Local Similarity in RNA Secondary Structures

Matthias Höchsmann[1], Thomas Töller[2], Robert Giegerich[3], and Stefan Kurtz[3]

[1] International Graduate School in Bioinformatics and Genome Research, Universität Bielefeld, Postfach 100 131, D-33501 Bielefeld, Germany.
[2] Graduiertenkolleg Bioinformatik, Universität Bielefeld, Postfach 100 131, D-33501 Bielefeld, Germany.
[3] Technische Fakultät, Universität Bielefeld, Postfach 100 131, D-33501 Bielefeld, Germany.
{mhoechsm, ttoeller, robert, kurtz}@techfak.uni-bielefeld.de

Abstract. We present a systematic treatment of alignment distance and local similarity algorithms on trees and forests. We build upon the tree alignment algorithm for ordered trees given by Jiang et. al (1995) and extend it to calculate local forest alignments, which is essential for finding local similar regions in RNA secondary structures. The time complexity of our algorithm is $O(|F_1| \cdot |F_2| \cdot deg(F_1) \cdot deg(F_2) \cdot (deg(F_1) + deg(F_2))$ where $|F_i|$ is the number of nodes in forest F_i and $deg(F_i)$ is the degree of F_i. We provide carefully engineered dynamic programming implementations using dense, two-dimensional tables which considerably reduces the space requirement. We suggest a new representation of RNA secondary structures as forests that allow reasonable scoring of edit operations on RNA secondary structures. The comparison of RNA secondary structures is facilitated by a new visualization technique for RNA secondary structure alignments. Finally, we show how potential regulatory motifs can be discovered solely by their structural preservation, and independent of their sequence conservation and position.

Keywords: tree alignment, dynamic programming, RNA secondary structures, regolatory motifs.

Note: The visualization uses color and can best be viewed on the screen.

1 Motivation

1.1 Introduction

RNA is a chain molecule, mathematically a string over a four letter alphabet. It is built from nucleotides containing the bases A(denine), C(ytosine), G(uanine), and U(racil). By folding back onto itself, an RNA molecule forms structure, stabilized by the forces of hydrogen bonds between certain pairs of bases (A–U, C–G, G–U), and dense stacking of neighbouring base pairs.

The investigation of RNA secondary structures is a challenging task in molecular biology. RNA molecules[1] have a large variety of functions in the cell which often depend on special structural properties. Evolutionary conserved tRNAs [22] and rRNAs [7] carry out protein synthesis. Small nuclear RNAs are important for the splicing of pre-mRNAs [24] and small nucleolar RNAs act as guide RNAs for the modification of other RNA molecules [13]. The untranslated terminal regions (UTRs) of mRNAs can contain regulatory motifs which play a role for posttranscriptional gene regulation. Such motifs can affect the mRNA localization [9], mRNA degradation [6] and translational regulation [5]. Therefore, discovering such similar motifs inside otherwise unrelated structures is important for the investigation of posttranscriptional gene regulation events. To use an analogy from sequence search: We need an equivalent to the Smith-Waterman algorithm[21], applicable to structures.

String edit distance [25] clearly is the most successful model in sequence comparison. It is used in document processing, file comparison, molecular sequence analysis, and numerous other applications of approximate string matching. The basic model is that one string is "edited" into another string by a sequence of edit operations, such as single character replacement (R), deletion (D) or insertion (I). The weights associated with the edit operations sum up to an overall score, and the edit sequence giving the minimal score defines the edit distance of the two strings. Equivalently, the editing process, ignoring the order of edit operations, can be represented as an alignment. This equivalence does not generalize to trees, as already mentioned in [1]. For each tree alignment one can construct a corresponding sequence of edit operations, but not vice versa. One can understand editing as finding a largest common sub-structure, while aligning means finding the smallest common super-structure[2]. Which model is favourable depends on the problem.

1.2 Previous work

The first generalization of the edit model from strings to rooted ordered trees is due to [23], algorithmically improved in [31] and implemented and applied to computational biology in [19]. Edit distance models on unordered trees are considered in [29, 32]. Problem variations on rooted and/or unrooted trees are considered in [15, 26, 31]. Algorithms that calculate local similarity of trees in the tree editing model are presented in [26, 28].

An alignment model for trees was first proposed in [12] and a faster algorithm for similar trees is provided in [10]. A problem similar to ours is studied by Wang and Zhang. In [27] they study the *similar consensus problem* for trees: "For fixed k, find two tree patterns (i.e. connected subgraphs) F' and G' of trees F and G within a distance k such that the sum of nodes of F' and G' is maximal". The unit distance function is hard-wired into the algorithm, and the authors conclude that it remains a challenging problem to incorporate more sophisticated cost functions. In analogy to Smith-Waterman [21], we use similarity rather than distance and solve the problem of finding the most similar subforests allowing arbitrary scoring schemes.

[1] They are distinguished by their function: tRNA = transfer RNA, mRNA=messenger RNA, rRNA = ribosomal RNA

[2] In fact, this depends on the scoring scheme.

2 Results

2.1 Outline

The contributions of this article are four-fold:

- We give a systematic generalization of the alignment distance model from strings to trees and forests.
- We introduce several variants of similarity problem on forests and provide efficient algorithms that solve these problems.
- We provide carefully engineered dynamic programming implementations using dense, two-dimensional tables which considerably reduces the space requirement.
- We introduce a new representation for RNA secondary structures as forests which allows reasonable scoring when comparing structures.

Regulatory elements, once known, can be searched for by a variety of approaches. Our local alignment algorithm, by contrast, can discover new conserved structural motifs without prior knowledge about their shape and position. While we exemplify this by a study of iron responsive elements [14], the reader must keep in mind that our approach locates these elements solely because of their structural conservation. As a consequence, it can discover previoulsy unknown, potentially regulatory elements.

2.2 Visualization of structure alignments

RNA secondary structures are represented graphically as circle plots, dot plots, mountain plots or 2D plots. We utilize this pool of methods for drawing alignments of RNA secondary structures. Since bases paired in a structure S_1 can be aligned to bases unpaired in a structure S_2, the presentation of a common secondary structure leaves some choice. For an alignment A of structures S_1, S_2, we draw an RNA secondary structure "S_2-at-S_1" that highlights the differences as deviations of S_2 from S_1, or vice versa, "S_1-at-S_2". Both

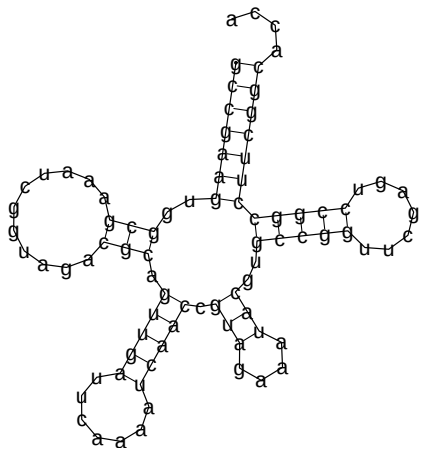


Fig. 1. Secondary structure of the *E.coli* tRNA for leucine, taken from the Genomic tRNA Database [17].

are alternative visualizations of the same alignment A. The drawings can be annotated using all the information of the alignment, e.g. show alternative base pairings.

Figure 2 shows an alignment of the structures of the *E.coli* tRNA for alanine (Anticodon GGC) and for leucine (Anticodon CAA) as found in the Genomic tRNA database [17], in the form S_{Ala}-at-S_{Leu}. The unaligned structures are shown in Figure 1 and 6.

2.3 Experiments

We applied our local structure alignment algorithm to search for regulatory structural motifs in untranslated terminal regions (*UTRs*) of mRNAs. One of the best investigated regulatory motifs in UTRs is the iron responsive element (*IRE*). It is a specific stem-loop structure that can be found in many mRNAs where it regulates for example the translational efficiency of these mRNAs depending on the amount of iron in the cell [8]. UTRs which are known to contain IREs were taken from the UTR data base [18]. Then their structures were predicted with *mfold 3.1* [33]. We always investigated suboptimal structures, because one cannot be sure that the energetically best structure is the

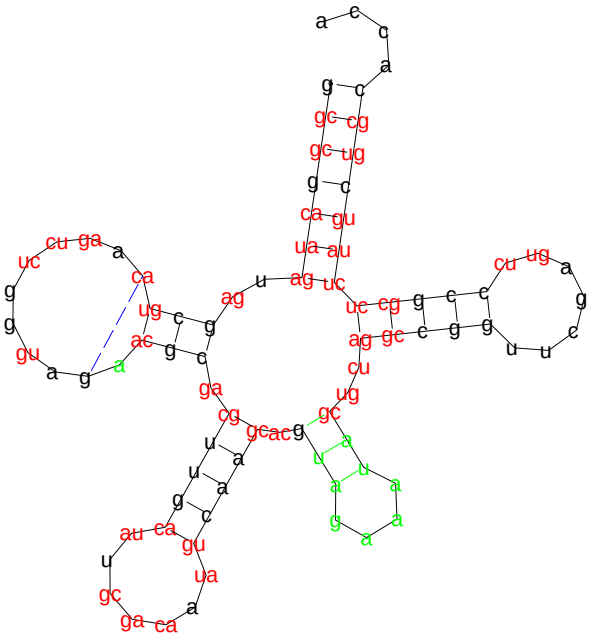


Fig. 2. 2D-plot of the structure alignment of the tRNAs for alanine (Figure 6) and leucine (Figure 1). Bases printed in black show structure elements that occur in both structures with the same sequence. Sequence variantions are displayed by using red letters. Bases or base pairs that can only be found in alanine are printed in blue, while bases that only occur in leucine are printed in green.

biologically correct one. We calculated structure alignments for the predicted structures with our tool using the local alignment option. The reader is encouraged to make his own experiments with the online version of our structure alignment tool, *RNA-forester* at the url `http://bibiserv.techfak.uni-bielefeld.de/rna-forester`.

In Figure 3(a) the local alignment of the 5'UTRs of the human ferritin heavy chain mRNA (5HSA015337) and the mouse ferritin heavy chain mRNA (5MMU002159) is displayed. Here the IRE is not only conserved in structure but also in sequence. In contrast, the local structure alignment of the 5'UTR of the human ferritin heavy chain mRNA and the 3'UTR of the human transferrin receptor mRNA

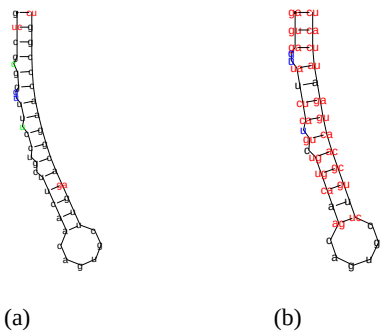


Fig. 3. (a) Local structure alignment of the human and mouse ferritin heavy chain 5'UTRs. (b) Local structure alignment of the human ferritin heavy chain 5'UTR and the human transferrin receptor 3'UTR.

(3HSA008842) (see Figure 3(b)) shows numerous compensatory mutations in the IRE.

This example also demonstrates that we are not restricted to small structures (the ferritin 5'UTRs are in a size range of 200 nucleotides), because here we calculated the local structure alignment of a 5'UTR (208 nucleotides) and a much larger 3'UTR (2464 nucleotides). Although it occurs at completely different positions in the two UTRs, we detected the IRE again (Figure 3(b)).

This shows that our approach can discover arbitrary conserved structural motifs in a larger structure, independent of their position and primary sequence.

3 A Uniform Notion of Alignment and Similarity of Strings, Trees and Forests

3.1 Preliminaries

Let Σ be a set of symbols, the *alphabet*. The *gap symbol* '-', not in Σ, will play the special role to indicate deletions. We define $\Sigma^{-} = \Sigma \cup \{-\}$ and the *pair alphabet* $\Sigma^2 = \Sigma^{-} \times \Sigma^{-} \setminus \{(-,-)\}$.

We consider *rooted, ordered, node-labelled trees*, called *trees* for short. An (ordered) *forest* is a sequence of trees. A function $label$ assigns a label to each node

in a tree or forest. We use $\mathcal{F}(\Sigma)$ for the set of Σ-*labelled* forests. Where convenient, we identify a tree with the forest containing only this tree. A *string* over Σ is a tree in $\mathcal{F}(\Sigma)$ where each node has at most one descendant. This latter definition implies that and how the string case is embedded into our generalization to trees.

3.2 Alignments of structures

In the tree edit model [23], deleting a tree node v means that the children of node v become the children of the parent node of v. Moreover, if v has any siblings, the deletion preserves the preorder relation of these nodes. If v is a root node, then its children have no common ancestor any more, and they split up into a forest. Figure 4 gives an example of the deletion operation. We speak of deletions and insertions when editing T into T', but an insertion into T is nothing but a deletion from T', and hence requires no extra definition. In contrast to the operational view of editing one structure into another, an alignment is a declarative model, a data structure rather than a process.

Our central notion is the following generic view of an alignment: An *alignment* of two structures, with labels from some alphabet, is the same type of structure, with labels from the pair alphabet. Labels of the form $(a, b), (a, \text{-}), (\text{-}, b)$ with $a, b \in \Sigma$ denote the edit operations $R, D,$ and I, respectively. Here this notion applies to (pairs of) strings, trees, and forests. Clearly, it generalizes to graphs, as well as to alignments of more than two items, but this is beyond our present scope. We now formalize this view.

Let $A \in \mathcal{F}(\Sigma^2)$. Its component wise projections $A|_1$ and $A|_2$ are elements of $\mathcal{F}(\Sigma^{\text{-}})$.

Definition 1. *Let $F \in \mathcal{F}(\Sigma^{\text{-}})$. $\pi(F) \in \mathcal{F}(\Sigma)$ is the forest that results from successive deletion of nodes v with $label(v) =$ -.*

It is easy to show that the order of node deletions is irrelevant and thus $\pi(F)$ is uniquely defined.

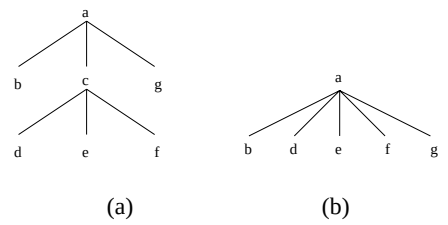


Fig. 4. (a) a tree with nodes $a, \ldots, f$; (b) the tree after node c has been deleted.

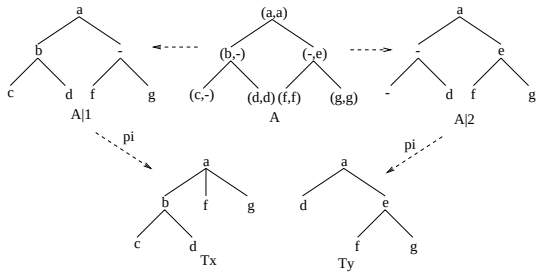


Fig. 5. A is an alignment of F and G.

Definition 2. *Let $F, G \in \mathcal{F}(\Sigma)$. $A \in \mathcal{F}(\Sigma^2)$ is an alignment of F and G iff $F = \pi(A|_1)$ and $G = \pi(A|_2)$.*

Since strings and trees are special cases of forests, Definitions 1 and 2 apply to these as well. An example of a pairwise tree alignment is given in Figure 5.

We now turn to scoring alignments. We are not interested in arbitrary alignments of certain forests, but just in those that satisfy an optimality criterion. For distance problems, optimality means minimality, while for similarity problems optimality means maximality. Distances are not negative, and the distance between two forests is 0 iff the forests are equal. A localized variant of distance makes no sense, as empty forests always have the minimal distance of 0. Similarity is slightly more flexible, allowing for positive and negative score contributions. The similarity of two equal trees is a positive number in most scoring schemes, and we can (and will) ask for the most similar subtrees or subforests of the given forests. Note that

a tree contains many subforests. Hence it makes sense to look not only for the best match to a tree in a forest, but also vice versa.

Given a *scoring function* $\sigma : \Sigma^2 \to \mathbb{R}$, the *similarity score* of an alignment $A \in \mathcal{F}(\Sigma^2)$ is defined by

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

The *global similarity of forests* F *and* G, written as $gs_\sigma(F, G)$, is the maximal score that can be obtained by an alignment of F and G. An alignment of F and G is *optimal* if it achieves this score.

Problem $gs_\sigma(F, G)$: Compute $gs_\sigma(F, G)$ and an optimal alignment of F and G.

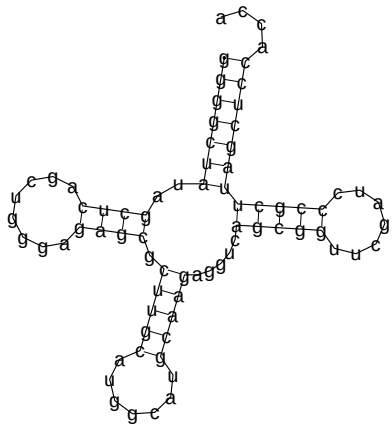


Fig. 6. Secondary structure of the *E.coli* tRNA for alanine taken from the Genomic tRNA Database [17]

3.3 Forest representation of RNA secondary structures

An RNA structure is denoted by an RNA sequence and the set of bases that form bonds. Representing the bonds as arcs over the sequence, an RNA structure is an RNA secondary structure iff the arcs are not crossing. A coarse grained representation of RNA secondary structures which uses the structural elements hairpin loop (H), bulge (B), interior loop (I) and multiloop (M) as its basic elements is proposed in [20]. This encoding produces small forests, but developing a scoring scheme on this level of abstraction is a difficult problem. Following [16] we can represent RNA secondary structures as forests on the level of paired and unpaired bases. The parent and sibling relationship of the forest nodes is determined by the nesting of base pair bonds. The $5'$ to $3'$ nature of the RNA molecule imposes the order among sibling nodes. Figure 6 shows a 2D plot of a RNA molecule and Figure 7 depicts the corresponding forest representation.

Bases that pair in one structure can be unpaired in a related structure because the pair is not stable in terms of energies or a mutation of one base forbids a pairing. Accordingly, the bases that are involved in these events should be replaced by each other. The RNA representation in Figure 7 is not suitable for creating an adequate scoring scheme for these basic events. Clearly, each node of an RNA forest is involved in exactly one edit operation in a forest alignment (This holds for the tree editing approach as well). Since a base pair is encoded as a single node, the score for deleting the pairing between bases a and u would be $\sigma((a, u), \text{-}) + \sigma(\text{-}, a) + \sigma(\text{-}, u)$. We extend the forest representation to allow explicit scoring of base pair deletions. Pairing bases are represented by three connected nodes: The *P-node* stands for the base pair bond and is labelled with P. Its children nodes are ordered according to the $5'$ to $3'$ ordering of the bases and the leftmost and rightmost child are the bases that pair. The children nodes of a P-node can be P-nodes, if they are that are not leftmost or rightmost. Figure 8 gives an example of our *extended* RNA forest representation.

The alphabet of labels of our extended forest representation is $\Sigma = \{P, a, c, g, u\}$. Since we are only interested in comparing the structure of RNAs, we ignore the primary sequence. This is facilitated by a reduced alphabet $\Sigma_{P,B} = \{P, B\}$ where B stands for base-node.[3] We use the following scoring scheme for given constants b_r, b_d, p_r, p_d:

[3] A scoring scheme σ such that $\sigma(a_1, b_1) = \sigma(a_2, b_2)$ for $a_1, a_2, b_1, b_2 \in \{a, c, g, u\}$ has the same effect.

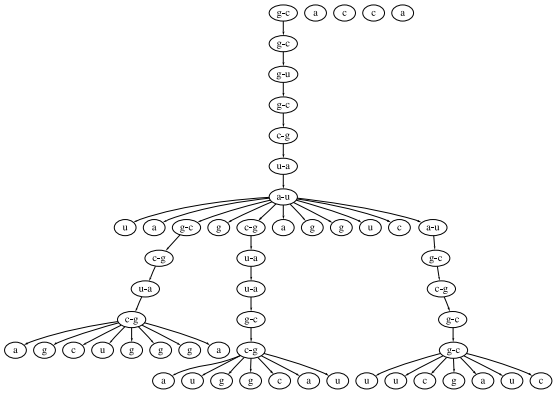


Fig. 7. Forest representation of the RNA shown in Figure 6. Pairing bases correspond to internal nodes which labels are the bases that pair. Unpaired bases correspond to leaf nodes and their label is a single base.

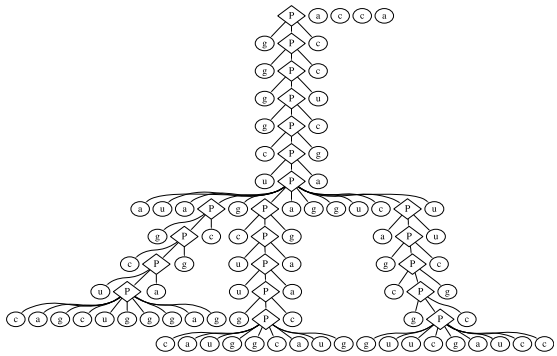


Fig. 8. Extended forest representation of the RNA shown in Figure 6. A base pair is represented explicitly by a P-node. The leftmost and rightmost child of a P-node are the bases that pair.

$\sigma(B, B) = b_r, \sigma(B, \text{-}) = \sigma(\text{-}, B) = b_d, \sigma(P, P) = p_r, \sigma(P, \text{-}) = \sigma(\text{-}, P) = p_d$. A replacement of a P-node and a B-node is not meaningful in our model. Therefore, the scoring contribution for this case must be $\sigma(P, B) = \sigma(B, P) = -\infty$.

In [11] a set of edit operations is suggested that consider both primary sequence and structure of RNA. These are *base-replacement*, *base-indel*, *basepair-replacement*, *basepair-indel*.[4] The latter two operations treat a basepair and the pairing bases as a unit. Calculating the edit distance (not alignment distance) of structures represented as shown in Figure 7 is suggested in [30] and corresponds to an edit model that concerns the four mentioned edit operations. Additionally, [11] extends these edit operations by new ones that consider a basepair separately of the pairing bases. These are *basepair-breaking* which is the deletion of the bond and *basepair-altering* which is the breaking of a bond because a base that pairs in one sequence is deleted in the other. If both bases are deleted, the edit operation would be a basepair-indel. Our model of aligning extended RNA forests provides the tree counterparts for these edit operations, but the scoring is different from [11] for some cases. Figure 9 shows how our model is related to the described edit operations.

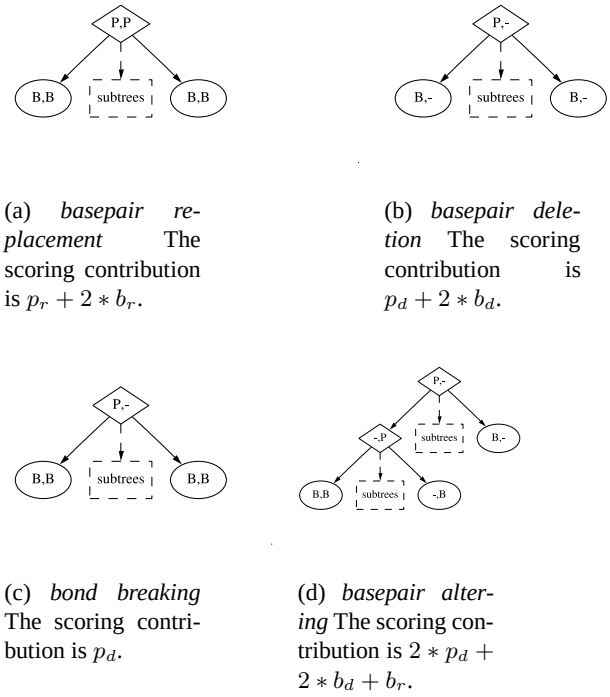
[4] indel stands for *insertion or deletion*

3.4 Local similarity of RNA forests

Calculating global similarity of RNA secondary structures is not sufficient when the focus is to find similar regions. This holds particularly if the regions are at positions that are far apart due to their $5'$ position.

Local similarity means finding the maximal similarity between two substructures. If these substructures are extended, the score decreases. This requires a scoring scheme that balances positive and negative scoring contributions. Otherwise, the similarity of the whole structures would always achieve the maximum score. It is generally assumed that an alignment of two empty structures scores zero.

A substring of a string is a prefix of a suffix, and local similarity on strings means the highest similarity over all pairs of substrings. The problem of finding most similar (complete) suffixes is not of great interest in the domain of strings. Moving from strings to forests, local similarity problems come in a greater variety.



(a) *basepair replacement* The scoring contribution is $p_r + 2 * b_r$.

(b) *basepair deletion* The scoring contribution is $p_d + 2 * b_d$.

(c) *bond breaking* The scoring contribution is p_d.

(d) *basepair altering* The scoring contribution is $2 * p_d + 2 * b_d + b_r$.

Fig. 9. (a)-(d) show how the alignment structure is related to edit operations, and show their scoring contributions.

The key notion for the local similarity problems on trees is the closed subforest:

Definition 3. *A sequence $v_1, \ldots, v_n$ of sibling trees in F such that v_{i+1} is the right brother of v_i for $i \in [1, n-1]$ is a closed subforest (*csf*) of F.*

Note that the empty forest and forest F itself are *csfs* of F. It is quite obvious that F'' is a *csf* of F, if F' is a *csf* of F and F'' is a *csf* of F' (*closed subforest transitivity*).

On trees, the counterpart of a suffix is a subtree. Finding the most similar subtrees *is* an interesting problem, and it generalizes to the following problem.

Problem $lcsfs_\sigma(F, G,)$: The *local closed subforest similarity problem* consists in finding the most similar *csfs* F' and G' of F and G. That is, one seeks $lcsfs_\sigma(F, G) = \max_{F',G'}(gs_\sigma(F', G'))$ over all *csfs* F' and G' of F and G, respectively.

Small-in-large is an intermediate between global and local similarity:

Problem $silcsfs_\sigma(F, G)$: The *small-in-large closed subforest similarity problem* means to match a "small" forest F completely against all *csfs* of the "larger" G. That is, one wants to compute $silcsfs_\sigma(F, G) = \max_{G'}(gs_\sigma(F, G'))$ over all *csfs* G' of G.

Continuing the analogy, the prefix of a (sub)tree T is a tree T' that is obtained by removing subtrees from T. This is called a *tree pattern*, and gives rise to the problem of local pattern similarity on trees and forests which is not considered here.

We can now turn to solve the $lcsfs_\sigma(F, G)$ and the $silcsfs_\sigma(F, G)$ problem.

4 The Dynamic Programming Similarity Algorithm

Dynamic programming is characterized by solving a problem in a recursive fashion and tabulating intermediate results that are re-used. Following the advice of [3, 4], we initially consider recursion and tabulation separately and put them together afterwards. In this way, we reproduce the recurrences of [12], adapted to similarity scoring, in a (as we hope) more lucid fashion. Then we carefully treat tabulation. This is actually the more challenging issue here and interesting modifications and improvements over [12] are achieved. Finally, the basic global similarity algorithm is adapted to the problem variants.

Recall that a forest F is a sequence of trees. Let $|F|$ be the number of nodes in F and $len(F)$ be the number of trees in F, i.e. the length of sequence F. Let $i{:}F$ be the forest consisting of the first i trees of F (*prefix*), while $F{:}j$ is the forest consisting of the last j trees of F (*suffix*). For each node v in F, $pre_F(v)$ is the index of v in a pre-order traversal of F. We use $F[i]$ to identify a node by its index, i.e. $F[pre_F(v)] = v$. If F is not the empty forest, $F[1]$ is the root node of the

first tree in F. Let $F^{\downarrow}$ be the forest consisting of the children trees of $F[1]$ and $F^{\rightarrow} = F{:}(len(F) - 1)$ be the forest of the right sibling trees of $F[1]$. Note that $F^{\downarrow}$ and $F^{\rightarrow}$ can be the empty forest.

4.1 The search space of forest alignments

To calculate similarity of forests, one must consider all their alignments, the *search space*. We can enumerate all alignments of two forests in a structurally recursive fashion. Suppose A is an alignment of F and G. Depending on $label(A[1])$, the possible forests $A^{\downarrow}$ and $A^{\rightarrow}$ are determined. Our case analysis is based on Definition 2.

Lemma 1. *Let A be an alignment of $F, G \in \mathcal{F}(\Sigma)$. If F or G are empty forests, A is either the empty forest, or its labels are solely deletions or solely insertions. If F and G are both non-empty forests, then $label(A[1])$ is of the form $(a, b), (\text{-}, b)$ or $(a, \text{-})$ for some $a, b \in \Sigma$. This leads to the following case distinction:*

1. *If $label(A[1]) = (a, b)$, then the following is true:*
   - *$a = label(F[1])$ and $b = label(G[1])$,*
   - *$A^{\downarrow}$ is an alignment of $F^{\downarrow}$ and $G^{\downarrow}$ and $A^{\rightarrow}$ is an alignment of $F^{\rightarrow}$ and $G^{\rightarrow}$.*
2. *If $label(A[1]) = (a, \text{-})$, then the following is true:*
   - *$a = label(F[1])$,*
   - *for some $r \in [0, len(G)]$, $A^{\downarrow}$ is an alignment of $F^{\downarrow}$ and $r{:}G$ and $A^{\rightarrow}$ is an alignment of $F^{\rightarrow}$ and $G{:}(len(G) - r)$.*
3. *If $label(A[1]) = (\text{-}, b)$, then the following is true:*
   - *$b = label(G[1])$,*
   - *for some $r \in [0, len(F)]$, $A^{\downarrow}$ is an alignment of $r : F$ and $G^{\downarrow}$ and $A^{\rightarrow}$ is an alignment of $F{:}(len(F) - r)$ and $G^{\rightarrow}$.*

Figure 10 gives a graphical view of Lemma 1. The search space of all possible alignments of F and G is determined by cases 1, 2, and 3, and by all possible choices of the split position r in cases 2 and 3.

Scoring the alignments of the search space follows the same structurally recursive pattern. The similarity of F and G is the maximum of the scores $\sigma(a, b)$, $\sigma(a, \text{-})$ and $\sigma(\text{-}, b)$, each added to the similarity scores of the appropriate subforests. Clearly, Bellman's principle of optimality [2] is satisfied. To turn our case analysis into a dynamic programming algorithm, we only need to add tabulation of intermediate results.

Fig. 10. Graphical illustration of Case 1 and 2 of Lemma 1. The shaded triangle stands for $F^{\downarrow}$ and the shaded rectangle for $F^{\rightarrow}$. Each prefix/suffix pair of G is indicated by the vertical line "splitting" G.

4.2 Tabulation

A dynamic programming tabulation method is based on a mapping from subproblems to table indices. In our case, subproblems are defined by subforests, and tabulation is more sophisticated than in the case of string comparison. The following insight considerably simplifies the tabulation problem for forest alignments:

Lemma 2. *All subforests considered in Lemma 1 are closed subforests of F or of G. By the closed subforest transitivity (see Section 3.4), all subforests considered in the recursive search space construction are closed subforests.*

We need a mapping from *csfs* to table indices, which allows for efficient transitions from *csf* F' to $F'^{\downarrow}$ and $F'^{\rightarrow}$, see Lemma 1. For a transparent description of our algorithms, we use a two stage mapping $\beta_F \cdot \alpha_F$. The function α_F provides a mapping from *csfs* of F to index pairs, and β_F maps these index pairs to linear table indices. In this way, we reduce table dimension and space consumption in practice.

For any non-empty *csf* F' of F, we define $\alpha_F(F') = (pre_F(F'[1]), len(F'))$. The empty forest is represented ambiguously by any index pair $(i, 0)$. Figure 11 illustrates this mapping. If (i, j) is an index pair representing a *csf*, then i is called the *node index* and j the *length index*.

Let $noc_F[i]$ be the number of children of $F[i]$ and $rb_F[i]$ be the pre-order index of the right brother node

Fig. 11. The pre-order number of a node is shown in parentheses behind its label. Index pair (5,2) represents the boxed closed subforest.

of $F[i]$. If there is no such right brother, then $rb_F[i] = 0$. If F' is a non-empty *csf* and $\alpha_F(F') = (i, j)$, then $\alpha_F(F'^{\downarrow}) = (i + 1, noc_F[i])$ and $\alpha_F(F'^{\rightarrow}) = (rb_F[i], j - 1)$. That is, $\alpha_F(F'^{\downarrow})$ and $\alpha_F(F'^{\rightarrow})$ can be computed in constant time, given $\alpha_F(F')$. Splitting F' into $r{:}F'$ and $F'{:}\,(len(F') - r)$ yields the subforests represented by (i, r) and $(rb^r_F[i], j - r)$ where rb^r_F is the r-fold application of rb_F. Since the splits will be determined in order of increasing r, the amortized cost of each split is $O(1)$.

Now one can derive matrix recurrences to specify the dynamic programming algorithm calculating forest similarity. We just have to substitute the subforests of Lemma 1 by the corresponding index pairs, and switch from enumeration of the search space to maximization of similarity. A four-dimensional matrix S^4_σ such that $S^4_\sigma(\alpha_F(F'), \alpha_G(G'))$ is the similarity of *csfs* F' and G' of F and G, respectively, would allow straightforward tabulation. If p and q are the maximum numbers of sibling nodes in F and G, respectively, this tabulation technique requires $O(|F| \cdot p \cdot |G| \cdot q)$ space. It wastes space for two reasons:

- The empty forest is represented ambiguously by all index pairs $(i, 0)$.
- Let i' be the number of siblings to the right of $F[i]$ including $F[i]$. Let k' be the number of siblings to the right of $G[k]$ including $G[k]$. For all $i' < j \leq p$ and $k' < l \leq q$, (i, j) and (k, l) do not represent *csfs*, and hence $S^4_\sigma((i, j), (k, l))$ is not used.

The concrete shape of the forests to be aligned determines the number of unused entries in S^4_σ. Even in the best case, when all internal nodes in the trees have the same out-degree p, nearly half of the table is not used. This becomes worse if the node degree varies. Our second stage mapping, β_F, from index pairs to indexes eliminates all unused entries. It is defined by $\beta_F(i, 0) = 0$ and $\beta_F(i, j) = \mathit{offset}_F[i] + j$ for $j \neq 0$, where $\mathit{offset}_F[i]$ is the number of non-empty *csfs* having a node index less than i. Table offset_F can be precomputed in $O(|F|)$ time and space. We define the right inverse β_F^{-1} of β_F by $\beta_F^{-1}(0) = (1, 0)$ and $\beta_F^{-1}(\beta_F(i, j)) = (i, j)$ for $\beta_F(i, j) \neq 0$.

4.3 Implementation based on matrix recurrences

We now combine our previous ideas to give a dense tabulating algorithm calculating global forest similarity. We compute a matrix S_σ defined by

$$S_\sigma(\beta_F(\alpha_F(F')), \beta_G(\alpha_G(G'))) = gs_\sigma(F', G') \qquad (*)$$

for all *csfs* F' and G' of F and G, respectively. Since $\alpha_F(F) = (1, len(F))$ and $\alpha_G(G) = (1, len(G))$, the value in $S_\sigma(\beta_F(1, len(F)), \beta_G(1, len(G)))$ gives the global similarity of F and G. The recurrences for S_σ are given in Figure 12.

To complete the dynamic programming algorithm, we must consider the order of evaluating the entries in S_σ. Each element must be evaluated before it is used. Evaluating S_σ row by row or column by column, as done in the dynamic programming algorithm for string similarity [21], does not work here. Let us consider the data dependencies in the recurrences of Figure 12. Obviously, $S_\sigma(0, 0)$ can be initialized to zero. If $\beta_F(i, j) > 0$, then $S_\sigma(\beta_F(i, j), 0)$ depends on entries $S_\sigma(\beta_F(i{+}1, noc_F[i]), 0)$ and $S_\sigma(\beta_F(rb_F[i], j{-}1), 0)$. That is, either the node index strictly increases, or if $rb_F[i] = 0$, then $j = 1$ and hence $\beta_F(rb_F[i], j - 1) = 0$. If $\beta_G(k, l) > 0$, then the corresponding holds for $S_\sigma(0, \beta_G(k, l))$. If $\beta_F(i, j) > 0$ and $\beta_G(k, l) > 0$, then in the delete case, $S_\sigma(\beta_F(i, j), \beta_G(k, l))$ depends on $S_\sigma(\beta_F(i + 1, noc_F[i]), \beta_G(k, r))$ and $S_\sigma(\beta_F(rb_F[i], j - 1), \beta_G(rb^r_G[k], l - r))$ for some $r \in [0, l]$. Thus either the node index strictly increases, or the length index decreases. The corresponding holds for the insert case. Thus we can evaluate S_σ in decreasing order of the node index and increasing order of the length index. This is done in the DP Algorithm shown in Figure 12. The iteration over the length

$$
S_\sigma(x,y) = \begin{cases}
0 & \text{if } x = 0 \text{ and } y = 0 \quad (1)\\[2ex]
\begin{array}{l}\sigma(label(F[i],\text{-}))\\ +S_\sigma(\beta_F(i+1, noc_F[i]), 0))\\ +S_\sigma(\beta_F(rb_F[i], j-1), 0)\end{array} & \text{if } x > 0 \text{ and } y = 0 \quad (2)\\[2ex]
\begin{array}{l}\sigma(label(\text{-}, G[k]))\\ +S_\sigma(0, \beta_G(k+1, noc_G[k]))\\ +S_\sigma(0, \beta_G(rb_G[k], l-1))\end{array} & \text{if } x = 0 \text{ and } y > 0 \quad (3)\\[2ex]
\max \left\{ \begin{array}{l} replace(x,y)\\ delete(x,y)\\ insert(x,y)\end{array} \right\} & \text{otherwise} \quad (4)
\end{cases}
$$

where $(i,j) = \beta_F^{-1}(x)$ and $(k,l) = \beta_G^{-1}(y)$

$$
\begin{aligned}
replace(x,y) &= \sigma(label(F[i]), label(G[k]))\\
&\quad +S_\sigma(\beta_F(i+1, noc_F[i]), \beta_G(k+1, noc_G[k]))\\
&\quad +S_\sigma(\beta_F(rb_F[i], j-1), \beta_G(rb_G[k], l-1))\\
delete(x,y) &= \sigma(label(F[i]), \text{-})\\
&\quad + \max_{0\le r\le l} \left\{ \begin{array}{l} S_\sigma(\beta_F(i+1, noc_F[i]), \beta_G(k, r))\\ +S_\sigma(\beta_F(rb_F[i], j-1), \beta_G(rb_G^r[k], l-r))\end{array} \right\}\\
insert(x,y) &= \sigma(\text{-}, label(G[k]))\\
&\quad + \max_{0\le r\le j} \left\{ \begin{array}{l} S_\sigma(\beta_F(i, r), \beta_G(k+1, noc_G[k]))\\ +S_\sigma(\beta_F(rb_F^r[i], j-r), \beta_G(rb_G[k], l-1))\end{array} \right\}
\end{aligned}
$$

Algorithm 1.
$S_\sigma(0,0) := 0$
for $i := |F|$ **downto** 1 **do**
  for $j := 1$ **to** $maxcsflen_F[i]$ **do**
    Calculate$S_\sigma(\beta_F(i,j), 0)$
for $k := |G|$ **downto** 1 **do**
  for $l := 1$ **to** $maxcsflen_G[k]$ **do**
    Calculate$S_\sigma(0, \beta_G(k,l))$
for $i := |F|$ **downto** 1 **do**
  for $k := |G|$ **downto** 1 **do**
    for $j := 1$ **to** $maxcsflen_F[i]$ **do**
      for $l := 1$ **to** $maxcsflen_G[k]$ **do**
        Calculate $S_\sigma(\beta_F(i,j), \beta_G(k,l))$

Fig. 12. The recurrences for S_σ and the corresponding DP Algorithm computing the entries of S_σ in an appropriate order. Cases (1)–(3) of the recurrences involving empty forests are obvious. The similarity of two non-empty forests is determined by the maximum score for alignments A that have a replacement, or a deletion, or an insertion at the root. The functions $replace$, $delete$, and $insert$ reflect the case distinction in Lemma 1.

index makes use of a table $maxcsflen_F$, defined by $maxcsflen_F[i] = \max\{j \mid (i,j) \text{ is a } csf \text{ of } F\}$.

Algorithm 1 tabulates $gs_\sigma(F,G)$ of all pairs[5] of *csfs* F' and G' of F and G, see $(*)$. Thus, scanning the matrix S_σ for maximum elements solves the $lcsfs_\sigma(F,G)$ problem. Matrix S_σ also contains the answer to the $silcsfs_\sigma(F,G)$ problem, since $silcsfs_\sigma(F,G) = \max_y(S_\sigma(\beta_F(1, len(F)), y))$ is the sought similarity.

If one is not only interested in the similarity value, but also in optimal alignments, these can be computed

[5] All pairs are required for the local similarity algorithm. Global similarity only requires a subset thereof [12].

by backtracking. To facilitate this, the split position r should be stored with each optimal value resulting from a deletion or an insertion.

4.4 Efficiency analysis

$maxcsflen_F$ can be precomputed in $O(|F|)$ time and space, since $maxcsflen_F[i] = 1$, if $i = |F|$ and $maxcsflen_F[i] = offset_F[i+1] - offset_F[i]$, otherwise. Let $deg(F) = \max\{maxcsflen_F[i] \mid i \in [1, |F|]\}$. Let $p = deg(F)$ and $q = deg(G)$. According to the recurrences of Figure 12, each $S_\sigma(x, y)$ is calculated in $O(p + q)$ time. Since each entry in S_σ is calculated exactly once, the overall time complexity of Algorithm 1 depends on the size of S_σ. This in turn depends on the number of *csfs* in F and G. Consequently, Algorithm 1 runs in $O(|F| \cdot p \cdot |G| \cdot q)$ space and $O(|F| \cdot p \cdot |G| \cdot q \cdot (p + q))$ time. Note that the asymptotic space complexity is not reduced by using dense two-dimensional tables, as proposed above. However, the space reduction can be huge in practice. For example, when comparing the small and the large RNA secondary structure (see last example of Section 2.3), we have handled pairs of trees, for which the four-dimensional table requires 1071 megabytes, while the corresponding dense two-dimensional table requires 39 megabytes of space. This is a 27-fold improvement.

References

1. A. Apostolico and Z. Galil. *Pattern Matching Algorithms*. Oxford University Press, 1997.
2. R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
3. R. Giegerich. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, **16**:665–677, 2000.
4. R. Giegerich and C. Meyer. Algebraic dynamic programming. In Hélène Kirchner and Christophe Ringeissen, editors, *Algebraic Methodology And Software Technology, 9th International Conference, AMAST 2002*, pages 349–364, Saint-Gilles-les-Bains, Reunion Island, France, 2002. Springer LNCS 2422.
5. N.K. Gray and M. Wickens. Control of translation initiation in animals. *Annu. Rev. Cell Dev. Biol.*, **14**:399–458, 1998.
6. J. Guhaniyogi and G. Brewer. Regulation of mRNA stability in mammalian cells. *Gene*, **265**:11–23, 2001.
7. R.R. Gutell, N. Larsen, and C.R. Woese. Lessons from an evolving rRNA: 16s and 23s rRNA structures from comparative perspective. *Microbiol Rev*, **58**:10–26, 1994.
8. M.W. Hentze and L.C. Kuehn. Molecular control of vertebrate iron metabolism: mRNA-based regulatory circuits operated by iron, nitric oxide, and oxidative stress. *Proc. Natl. Acad. Sci. USA*, **93**:8175–8182, 1996.
9. R.P. Jansen. mRNA localization: Message on the move. *Nature Rev Mol Cell Biol*, **2**:247–256, 2001.
10. J. Jansson and A. Lingas. A fast algorithm for optimal alignment between similar ordered trees. *Lecture Notes in Computer Science*, **2089**:232–240, 2001.
11. T. Jiang, G.H. Lin, B. Ma, and K. Zhang. A general edit distance between RNA structures. *J. of Computational Biology*, **9**(2):371–388, 2002.
12. T. Jiang, J.T.L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Science*, **143**(1):137–148, 1995.
13. T. Kiss. Small nucleolar RNAs: An abundant group of noncoding RNAs with diverse cellular functions. *Cell*, **109**:145–148, 2002.
14. R.D. Klausner, T.A. Rouault, and J.B. Harford. Regulating the fate of mRNA: The control of cellular iron metabolism. *Cell*, **72**:19–28, 1993.
15. P. N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium*, number 1461, pages 91–102. Springer-Verlag, Berlin, 1998.
16. S.Y. Le, R. Nussinov, and J.V. Mazel. Tree graphs of RNA secondary structures and their comparison. *Computational Biomedical Research*, **22**:461–473, 1989.
17. T.M. Lowe and S.R. Eddy. tRNAscan-se: A program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Res*, **25**:955–964, 1997.
18. G. Pesole, S. Liuni, G. Grillo, F. Licciulli, F. Mignone, C. Gissi, and C. Saccone. Utrdb and utrsite: specialized database of sequences and functional elements of 5' and 3' untranslated regions of eukaryotic mRNAs. *Nucleic Acids Research*, **30**:335–340, 2002.
19. B. A. Shapiro and K. Zhang. Comparing multiple rna secondary structures using tree comparisons. *Comp. Appl. Biosci.*, **6**(4):309–318, 1990.
20. B.A. Shapiro. An algorithm for comparing multiple RNA secondary stuctures. *Comp. Appl. Biosci.*, **4**(3):387–393, 1988.
21. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, **147**:195–197, 1981.
22. M. Sprinzl, C. Horn, M. Brown, A. Ioudovitch, and S. Steinberg. Compilation of tRNA sequences and sequences of tRNA genes. *Nucleic Acids Research*, **26**:148–153, 1998.

23. K.C. Tai. The tree-to-tree corection problem. *Journal of the ACM*, **26**:422–433, 1979.
24. T. Villa, J.A. Pleiss, and C. Guthrie. Spliceosomal snRNAs:mg(2+)-dependent chemistry at the catalytic core? *Cell*, **109**:149–152, 2002.
25. R.A. Wagner and M.J. Fischer. The string to string correction problem. *Journal of the ACM*, **21**(1):168–173, 1974.
26. J.T.L. Wang, B. A. Shapiro, D. Sasha, K. Zhang, and K. M. Currey. An algorithm for finding the largest approxemately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(8):889–895, 1998.
27. J.T.L. Wang and K. Zhang. Identifying consensus of trees through alignment. *Information Sciences*, **126**:165–189, 2000.
28. J.T.L. Wang, K. Zhang, and Chia-Yo Chang. Identifying approximate common substructures in trees based on a restricted edit distance. *Information Sciences*, **121**:367–386, 1999.
29. K. Zhang. A new editing based distance between unordered labeled trees. In *Proc. CPM Combinatorial Pattern Matching*, number 684, pages 254–265, 1993.
30. K. Zhang. Computing similarity between RNA secondary structures. *IEEE International Joint Symposia on Intelligence and Systems*, pages 126–132, 1998.
31. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, **18**(6):1245–1262, 1989.
32. K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, **42**(3):133–139, 1992.
33. M. Zuker, D.H. Mathews, and D.H. Turner. Algorithms and thermodynamics for RNA secondary structure prediction: A practical guide. *In RNA Biochemistry and Biotechnology, J.Barciszewski & B.F.C. Clark,eds.,NATO ASI Series, Kluwer Academic Publishers*, pages 11–43, 1999.