Fixed Parameter Tractable Alignment of RNA Structures Including Arbitrary Pseudoknots

Mathias Möhl*[1], Sebastian Will*[2], and Rolf Backofen[2]

[1] Programming Systems Lab, Saarland University, Saarbrücken, Germany, mmohl@ps.uni-sb.de

[2] Bioinformatics, Institute of Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany,{will,backofen}@informatik.uni-freiburg.de

Abstract. We present an algorithm for computing the edit distance of two RNA structures with arbitrary kinds of pseudoknots. A main benefit of the algorithm is that, despite the problem is NP-hard, the algorithmic complexity adapts to the complexity of the RNA structures. Due to fixed parameter tractability, we can guarantee polynomial run-time for a parameter which is small in practice. Our algorithm can be considered as a generalization of the algorithm of Jiang *et al.* [1] to arbitrary pseudoknots. In their absence, it gracefully degrades to the same polynomial algorithm. A prototypical implementation demonstrates the applicability of the method.

Keywords: RNA alignment, pseudoknots, fixed parameter tractability

1 Introduction

Over the last years, numerous discoveries attribute to RNA a central role that goes far beyond being a messenger and comprises regulatory as well as catalytic functions [2]. The turn of focus from purely sequence based analysis, as largely applied for DNA and proteins, to structure based analysis, as required for RNA, imposes a challenge to bioinformatics.

For this reason, RNA sequence/structure alignment is a rich and active field of research [1, 3–6]. Almost all current approaches rely on the assumption that the pseudoknot-free representation of RNA structures suffices to obtain reasonable alignments. This is justified, algorithmically, since this restriction allows for an efficient treatment, as well as biologically, since the function of an RNA-molecule is mainly determined by its pseudoknot-free, secondary structure, which is usually more conserved than its sequence. Recent findings at least question the assumption that pseudoknots can be neglected. Today, it is known that many natural RNA molecules not only contain pseudoknots, but that these pseudoknots have diverse and important functions in the cell [7] and are therefore highly conserved [8]. Moreover, the concrete alignment of the pseudoknot region is of major interest, since pseudoknots often occur at the functional centers of RNAs.

Many problems associated with the prediction or alignment of structures with arbitrary pseudoknots are NP-hard [1, 9]. To overcome the limitation to

* These authors contributed equally.

pseudoknot-free structures, but still maintain a complexity that is affordable in practice, one has several alternatives. A first approach is to consider only a restricted class of pseudoknots, which allows a polynomial algorithm [10–12]. Second, there are heuristic approaches which are usually fast, but which are not guaranteed to find the optimal structure or do not give a performance guarantee, or both [6]. Here we will follow a third direction, namely to design an algorithm that can align arbitrary pseudoknots, always computes optimal structures and nevertheless has a performance guarantee in terms of fixed parameter tractability. Whereas polynomial runtime cannot be guaranteed in general for NP-hard problems, unless P=NP, fixed parameter tractability allows to guarantee polynomial runtime if some parameter, which is usually small on practical instances, is considered as constant.

We present an algorithm that computes the optimal alignment of two RNA structures with respect to their edit distance. The parameter determining the exponential runtime depends on how complex the crossing stems are arranged and is small in practice. As a nice property, the algorithm gracefully degrades to the algorithm of Jiang *et al.* [1] for the simpler class of structures handled by their algorithm.

Related Work. Most of the algorithms for RNA sequence structure alignment are not able to align pseudoknots [13, 5, 3, 1].

Among the algorithms supporting pseudoknots, there are several grammar-based approaches for motif finding, which try to align a sequence with given structure to a sequence with unknown structure (usually a genomic sequence)[14, 15]. In these approaches, the class of supported pseudoknots depends on the expressivity of the underlying grammar formalism.

Concerning the alignment of two pseudoknotted structures, Evans [9] developed a fixed parameter tractable algorithm that computes the longest arc preserving common subsequence of two sequences with arbitrary kinds of pseudoknots. This problem is related to edit distance. However, on input classes where our algorithm guarantees polynomial run-time due to the fixed parameter, the run-time of Evan's algorithm is not polynomially bounded.

Another algorithm by Evans [12] finds the maximum common ordered substructure of two RNA structures in polynomial time (more precisely in $O(n^{10})$ time and $O(n^8)$ space, where n denotes the length of the sequences), but only for a restricted class of pseudoknots.

Bauer *et al.* [6] give an algorithm based on integer linear programming with Lagrangian relaxation that aligns two sequences with arbitrary pseudoknots. As a heuristic approach, it works usually well in practice but gives no guarantees on performance and may even fail to yield optimal results.

Furthermore, there is a fixed parameter tractable algorithm by Blin *et al.* [16] for protein design involving RNA, which shares an important idea with our approach, namely the bipartitioning of the complete set of base pairs into an efficiently tractable subset and the remaining "hard" base pairs (in our case, pairs of base pairs).

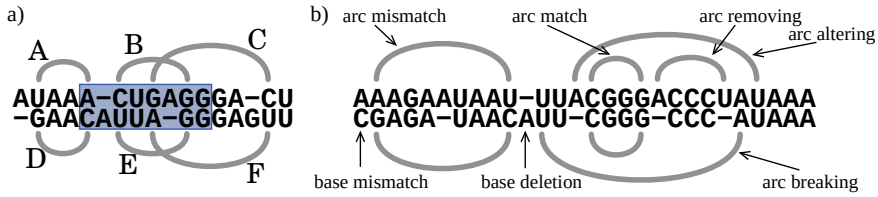

Fig. 1. a) Realized arc pairs (A,D), (B,E), and (C,F). (A,D) and (C,F) are open for the highlighted subalignment. b) Edit operations (cf. [1])

2 Preliminaries

An *arc-annotated sequence* is a pair (S, P), where S is a string over the set of bases $\{A, U, C, G\}$ and P is a set of arcs (l, r) with $1 \leq l < r \leq |S|$ representing bonds between bases, such that each base is adjacent to at most one arc, i.e. $\forall (l, r) \neq (l', r') \in P : l \neq l' \wedge l \neq r' \wedge r \neq l' \wedge r \neq r'$. We denote the i-th symbol of S by $S[i]$. For an arc $p = (l, r)$, we denote its left end l and right end r by p^{L} and p^{R}, respectively.

For an arc-annotated sequence (S, P), an arc $p \in P$ is called *crossing* if there is an arc $p' \in P$ such that $p^{\mathrm{L}} < p'^{\mathrm{L}} < p^{\mathrm{R}} < p'^{\mathrm{R}}$ or $p'^{\mathrm{L}} < p^{\mathrm{L}} < p'^{\mathrm{R}} < p^{\mathrm{R}}$. In the first case, p is called *right crossing*, in the second case *left crossing*; p and p' form a *pseudoknot*. An arc-annotated sequence (S, P) containing crossing arcs is called *crossing*, otherwise *non-crossing* or *nested*.

For two arc annotated sequences (S_1, P_1) and (S_2, P_2), we define χ, ψ_1, ψ_2:

$$\chi(i, j) := \text{ if } S_1[i] \neq S_2[j] \text{ then 1 else 0,}$$
$$\psi_k(i) := \text{ if } \exists j : (i, j) \in P_k \text{ or } (j, i) \in P_k \text{ then 1 else 0} \quad (\text{for } k = 1, 2).$$

An *alignment* A of two arc-annotated sequences (S_1, P_1) and (S_2, P_2) is a set $A \subseteq [1..|S_1|] \times [1..|S_2|]$ of *alignment edges* such that for all $(i, j), (i', j') \in A$ holds 1.) $i > i'$ implies $j > j'$ and 2.) $i = i'$ if and only if $j = j'$. For an alignment A and i, i', j, j' such that neither (i, j) nor (i', j') cuts any alignment edge (formally $A \cap [i..i'] \times [1..|S_2|] = A \cap [1..|S_1|] \times [j..j']$), we define the *subalignment* $A(i, i'; j, j')$ *of* A by $A \cap [i..i'] \times [j..j']$. An *arc pair* is a pair of arcs $a = (p_1, p_2) \in P_1 \times P_2$. We call $a = (p_1, p_2)$ *realized by* A if and only if $(p_1^{\mathrm{L}}, p_2^{\mathrm{L}}), (p_1^{\mathrm{R}}, p_2^{\mathrm{R}}) \in A$, i.e. when the arcs p_1 and p_2 are *matched by* A. The *set* $\mathrm{OA}(A; i, i'; j, j')$ *of open arc pairs of a subalignment* $A(i, i'; j, j')$ *in* A is the set of arc pairs (p_1, p_2) that are realized by A and where either $p_1^{\mathrm{L}} < i \leq p_1^{\mathrm{R}} \leq i'$ and $p_2^{\mathrm{L}} < j \leq p_2^{\mathrm{R}} \leq j'$ or $i \leq p_1^{\mathrm{L}} \leq i' < p_1^{\mathrm{R}}$ and $j \leq p_2^{\mathrm{L}} \leq j' < p_2^{\mathrm{R}}$. In Fig. 1a), we show realized arc pairs and a subalignment; its set of open arc pairs is $\{(A, D), (C, F)\}$.

Each alignment has an associated cost based on an edit distance with two classes of operations. The operations are illustrated in Fig. 1b). Base operations (mismatch and insertion/deletion) work solely on positions that are not incident to an arc. *Base mismatch* replaces a base with another base and has associated cost w_m. A *base insertion/deletion* removes or adds one base and costs w_d.

The second class consists of operations that involve at least one position that is incident to an arc. An *arc mismatch* replaces one or both of the bases incident to an arc. It costs $\frac{w_{am}}{2}$ if one base is replaced or w_{am} if both are replaced. An *arc breaking* removes one arc and leaves the incident bases unchanged. The associated cost is w_b. *Arc removing* removes one arc and both incident bases and costs w_r. Finally, *arc altering* removes one of the two bases that are incident to an arc and costs w_a.

An alignment A has a corresponding minimal sequence of edit operations. The cost of A is defined as the sum of the cost of these edit operations.

3 A Fixed Parameter Tractable Algorithm

The algorithm we present computes the minimum cost alignment of two arc annotated sequences (S_1, P_1) and (S_2, P_2) containing arbitrary pseudoknots. In terms of Jiang *et al.* [1], we solve EDIT(CROSSING,CROSSING) for their class of *reasonable* scoring schemes. These schemes are restricted by $w_a = \frac{w_b}{2} + \frac{w_r}{2}$.

The central idea of the algorithm is to partition the set of arc pairs $P_1 \times P_2$ into a set NC of "non-crossing" arc pairs and a set of "crossing" arc pairs $\mathrm{CR} = P_1 \times P_2 - \mathrm{NC}$ such that the algorithm can interleave a polynomial alignment method for the arc pairs in NC with an exponential method for the arc pairs in CR. The exact requirement for such a partition is made precise in the definition of "valid partition".

The immediate result is a fixed parameter tractable algorithm whose parameter is loosely understood as the number of arc pairs in CR that cover a common base match. The presented algorithm further reduces this factor substantially by precomputing the alignment of stems of arcs in CR.[3]

3.1 Partition into Crossing and Non-Crossing Arc Pairs

Two arcs p and p' of a sequence cross, iff $p^{\mathrm{L}} < p'^{\mathrm{L}} < p^{\mathrm{R}} < p'^{\mathrm{R}}$ or $p'^{\mathrm{L}} < p^{\mathrm{L}} < p'^{\mathrm{R}} < p^{\mathrm{R}}$. To generalize this from arcs to arc pairs, we define the left and right end point of an arc pair as

$$\nwarrow\!(p_1, p_2) = (p_1^{\mathrm{L}}, p_2^{\mathrm{L}}) \qquad \text{and} \qquad \searrow\!(p_1, p_2) = (p_1^{\mathrm{R}}, p_2^{\mathrm{R}}),$$

respectively. On those points we consider the partial order $\prec$ defined as $(x_1, y_1) \prec (x_2, y_2)$ if and only if $x_1 < x_2$ and $y_1 < y_2$.

Two arc pairs a and a' *cross*, iff $\nwarrow\!a \prec \nwarrow\!a' \prec \searrow\!a \prec \searrow\!a'$ or $\nwarrow\!a' \prec \nwarrow\!a \prec \searrow\!a' \prec \searrow\!a$. Figure 2 represents arc pairs as rectangles in the plane whose dimensions correspond to the two sequences. If two arc pairs cross, the rectangles partially overlap, but note that the converse implication does not hold. In Fig. 2 for example, (D, I) and (E, J) cross, whereas (D, I) and (E, G) do not cross.

[3] In principle, the idea can be extended from stems to arbitrary non-crossing substructures that are, like stems, closed by an inner and an outer arc. At the cost of precomputation this lowers the exponential factor of the algorithm further.

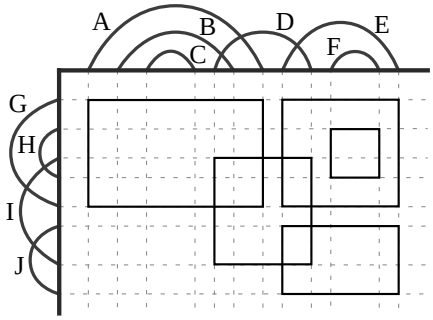

Fig. 2. Visualization of the arc pairs of two sequences. The first sequence has arcs A to E, the second sequence arcs G to J. To maintain readability, only some of the arc pairs are visualized.

Definition 1 (valid partition). *A (bi-)partition of $P_1 \times P_2$ into NC and CR is* valid *if and only if for all $a, a' \in NC$ it holds that a and a' do not cross.*

A valid partition of $P_1 \times P_2$ can be lifted from a partition of the arcs of P_1 and P_2 by choosing appropriate sets $\mathrm{CR}_1 \subseteq P_1$ and $\mathrm{CR}_2 \subseteq P_2$ such that $P_1 - \mathrm{CR}_1$ and $P_2 - \mathrm{CR}_2$ are non-crossing and set $\mathrm{CR} = \mathrm{CR}_1 \times \mathrm{CR}_2$. However, this does not work for arbitrary non-crossing sets $P_1 - \mathrm{CR}_1$ and $P_2 - \mathrm{CR}_2$. For example, in Fig. 2 choosing $\mathrm{CR} = \{A, B, E\} \times \{I\}$ is not valid, since it contains none of the two crossing arc pairs (A, G) and (D, I). Valid partitions are obtained, if CR_1 and CR_2 contain all left crossing edges.

Lemma 1 (sufficient criterion for a partition). *The partition of $P_1 \times P_2$ into $CR = \{\, p_1 \in P_1 \mid p_1$ is left crossing$\,\} \times \{\, p_2 \in P_2 \mid p_2$ is left crossing$\,\}$ and $NC = P_1 \times P_2 - CR$ is valid.*

The claim holds since for two arbitrary crossing arc pairs one of them is in CR: for arc pairs a, a' with $\nwarrow a \prec \nwarrow a' \prec \searrow a \prec \searrow a'$ the two arcs of a' are left crossing. Analogously, a valid partition is obtained, if CR_1 and CR_2 contain all right crossing arcs.

Since our algorithm handles arc pairs in NC more efficient than arc pairs in CR, the partition into NC and CR is crucial for the runtime. A good partition should be minimal in the sense that it becomes invalid, if any element is removed from CR. Finding the best partition among these local minima involves balancing several parameters, since not only the cardinality of CR influences the complexity. Thinking of the arc pairs in CR as rectangles (as indicated in Fig. 2), both the area of the rectangles and the number of rectangles that overlap in a common point influence the runtime.

The partition according to Lem. 1 is not yet aware of these aspects and sometimes does not lead to a local minimum. As an example, in Fig. 2 we would have $\mathrm{CR} = \{D, E\} \times \{I, J\}$, but (E, J) can safely be added to NC, since it only crosses with $(D, I) \in \mathrm{CR}$. The fact that no other arc pair containing arc E or J

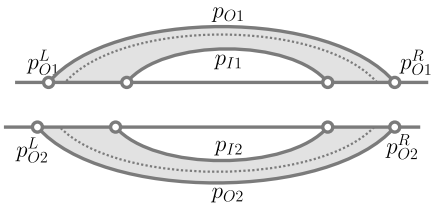

Fig. 3. Stem pair $(a_O, a_I) = ((p_{O1}, p_{O2}), (p_{I1}, p_{I2}))$, which covers the dotted arc pair.

can be removed from CR indicates that this is a general limitation of partitions of arc pairs that are lifted from partitions of arcs. Instead of using these partitions directly, they could serve as a starting point for further optimization at the level of arc pairs with techniques like stochastic local search or genetic algorithms.

In the following we assume a valid partition of the arc pairs into CR and NC.

3.2 Precomputation of Stem Pairs

In order to align whole stems in one step, we group arc pairs in CR into pairs of stems. We define a stem Q in P (for $P \in \{P_1, P_2\}$) as a set of arcs $\{p_1, \ldots, p_k\} \subseteq P$ with $p_1^{\mathrm{L}} < \cdots < p_k^{\mathrm{L}} < p_k^{\mathrm{R}} < \cdots < p_1^{\mathrm{R}}$ such that no end of arcs in $P - Q$ is in one of the intervals $[p_1^{\mathrm{L}}..p_k^{\mathrm{L}}]$ or $[p_k^{\mathrm{R}}..p_1^{\mathrm{R}}]$. In Fig. 2, for example, $\{A, B\}$ is a stem, but $\{A, B, C\}$ is not, since the left end of D is between the right endpoints of B and C. Note that, according to this notion, stems are allowed to include bulges and internal loops and do not need to be maximal.

The *stem pair* of two stems $Q_1 \subseteq P_1$ and $Q_2 \subseteq P_2$ is characterized by the pair (a_O, a_I) of arc pairs, where $a_O = (p_{O1}, p_{O2})$ is the pair of the outermost arcs and $a_I = (p_{I1}, p_{I2})$ is the pair of the innermost arcs of Q_1 and Q_2, i.e. Q_k consists of the arcs $P_k \cap [p_{Ok}^{\mathrm{L}}..p_{Ik}^{\mathrm{L}}] \times [p_{Ik}^{\mathrm{R}}..p_{Ok}^{\mathrm{R}}]$ $(k = 1, 2)$ (cf. Fig. 3). The stem pair *covers* an arc pair a iff $a \in Q_1 \times Q_2$. A stem pair is *realized in an alignment* A if and only if a_O and a_I are realized in A.

We write the set of all stem pairs (a_O, a_I) where $\{a_O, a_I\} \subseteq \mathrm{CR}$ as $\mathrm{ST}_{\mathrm{CR}}$.[4] A stem pair (a_O, a_I) is *open for a subalignment* $A(i, i'; j, j')$ *in* A if and only if a_O and a_I are open for $A(i, i'; j, j')$ in A. The *set of maximal open stem pairs of* $A(i, i'; j, j')$ *in* A is the smallest set M of open stem pairs of $A(i, i'; j, j')$ in A such that each $a \in \mathrm{OA}(A; i, i'; j, j')$ is covered by a stem pair in M.

For each $(a_O, a_I) \in \mathrm{ST}_{\mathrm{CR}}$, we precompute the cost to align the respective stem pair as the value of an item $S(a_O, a_I)$. More precisely, for $a_O = (p_{O1}, p_{O2})$ and $a_I = (p_{I1}, p_{I2})$, the value of $S(a_O, a_I)$ is the cost to align $S_1[p_{O1}^{\mathrm{L}}] \ldots S_1[p_{I1}^{\mathrm{L}}]$ to $S_2[p_{O2}^{\mathrm{L}}] \ldots S_2[p_{I2}^{\mathrm{L}}]$ and simultaneously $S_1[p_{I1}^{\mathrm{R}}] \ldots S_1[p_{O1}^{\mathrm{R}}]$ to $S_2[p_{I2}^{\mathrm{R}}] \ldots S_2[p_{O2}^{\mathrm{R}}]$. In this sense, an S item describes the cost of two subalignments that are not independent of each other due to arcs in CR.

[4] We assume that the arc pairs of a stem pair are either completely contained in CR or completely contained in NC, since minimal partitions (as well as partitions according to Lem. 1) satisfy this property.

$$
S'(i,i';j,j';a_I) = \\
\min \begin{cases}
S'(i+1,i';j,j';a_I) + w_d + \psi_1(i)(\frac{w_r}{2} - w_d) & \text{(gap)} \\
S'(i,i';j+1,j';a_I) + w_d + \psi_2(j)(\frac{w_r}{2} - w_d) & \text{(gap)} \\
S'(i,i'-1;j,j';a_I) + w_d + \psi_1(i')(\frac{w_r}{2} - w_d) & \text{(gap)} \\
S'(i,i';j,j'-1;a_I) + w_d + \psi_2(j')(\frac{w_r}{2} - w_d) & \text{(gap)} \\
S'(i+1,i';j+1,j';a_I) + \chi(i,j)w_m + (\psi_1(i) + \psi_2(j))\frac{w_b}{2} & \text{(align bases)} \\
S'(i,i'-1;j,j'-1;a_I) + \chi(i',j'l)w_m + (\psi_1(i') + \psi_2(j'))\frac{w_b}{2} & \text{(align bases)} \\
\text{if } ((i,i'),(j,j')) \in \mathrm{CR} & \\
\quad S'(i+1,i'-1;j+1,j'-1;a_I) + (\chi(i,j) + \chi(i',j'))\frac{w_{am}}{2} & \text{(align arcs)}
\end{cases}
$$

Fig. 4. Recursion equation to compute S' items

The computation of S items is based on temporary items $S'(i,i';j,j';a_I)$ that correspond to $S(((i,i'),(j,j'));a_I)$ if $((i,i'),(j,j'))$ is an arc pair, but are not limited to this case. $S'(i,i';j,j';((i_a,i'_a),(j_a,j'_a)))$ is invalid if $i > i_a$, $i' < i'_a$, $j > j_a$ or $j' < j'_a$. The alignment of the innermost arc is computed as $S'(i,i';j,j';((i,i'),(j,j'))) = (\chi(i,j) + \chi(i',j'))\frac{w_{am}}{2}$ and step by step enlarged with the recursions given in Fig. 4, where implicitly recursive cases relying on invalid items are skipped.

By the recursion for S', only the arc pair a_I is guaranteed to be realized in the precomputed optimal stem alignments. However, we want to consider in the core dynamic programming algorithm only items $S(a_O,a_I)$ where a_I *and* a_O are realized, in order to avoid ambiguity in the recursion. Therefore, we define items where a_O is not realized as invalid. In consequence, cases referring to these items are skipped in the core algorithm.

3.3 Core of the Algorithm

The main part of the algorithm recursively computes costs of subalignments. The recursions are given in Fig. 6 and an illustration is provided in Fig. 5.

The subalignment costs are represented by items $D(i,i';j,j'|M)$ where i, i', j, and j' specify the range of the subalignment and $M \subseteq \mathrm{ST}_{\mathrm{CR}}$ is its set of maximal open stem pairs. The precise semantics is that the value of $D(i,i';j,j'|M)$ is the minimal cost among all subalignments $A(i,i';j,j')$ of all alignments A that satisfy (a) M is the set of maximal open stem pairs of $A(i,i';j,j')$ in A and (b) for all $(a_O,a_I) \in M$ the precomputed subalignment corresponding to $S(a_O,a_I)$ is a subalignment of A. A helpful intuition of M in the D items is that one end of the stem pairs in M is aligned and the other half is required to be aligned outside of the range $(i,i';j,j')$.

The semantics is reasonable only for a restricted class of items, which we call *valid items*. $D(i,i';j,j'|M)$ is valid if $i' \geq i-1$, $j' \geq j-1$, and there is an alignment A such that M is the set of maximal open stem pairs of $A(i,i';j,j')$ in A.

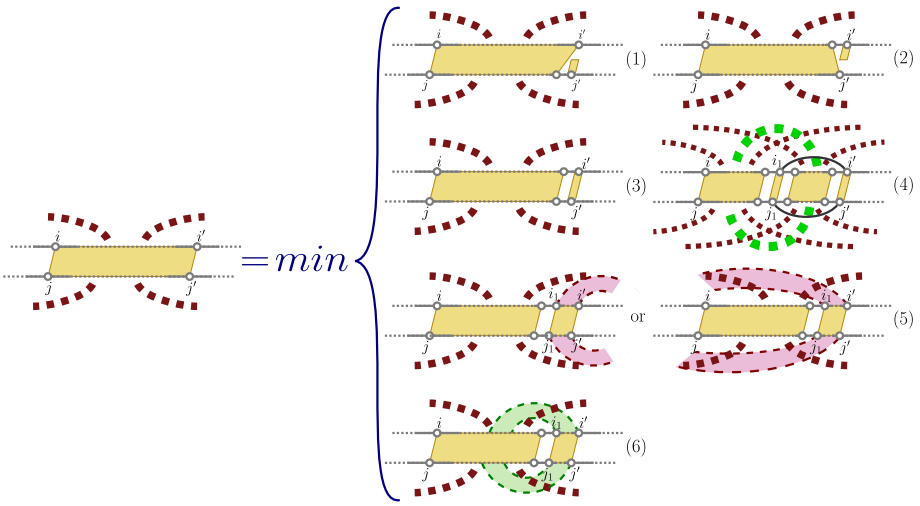

Fig. 5. Illustration of the recursion for computing items $D(i, i'; j, j'|M)$. The red dotted arcs represent the set of open stem pairs M. Case (4) recurses to $D(i, i_1-1; j, j_1-1|M_1)$ and $D(i_1+1, i'-1; j_1+1, j'-1|M_2)$. There, the green dotted arcs represent the set of stem pairs shared between the two alignment fragments (i.e. $M_1 \cap M_2$) and the red dotted arcs represent the remaining elements of $M_1 \cup M_2$, which make up M. Cases (5) and (6) show concrete stem pairs in light red (in M) and light green (not in M), respectively.

Given the semantics of D items, the cost of the entire alignment is the value of $D(1, |S_1|; 1, |S_2|\ |\emptyset)$. It is computed following the recursion in Fig. 6 with base cases $D(i, i-1; j, j-1|\emptyset) = 0$ (for all i, j). Implicitly, in each recursive step the cases involving invalid items are skipped.

We will take a closer look at the cases of the recursion. First note that for $\mathrm{CR} = \emptyset$, only items of the form $D(i, i'; j, j'|\emptyset)$ are valid. Then, the cases (5) and (6) are always skipped and the recursion degenerates to the recursion of Jiang *et al.* [1], shown in Fig. 7, where the items $D(i, i'; j, j'|\emptyset)$ directly correspond to matrix entries $DP(i, i'; j, j')$.[5]

In the absence of crossing arcs, the recursion in Fig. 7 is correct since the case distinction is exhaustive and each case assigns the correct cost. To see this assume an optimal alignment A with a subalignment $A(i, i'; j, j')$ without open arc pairs. Considering the positions i' and j', there are exactly the following cases directly corresponding to the recursion. (1) i' is not aligned in A. If i' is not adjacent to an arc this is due to a base deletion with cost w_d. Otherwise, the arc is either removed or altered, which causes cost $w_r/2$. (2) j' is not aligned in A, analogously. (3) $(i', j') \in A$, but A realizes no arc pair involving (i', j').

[5] Note that the restrictions $i > i_1, j > j_1$ in case 4 of the recursion in Fig. 7 is implicit in our recursion by skipping cases with invalid items; here $D(i, i_1 - 1; j, j_1 - 1|M_1)$ is invalid.

$$
\begin{aligned}
&D(i,i';j,j'|M) = \min \\
&\Bigg\{
\begin{array}{ll}
D(i,i'-1;j,j'|M) + w_d + \psi_1(i')(\frac{w_r}{2} - w_d) & (1)\\
D(i,i';j,j'-1|M) + w_d + \psi_2(j')(\frac{w_r}{2} - w_d) & (2)\\
D(i,i'-1;j,j'-1|M) + \chi(i',j') \cdot w_m + (\psi_1(i') + \psi_2(j'))\frac{w_b}{2} & (3)\\
\text{if there exist some } i_1,\ j_1 \text{ with } ((i_1,i'),(j_1,j')) \in \mathrm{NC} & \\
\quad \min \left\{ \begin{array}{l} D(i,i_1-1;j,j_1-1|M_1) \\ +D(i_1+1,i'-1;j_1+1,j'-1|M_2) \\ +(\chi(i_1,j_1)+\chi(i',j'))\frac{w_{am}}{2} \end{array} \middle| \begin{array}{l} M_1, M_2 \subseteq \mathrm{ST_{CR}}, \text{ where} \\ M = (M_1 \cup M_2) - (M_1 \cap M_2) \end{array} \right\} & (4)\\
\text{if there exists some } (a_O,a_I) \in M \text{ with} & \\
\quad \nwarrow a_O = (i_1,j_1) \wedge \nwarrow a_I = (i',j') \text{ or } \ \searrow a_I = (i_1,j_1) \wedge \searrow a_O = (i',j') & \\
\quad D(i,i_1-1;j,j_1-1|M - \{(a_O,a_I)\}) + \frac{S(a_O,a_I)}{2} & (5)\\
\min \left\{ \begin{array}{l} D(i,i_1-1;j,j_1-1|M \cup \{(a_O,a_I)\}) \\ +\dfrac{S(a_O,a_I)}{2} \end{array} \middle| \begin{array}{l} (a_O,a_I) \in \mathrm{ST_{CR}}, \text{ where} \\ \searrow a_O = (i',j') \text{ and} \\ \searrow a_I = (i_1,j_1) \end{array} \right\} & (6)
\end{array}
\end{aligned}
$$

Fig. 6. Recursion equation to compute D items

All adjacent arcs are broken, each causing cost $w_b/2$. If $S_1[i']$ and $S_2[j']$ mismatch this causes additional cost w_m. (4) $(i',j') \in A$ and A realizes an arc pair $((i_1,i'),(j_1,j'))$ with right end (i',j'). Then, the cost of the whole arc pair is charged and the subalignment is decomposed into a subalignment before the arc pair and the subalignment inside the arc pair.

Note that due to the assumption that $A(i,i';j,j')$ has no open arc pairs the case where $(i',j') \in A$ and A realizes an arc pair with left end (i',j') can not occur. Furthermore, all cases only decompose into subalignments without open arc pairs. In particular in case (4), the two subalignments can not have open arc pairs since such arc pairs would be open arc pairs of both subalignments and then cross the arc pair $((i_1,i'),(j_1,j'))$.

The key to understand the recursion in Fig. 6 (illustrated in Fig. 5) is that it maintains the decomposition of the algorithm of Jiang *et al.* as much as possible in the presence of crossing arc pairs. Then, Jiang's case (4) is no more exhaustive and has to be extended to consider all cases where the recursive subalignments contain open arc pairs. To achieve this, we make the open arc pairs explicit via the additional component M of our items. In principle, it suffices to directly represent the set of open arc pairs in M. For efficiency reasons, we combine open arc pairs into open stem pairs in order to keep the sets M small. As a direct consequence of making the open stem pairs explicit, we can exhaustively minimize over all alternatives in case (4). In particular, these include the cases where M_1 and M_2 contain open stem pairs that are not contained in M, namely those where $M_1 \cap M_2 \neq \emptyset$.

$$DP(i,i';j,j') = \min$$
$$\begin{cases} DP(i,i'-1;j,j') + w_d + \psi_1(i')(\frac{w_r}{2} - w_d) & (1) \\ DP(i,i';j,j'-1) + w_d + \psi_2(j')(\frac{w_r}{2} - w_d) & (2) \\ DP(i,i'-1;j,j'-1) + \chi(i',j') \cdot w_m + (\psi_1(i') + \psi_2(j'))\frac{w_b}{2} & (3) \\ \text{if there exist some } i < i_1,\ j < j_1 \text{ with } ((i_1,i'),(j_1,j')) \in P_1 \times P_2 & \\ \quad DP(i,i_1-1;j,j_1-1) + DP(i_1+1,i'-1;j_1+1,j'-1) & (4) \\ \quad +(\chi(i_1,j_1) + \chi(i',j'))\frac{w_{am}}{2} & \end{cases}$$

Fig. 7. Recursion equation for the algorithm of Jiang *et al.*

For cases (1) to (3), M is invariant since no arc pairs are realized in these cases. In consequence, the generalization of these cases is straightforward.

In order to make our case distinction exhaustive for CR $\neq \emptyset$, we need additional cases (5) and (6) that cover the situations where an arc pair a in CR has left or right end in (i',j') (recall that only the arc pairs in NC are handled in case (4)). There are two such cases: either a is open in $A(i,i';j,j')$ or not. In the first case, the maximal open stem pair that covers a is contained in M and hence uniquely determined. We can therefore decompose into (the respective subalignment of) this maximal open stem pair and the remaining subalignment, where this stem pair is no more open (case (5)). In the second case, we minimize over all possible maximal open stem pairs that cover a. Each time, we decompose again into (the respective subalignment of) this maximal open stem pair and the remaining subalignment, where now the stem pair is open in this remaining subalignment (case (6)). Note that we distribute the cost of the precomputed stem pairs equally among the two subalignments. This is correct, since it is guaranteed that each alignment contains either both subalignments or none of them. Further note that, when descending in the recursion, open stem pairs are introduced via cases (4) or (6) and are removed again via case (5).

When the cost of the alignment is determined, the actual alignment can be constructed by the usual backtracing techniques.

3.4 Complexity

Let n be $\max(|S_1|,|S_2|)$, let s and s' be the maximal number of arcs and bases in a crossing stem, respectively. For an item $S(a_O,a_I)$ we have $O(n^2s^2)$ possible instances: for a_O, we can freely choose among the $O(n^2)$ arc pairs in CR and for a_I we have $O(s^2)$ possible choices, since the arcs of a_O and a_I must belong to the same stems. Analogously, for the S' items we need $O(n^2s'^4)$ space. Since each of these items can be computed in constant time, the time complexity coincides with the required space.

An item $D(i,i';j,j'|M)$ has $O(n^4)$ possible instances of i,i',j,j', but analogously to the algorithm of Jiang *et al.* [1] only $O(n^2)$ of them need to be main-

tained permanently. To measure the number of instances of M, we need the notion of the *crossing number of a point* $(x, y) \in [1..|S_1|] \times [1..|S_2|]$, defined as $\mathrm{C}(x, y) = |\{\, (a_O, a_I) \in \mathrm{ST}_{\mathrm{CR}}^{\mathrm{MAX}} \mid \nwarrow a_I \prec (x, y) \prec \searrow a_I \,\}|$, where $\mathrm{ST}_{\mathrm{CR}}^{\mathrm{MAX}}$ denotes the subset of $\mathrm{ST}_{\mathrm{CR}}$ that only contains pairs of maximal stems (with respect to set inclusion). We denote the maximal crossing number with k. Since each maximal stem pair has $O(s^4)$ fragments, there are at most $O((s^4)^{\mathrm{C}(i,j)+\mathrm{C}(i'j')}) = O(s^{8k})$ possible instances of M for fixed i, j, i', j'.[6] Hence we need to compute $O(n^4 s^{8k})$ D items and maintain $O(n^2 s^{8k})$ of them in memory at the same time.

The computation of a D item needs only for the recursive alternatives (4) and (6) of Fig. 6 more than constant time. In alternative (6), iteration over all $O(s^2)$ possible instances of a_I is necessary and in alternative (4) we need to iterate over all possible instances of M_1 and M_2. Since M_2 is uniquely determined by M and M_1, there are $O(s^{8k})$ of these instances. The computation of all the $O(n^4 s^{8k})$ D items hence requires $O(n^4 s^{8k} \cdot s^{8k}) = O(n^4 s^{16k})$ time.

If at least one of two sequences does not contain pseudoknots, the only minimal partition is $\mathrm{CR} = \emptyset$ and $\mathrm{NC} = P_1 \times P_2$. In this case the algorithm gracefully degrades to the one of Jiang *et al.* [1] requiring $O(n^4)$ time and $O(n^2)$ space.

4 Practical Evaluation

We implemented a prototype of the algorithm in C++ to evaluate its applicability in practice. With the prototype, we computed pairwise alignments of some RNA structures of the tmRNA database [17]. For our evaluation we have chosen the longest tmRNA sequence (*Mycobacteriophage Bxz1*, MB), the shortest sequence (*Cyanidium caldarium*, CC), the sequence that contains the largest crossing stems (*Ureaplasma parvum*, UP), and a nested version (UPnest) of the latter, where we removed all left crossing arcs.

We were able to compute the pairwise alignments of these sequences with 1 GB of memory with one exception using 2 GB. Table 1 shows that the runtime scales well with the complexity of the involved pseudoknots. As we suggested, the exponential factor k is small on all instances. Whereas alignments of sequences with large pseudoknots take several hours, sequences with small pseudoknots can be aligned in a few minutes. In contrast, sequence length has a much smaller impact on runtime, as in particular the alignments with UPnest show.

For the results in Table 1 we partitioned into NC and CR according to the left crossing stem criterion (see Lem. 1). However, the runtime can depend heavily on the partition into NC and CR. For example the alignment of *Ureaplasma parvum* and *Mycobacteriophage Bxz1* took less than three hours if we chose CR to contain the pairs of left crossing arcs, but more than 6 hours if we chose the right crossing arcs instead. Notably, in this case the better partitioning can be identified in advance by comparing the parameters k and s; k is equal for both cases, s is 10/7 for the left crossing and 12/12 for the right crossing case. This comparison indicates that a more sophisticated partitioning into crossing

[6] This is a coarse estimate, that counts many invalid requirement sets, in particular those, where some stem pairs cannot be realized in the same aligment.

Table 1. Runtime of the alignments (on a single Xeon 5160 processor with 3.0 GHz) and the properties of the aligned structures (n=sequence length, s=max. number of arcs in crossing stem, pk=number of pseudoknots, k=fixed parameter of the algorithm) for left crossing partitioning.

aligned sequences	n	s	k	pk	memory	runtime
UP / UP	413/413	10/10	1	4/4	$\leq$ 2 GB	726m 52s
UP / MB	413/437	10/7	1	4/2	$\leq$ 1 GB	172m 53s
UP / CC	413/254	10/2	1	4/1	$\leq$ 1 GB	11m 51s
UP / UPnest	413/413	10/0	0	4/0	$\leq$ 1 GB	4m 43s
MB / MB	437/437	7/7	1	2/2	$\leq$ 1 GB	43m 20s
MB / CC	437/254	7/2	1	2/1	$\leq$ 1 GB	3m 56s
MB / UPnest	437/413	7/0	0	2/0	$\leq$ 1 GB	3m 27s
CC / CC	254/254	2/2	1	1/1	$\leq$ 1 GB	1m 11s
CC / UPnest	254/413	2/0	0	1/0	$\leq$ 1 GB	2m 6s
UPnest/UPnest	413/413	0/0	0	0/0	$\leq$ 1 GB	4m 21s

and nested arc pairs, e.g. greedy or stochastic local optimization, may result in significant speed-ups in practice.

Finally note that the efficiency could be improved further by heuristic optimizations as utilized in many existing alignment tools. For example, skipping the computation of items that are unlikely to contribute to the optimal alignment can significantly reduce computation time.

5 Conclusion

We have presented an algorithm that is able to align RNA structures with arbitrary pseudoknots using a general edit distance for reasonable scoring schemes. The algorithm is fixed parameter tractable and our prototypical implementation shows its applicability in practice.

A central insight due to our method is that pseudoknots can be effectively handled by partitioning the RNA structure into a set of "easy" and "difficult" interactions. Then, expensive, exponential computation can be restricted to the "difficult" part, whereas state-of-the art polynomial methods can be applied to the "easy" part. Furthermore, since for alignment the dynamic programming recursions operate on pairs of sequences even more effective partitionings can be obtained on the level of arc pairs instead of lifting partitions on single arcs.

The idea of partitioning and making this level of abstraction explicit in the algorithm offers possibilities for further optimization. First, since the concrete partition strongly impacts the run-time, optimizing the partition is worth investigating. Second, one obtains heuristic versions of our algorithm by filtering out unlikely arc pairs.

Acknowledgments. We thank Marco Kuhlmann and the anonymous reviewers for useful comments. M. Möhl is funded by the German Research Foundation.

References

1. Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. J. Comput. Biol. **9**(2) (2002) 371–88
2. Couzin, J.: Breakthrough of the year. Small RNAs make big splash. Science **298**(5602) (2002) 2296–7
3. Siebert, S., Backofen, R.: MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons. Bioinformatics **21**(16) (2005) 3352–9
4. Havgaard, J.H., Torarinsson, E., Gorodkin, J.: Fast pairwise structural RNA alignments by pruning of the dynamical programming matrix. PLoS Comput. Biol. **3**(10) (2007) 1896–908
5. Will, S., Reiche, K., Hofacker, I.L., Stadler, P.F., Backofen, R.: Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. PLoS Comput. Biol. **3**(4) (2007) e65
6. Bauer, M., Klau, G.W., Reinert, K.: Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. BMC Bioinformatics **8** (2007) 271
7. Staple, D.W., Butcher, S.E.: Pseudoknots: RNA structures with diverse functions. PLoS Biol. **3**(6) (2005) e213
8. Theimer, C.A., Blois, C.A., Feigon, J.: Structure of the human telomerase RNA pseudoknot reveals conserved tertiary interactions essential for function. Mol. Cell **17**(5) (Mar 2005) 671–682
9. Evans, P.A.: Finding common subsequences with arcs and pseudoknots. In: CPM '99: Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching, London, UK, Springer-Verlag (1999) 270–280
10. Rivas, E., Eddy, S.R.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. J Mol Biol **285**(5) (Feb 1999) 2053–2068
11. Reeder, J., Giegerich, R.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinformatics **5** (Aug 2004) 104
12. Evans, P.A.: Finding common RNA pseudoknot structures in polynomial time. In: Combinatorial Pattern Matching (CPM 2006). Volume 4009/2006 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2006) 223–232
13. Sankoff, D.: Simultaneous solution of the RNA folding, alignment and protosequence problems. SIAM J. Appl. Math. **45**(5) (1985) 810–825
14. Matsui, H., Sato, K., Sakakibara, Y.: Pair stochastic tree adjoining grammars for aligning and predicting pseudoknot RNA structures. Bioinformatics **21**(11) (Jun 2005) 2611–2617
15. Dost, B., Han, B., Zhang, S., Bafna, V.: Structural alignment of pseudoknotted RNA. In: 10th Annual International Conference on Research in Computational Biology (RECOMB 2006). (2006) 143–158
16. Blin, G., Fertin, G., Hermelin, D., Vialette, S.: Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. In: Graph-Theoretic Concepts in Computer Science, WG 2005. Volume 3787 of Lecture Notes in Computer Science., Springer (2005) 271–282
17. Zwieb, C., Gorodkin, J., Knudsen, B., Burks, J., Wower, J.: tmRDB (tmRNA database). Nucleic Acids Res **31**(1) (Jan 2003) 446–447