J. Huber. *Robust Statistics*. Wiley-Interscience, 1981.

[50] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Commun. ACM*, 20(5):350–353, 1977.

[51] J. Jaeger, D. Turner, and M. Zuker. Improved predictions of secondary structures for rna. *Proceedings of the National Academy of Sciences*, 86(20):7706, 1989.

[52] S. Janssen, J. Reeder, and R. Giegerich. Shape based indexing for faster search of RNA family databases. *BMC Bioinformatics*, 9(1):131, 2008.

[53] J. Jansson and A. Lingas. A fast algorithm for optimal alignment between similar ordered trees. In *CPM '01: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*, pages 232–240, London, UK, 2001. Springer-Verlag. ISBN 3-540-42271-4.

[54] Y. Ji, X. Xu, and G. Stormo. A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences. *Bioinformatics*, 20(10):1591, 2004.

[55] T. Jiang, L. Wang, and K. Zhang. Alignment of trees – an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[56] T. Jiang, G. Lin, B. Ma, and K. Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–388, 2002.

[57] K. Katoh and H. Toh. Recent developments in the mafft multiple sequence alignment program. *Brief Bioinform*, 9(4):286–298, Jul 2008.

[58] D. Keselman and A. Amir. Maximum agreement subtree in a set of evolutionary trees-metrics and efficient algorithms. *Symposium on Foundations of Computer Science*, pages 758–769, 1994.

[59] P. Kilpelainen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, Faculty of Science, Department of Computer Science, 1992.

[60] P. Kilpelainen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal of Computing*, 24(2):340–356, 1995.

[61] S. Kim, G. Quigley, F. Suddath, and A. Rich. High-resolution x-ray diffraction patterns of crystalline transferRNA that show helical regions. *Proceedings of the National Academy of Sciences*, 68(4):841, 1971.

[62] J. Kitagawa, Y. Futamura, and K. Yamamoto. Analysis of the conformational energy landscape of human snRNA with a metric based on tree representation of RNA structures. *Nucleic Acids Research*, 31(7):2006–2013, 2003.

[63] P. N. Klein. Computing the edit-distance between unrooted ordered trees. In *In Proceedings of the 6th annual European Symposium on Algorithms (ESA)*, pages 91–102. Springer-Verlag, 1998.

[64] B. Knudsen and J. Hein. Pfold: Rna secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res*, 31(13):3423–3428, Jul 2003.

[65] D. Konings and P. Hogeweg. Pattern analysis of RNA secondary structure:: Similarity and consensus of minimal-energy folding. *Journal of molecular biology*, 207(3):597–614, 1989.

[66] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden markov models in computational biology. applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

[67] S. Y. Le, R. Nussinov, and J. V. Maizel. Tree graphs of RNA secondary structures and their comparisons. *Computers and Biomedical Research*, 22(5):461–473, Oct 1989.

[68] A. Lee and D. Crothers. The solution structure of an RNA loop-loop complex: the cole1 inverted loop sequence. *Structure*, 6(8):993–1007, 1998.

[69] H.-P. Lenhof, K. Reinert, and M. Vingron. A polyhedral approach to RNA sequence structure alignment. In *RECOMB '98: Proceedings of the second annual international conference on Computational molecular biology*, pages 153–162, New York, NY, USA, 1998. ACM. ISBN 0-89791-976-9.

[70] N. Leontis and E. Westhof. Geometric nomenclature and classification of RNA base pairs. *RNA*, 7(04):499–512, 2001.

[71] Y. Li and Z. Chenguang. A metric normalization of tree edit distance. *Frontiers of Computer Science in China*, pages 1–7, 2011.

[72] S. Liu and E. Tanaka. A largest common similar substructure problem for trees embedded in a plane. *IEICE technical report. Theoretical foundations of Computing*, 95(498):11–20, 1996.

[73] S. Liu and E. Tanaka. Largest common similar substructures of rooted and unordered trees. *Memoirs of the Graduate School of Science and Technology, Kobe University. A*, 14:107–119, 1996.

[74] S. Liu and E. Tanaka. The largest common similar substructure problem (special section on discrete mathematics and its applications). *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 80(4):643–650, 1997.

[75] A. Lozano, R. Pinter, O. Rokhlenko, G. Valiente, and M. Ziv-Ukelson. Seeded tree alignment. *IEEE Transactions on Computational Biology and Bioinformatics*, pages 503–513, 2008.

[76] C. L. Lu, Z.-Y. Su, and C. Y. Tang. A new measure of edit distance between labeled trees. In *COCOON '01: Proceedings of the 7th Annual International Conference on Computing and Combinatorics*, pages 338–348, London, UK, 2001. Springer-Verlag. ISBN 3-540-42494-6.

[77] S. Lu. A tree-to-tree distance and its application to cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):219–224, 1979.

[78] S. Y. Lu. A tree-matching algorithm based on node splitting and merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(2):249 – 256, March 1984.

[79] D. Mathews and D. Turner. Dynalign: An algorithm for finding the secondary structure common to two RNA sequences. *Journal of Molecular Biology*, 317(2):191–203, 2002.

[80] D. Mathews, M. Disney, J. Childs, S. Schroeder, M. Zuker, and D. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of rna secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287, 2004.

[81] D. H. Mathews and D. H. Turner. Prediction of RNA secondary structure by free energy minimization. *Current Opinion in Structural Biology*, 16(3):270–278, June 2006.

[82] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992.

[83] J. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

[84] M. Möhl, S. Will, and R. Backofen. Fixed parameter tractable alignment of RNA structures including arbitrary pseudoknots. In *Combinatorial Pattern Matching*, pages 69–81. Springer, 2008.

[85] V. Moulton, M. Zuker, M. A. Steel, R. Pointon, and D. Penny. Metrics on RNA secondary structures. *Journal of Computational Biology*, 7(1-2):277–292, 2000.

[86] E. Nawrocki. *Structural RNA Homology Search and Alignment Using Covariance Models*. PhD thesis, PhD thesis, Washington University School of Medicine, 2009.

[87] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48 (3):443 – 453, 1970.

[88] A. Ouangraoua and P. Ferraro. A new constrained edit distance between quotiented ordered trees. *Journal of Discrete Algorithms*, 7(1):78–89, 2009.

[89] O. Poirot, E. O'Toole, and C. Notredame. Tcoffee@igs: A web server for computing, evaluating and combining multiple sequence alignments. *Nucleic Acids Res*, 31(13): 3503–3506, Jul 2003.

[90] J. Reeder. *Algorithms for RNA secondary structure analysis: prediction of pseudoknots and the consensus shapes approach*. PhD thesis, Bielefeld University, 2008.

[91] J. Reeder and R. Giegerich. Consensus shapes: an alternative to the sankoff algorithm for RNA consensus structure prediction. *Bioinformatics*, 21(17):3516–3523, September 2005.

[92] T. Richter. A new algorithm for the ordered tree inclusion problem. In *CPM '97: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, pages 150–166, London, UK, 1997. Springer-Verlag. ISBN 3-540-63220-4.

[93] T. Richter. A new measure of the distance between ordered trees and its applications. Technical report, University of Bonn, 1997.

[94] J. Rico-Juan and L. Micó. Comparison of aesa and laesa search algorithms using string and tree-edit-distances. *Pattern Recognition Letters*, 24(9-10):1417–1426, 2003.

[95] W. Ritchie, M. Legendre, and D. Gautheret. RNA stem loops: to be or not to be cleaved by RNAse III. *RNA*, 13(4), February 2007.

[96] F. Rosselló and G. Valiente. An algebraic view of the relation between largest common subtrees and smallest common supertrees. *Theoretical Computer Science*, 362(1):33–53, 2006.

[97] J. Ruan, G. D. Stormo, and W. Zhang. Ilm: a web server for predicting rna secondary structures with pseudoknots. *Nucleic Acids Res*, 32(Web Server issue):146–149, Jul 2004.

[98] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22(23):5112–5120, November 1994.

[99] D. Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal of Applied Mathematics*, 45(5):810–825, 1985.

[100] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Center for the Study of Language and Inf, December 1999. ISBN 1575862174.

[101] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6): 184–186, 1977.

[102] B. A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer Applications in Bioscience*, 4(3):387–393, Aug 1988.

[103] B. A. Shapiro and K. Z. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in Bioscience*, 6(4):309–318, Oct 1990.

[104] D. Shasha and K. Zhang. Fast algorithms for the unit cost editing distance between trees. *Journal of Algorithms*, 11(4):581 – 621, December 1990.

[105] S. Siebert and R. Backofen. MARNA: A server for multiple alignment of RNAs. In *Proceedings of the German Conference on Bioinformatics*, pages 135–140, 2003.

[106] S. Siebert and R. Backofen. Marna: multiple alignment and consensus structure prediction of rnas based on sequence structure comparisons. *Bioinformatics*, 21(16):3352, 2005.

[107] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, Mar 1981.

[108] D. Staple and S. Butcher. Pseudoknots: RNA structures with diverse functions. *PLoS Biology*, 3(6):e213, 2005.

[109] P. Steffen, B. Voß, M. Rehmsmeier, J. Reeder, and R. Giegerich. RNAshapes: an integrated RNA analysis package based on abstract shapes. *Bioinformatics*, 22(4):500, 2005.

[110] J. Stoye. Multiple sequence alignment with the divide-and-conquer method. *Gene*, 211: 2, 1998.

[111] M. Szymanski, M. Barciszewska, V. Erdmann, and J. Barciszewski. 5s ribosomal rna database. *Nucleic Acids Research*, 30(1):176, 2002.

[112] K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[113] E. Tanaka. A metric between unrooted and unordered trees and its bottom-up computing method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(12):1233–1238, 1994.

[114] E. Tanaka and K. Tanaka. The tree-to-tree editing problem. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):221–240, 1988.

[115] E. Torarinsson and S. Lindgreen. War: Webserver for aligning structural rnas. *Nucleic acids research*, 36(suppl 2):W79, 2008.

[116] H. Touzet. Tree edit distance with gaps. *Information Processing Letters*, 85(3):123–129, 2003.

[117] H. Touzet. A linear tree edit distance algorithm for similar ordered trees. In *CPM '05: Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching*, pages 334–345, 2005.

[118] G. Valiente. An efficient bottom-up distance between trees. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval*, pages 212–219, 2001.

[119] G. Valiente. *Algorithms on trees and graphs*. Springer Verlag, 2002.

[120] M. Vingron and M. Waterman. Sequence alignment and penalty choice:: Review of concepts, case studies and implications. *Journal of molecular biology*, 235(1):1–12, 1994.

[121] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[122] J. Wang and K. Zhang. Finding similar consensus between trees: an algorithm and a distance hierarchy. *Pattern Recognition*, 34(1):127–137, 2001.

[123] J. Wang, K. Zhang, and C. Chang. Identifying approximately common substructures in trees based on a restricted edit distance. *Information Sciences*, 121:367–386, 1999.

[124] J.-L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):559–571, 1994.

[125] J. T. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):889–895, 1998.

[126] S. Washietl and I. Hofacker. Consensus folding of aligned sequences as a new measure for the detection of functional RNAs by comparative genomics. *Journal of Molecular Biology*, 342, 2004.

[127] S. Washietl, I. Hofacker, M. Lukasser, A. Hüttenhofer, and P. Stadler. Mapping of conserved RNA secondary structures predicts thousands of functional noncoding RNAs in the human genome. *Nature biotechnology*, 23(11):1383–1390, 2005.

[128] E. Westhof and P. Auffinger. *RNA Tertiary Structure*, pages 5222–5232. John Wiley & Sons, Ltd, 2006. ISBN 9780470027318.

[129] J. Whisstock and A. Lesk. Prediction of protein function from protein sequence and structure. *Quarterly reviews of biophysics*, 36(03):307–340, 2003.

[130] Wikipedia. Transfer RNA — Wikipedia, The Free Encyclopedia. Online, 2011. URL `http://en.wikipedia.org/wiki/Transfer_RNA`. [Online; accessed 18-May-2011].

[131] R. Wilhelm. A modified tree-to-tree correction problem. *Information processing letters*, 12(3):127–132, 1981.

[132] S. Will, K. Reiche, I. Hofacker, P. Stadler, and R. Backofen. Inferring noncoding RNA families and classes by means of genome-scale structure-based clustering. *PLoS Computational Biology*, 3(4):e65, 2007.

[133] A. Wilm, K. Linnenbrink, and G. Steger. Construct: improved construction of RNA consensus structures. *BMC Bioinformatics*, 9(219), 2008.

[134] W. Yang. Identifying syntactic differences between two programs. *Software - Practice and Experience*, 21:739–755, 1991.

[135] B. Yoon and P. Vaidynathan. HMM with auxiliary memory: a new tool for modeling RNA structures. In *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems & Computers*, volume 2, pages 1651–1655. IEEE, 2004.

[136] K. Zhang. Algorithms for the constrained editing distance between ordered labeled trees and related problems. *Pattern Recognition*, 28(3):463 – 474, 1995.

[137] K. Zhang. Efficient parallel algorithms for tree editing problems. In *CPM '96: Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, pages 361–372, London, UK, 1996. Springer. ISBN 3-540-61258-0.

[138] K. Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15 (3):205–222, 1996.

[139] K. Zhang and T. Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.

[140] K. Zhang and D. Shasha. On the editing distance between trees and related problems. *Ultra-computer Note 122, NYU C.S TR 310,*, August 1987.

[141] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262, December 1989.

[142] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

[143] K. Zhang, J. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs. *International Journal of Foundations of Computer Science*, 7(1):43–58, 1996.

[144] K. Zhang, L. Wang, and B. Ma. Computing similarity between RNA structures. In *CPM '99: Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, pages 281–293, London, UK, 1999. Springer-Verlag. ISBN 3-540-66278-2.

[145] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244:48–52, 1989.

[146] M. Zuker. *The use of dynamic programming algorithms in RNA secondary structure prediction*, pages 159–184. CRC Press, Boca Raton, RL, 1989.

[147] M. Zuker, J. Jaeger, and D. Turner. A comparison of optimal and suboptimal RNA secondary structures predicted by free energy minimization with structures determined by phylogenetic comparison. *Nucleic Acids Research*, 19(10):2707–2714, 1991.