Fast Detection of Common Sequence Structure Patterns in RNAs

Rolf Backofen and Sven Siebert

Friedrich-Schiller Universität
{backofen,siebert}@inf.uni-jena.de
Ernst-Abbe Platz 2
07743 Jena, Germany

Abstract We developed a dynamic programming approach of computing common sequence/structure patterns between two RNAs given by their sequence and secondary structures. Common patterns between two RNAs are meant to share the same local sequential and structural properties. Nucleotides which are part of an RNA are linked together due to their phosphodiester or hydrogen bonds. These bonds describe the way how nucleotides are involved in patterns and thus delivers a bond-preserving matching definition. Based on this definition, we are able to compute all patterns between two RNAs in time $O(nm)$ and space $O(nm)$, where n and m are the lengths of the RNAs, respectively. Our method is useful for describing and detecting local motifs and for detecting local regions of large RNAs although they do not share global similarities. An implementation is available in C++ and can be obtained by contacting one of the authors.

1 Introduction

RNAs are polymers consisting of the four nucleotides A,C,G and U which are linked together by their phosphodiester bonds. Bases which are part of the nucleotides form hydrogen bonds within the same molecule leading to structure formation. One major challenge is to find (nearly) common patterns in RNAs since they suggest functional similarities of these molecules. For this purpose, one has to investigate not only sequential features, but also structural features. Finding common RNA motifs is currently a hot topic in bioinformatics since RNA has been identified as one of the most important research topics in life sciences. RNA was selected as *the* scientific breakthrough of the year 2002 by the reader of the science journal.

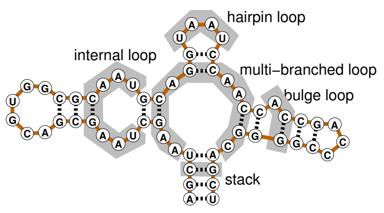

Figure 1. Structure elements of an RNA secondary structure.

Most approaches on finding RNA sequence/structure motifs are based on (locally) aligning two RNAs of lengths n. They use dynamic programming methods with a high complexity between $O(n^4)$ and $O(n^6)$ ([1], [9]). Hence, these approaches are suited for RNAs with just moderate sizes. For that reason, we want to use a general approach that is inspired by the DIALIGN [10] method for multiple sequence alignments. The basic idea is to find exact patterns in large RNAs first, and then to locally align only subsequences containing many exact patterns by using a more complex approach like [1].

So far, the problem of finding local, exact common sequence/structure patterns was unsolved. This is the problem which is considered in this paper. We can list all patterns between two RNAs in time $O(nm)$ and space $O(nm)$, where n and m are the lengths of the RNAs, respectively. The key idea is a dynamic programming method that describes secondary structures not only as base pairing interactions but at a higher level of structure elements known as hairpin loops, right bulges, left bulges, internal loops or multi-branched loops (see Figure 1). The computation of RNA patterns is performed on loop regions from inside to outside. Base-pairs which enclose loops occur in a nested fashion, i.e nested base-pairs fulfill for any two base-pairs (i_1, i_2) and (j_1, j_2) either $i_1 < i_2 < j_1 < j_2$ or $i_1 < j_1 < j_2 < i_2$.

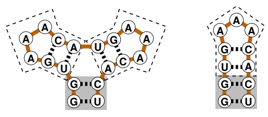
Figure 2. Alternative matching

A naive attempt is to consider all combinations of positions i in the first RNA and positions j in the second RNA and to extend these starting patterns by looking at neighbouring nucleotides sharing the same sequential and structural properties. If these properties are fulfilled then the nucleotides are taken into the pattern. At a first glance, this idea may work, but the crucial point are the loops. Consider e.g. the case shown in Figure 2. Suppose the algorithm starts at position 1 in the first RNA and position 1 in the second RNA and is working towards the multiple loop in the first RNA. The lower stem has been successfully matched. But now there is no clear decision to match the upper part of the stem-loop of the second RNA either to the left side or to the right side of the multiple loop. This decision depends on how a common pattern is defined, of course, and how to reach a maximally extended pattern. Therefore, the only solution is to make some precomputations of sequential and structural components of RNAs. Finally, we end up in a dynamic programming approach which compares inner parts of RNAs first, stores the results in different matrices and build up the solutions successively. Note, that it is also a mistake to compute common sequential parts first and then to recompose these parts by their structural properties. This problem is obviously a computational intractable problem because of considering all combinations of subsets of sequence parts.

Related Work: Wang et al.[13] published an algorithm for finding a largest approximately common substructure between two trees. This is an inexact pattern matching algorithm suitable for RNA secondary structures. A survey of computing similarity between RNAs with and without secondary structures until 1995 is given by Bafna et al.[2]. Gramm et al. [5] formulated the arc-preserving problem : given two nested RNAs S_1 and S_2 with lengths n and m $(n \geq m)$, respectively, does S_2 occurs in S_1 such that S_2 can be obtained by deleting bases from S_1 with the property that the arcs are preserved ? This problem can not be seen as biological motivated because the structure of S_2 would be found splitted in S_1. It has been shown by Jiang et al. [8] that finding the longest common arc-preserving subsequence for arc-annotated sequences (LAPCS), where at least one of them has crossing arc structure is MAXSNP-hard. Exact pattern matching on RNAs has been done by Gendron et al. [4]. They propose a backtracking algorithm, similar to an algorithm from Ullman [11] solving the subgraph isomorphism problem from graph theory. It aims at finding recurrent patterns in one RNA.

The paper is organized as follows : In section 2, we introduce the reader into definitions and notations of RNAs. In section 3, we define matchings between two RNAs such that they can be described by matching and matched paths. In Section 4, a bond preserving matching is proposed which is used for the dynamic programming matrices (section 5). The matrices are computed by recursion equations in section 6. The pseudo code is given in section 7.

2 Definitions and Notations

An *RNA* is a tuple (S, P), where S is a string of length n over the alphabet $\Sigma = \{A, C, G, U\}$. We denote $S(i)$ as the base at position i. P is a set of base-pairs (i, i'), $1 \leq i < i' \leq n$, such that $S(i)$ and $S(i')$ are complementary bases. Here, we refer to Watson-Crick base-pairs A—U and C—G, as well as the non-standard base-pair G—U. In the following, we write $i \overset{P}{\textbf{—}} i'$ instead of $(i, i') \in P$ meaning that the two bases S(i) and S(i') are linked together by a bond. For the rest of the paper, we restrict our set of base-pairs to secondary structures holding the following property : for any two base-pairs (i, i') and (j, j') either $i < i' < j < j'$ (*independent*) or $i < j < j' < i'$ (*nested).* The nestedness condition allows us to partially order the bases of an RNA.

Definition 1 (Stacking Order). *Let* (S, P) *be an RNA. The* stacking order *of a base S(i) (abbr. as stord*$_P(i)$*) is the number of bonds* $k \overset{P}{\textbf{—}} l$ *with* $k < i < l$, *plus one.*

Hence, we are able to partition a secondary structure into structure elements with the same stacking order. We call them loops. See e.g. Figure 1 for various loop names. For our algorithmic approach, we have to look at neighbouring bases belonging to the same loop. This is achieved by a function right (left) of an RNA (S, P), with $\text{right}_P(i) = j$ if $(i, j) \in P$, and $i + 1$ otherwise. $\text{left}_P(i)$ is defined analogously. The function $\text{right}_P^k(i)$ (resp. $\text{left}_P^k(i)$) is a short term of applying the right function (resp. left) to i k-times. We define $\text{rbd}_P(i)$ (resp. $\text{lbd}_P(i)$) to be true if there is a bond $i \overset{P}{\textbf{—}} i'$ (resp. $i' \overset{P}{\textbf{—}} i$), false otherwise. Thus, we can describe loops mathematically as follows. Let (S, P) be an RNA. The *loop* (written $\text{loop}(i \overset{P}{\textbf{—}} i')$) which is enclosed by a bond $i \overset{P}{\textbf{—}} i'$ is the set of positions $\{r \mid i < r < i' \wedge \exists k : r = \text{right}_P^k(i)\}$.

3 Matchings

Suppose we are given two RNAs (S_1, P_1) and (S_2, P_2). The sets $V_1 = \{i \mid 1 \leq i \leq |S_1|\}$ and $V_2 = \{j \mid 1 \leq j \leq |S_2|\}$ contains the positions of both RNAs.

Definition 2 (Matching). *A* matching M *between two RNAs* (S_1, P_1) *and* (S_2, P_2) *is a set of pairs* $M = \{(i,j) \mid i \in V_1 \wedge j \in V_2\}$ *which describes a partial bijection from* V_1 *to* V_2 *and satisfies the following conditions:*

1. ***structure cond.:*** $\forall (i,j) \in M$, $rbd_{P_1}(i) \Leftrightarrow rbd_{P_2}(j) \wedge$ $lbd_{P_1}(i) \Leftrightarrow lbd_{P_2}(j)$
2. ***base cond.:*** $\forall (i,j) \in M$, $S_1(i) = S_2(j)$

The matching definition is applied to single bases. Since bases are sometimes part of base-pairs, we may see them as units given as an additional bond condition:

3. **bond cond.:** for each $\{(i,j),(i',j')\} \subseteq M$ with $i \overset{P_1}{\textbf{—}} i'$ and $j \overset{P_2}{\textbf{—}} j'$ it follows $S_1(i) = S_2(j) \wedge S_1(i') = S_2(j')$

The range of the first RNA is given as the set $\text{ran}_1(M) = \{i \mid \exists j : (i,j) \in M\}$. It describes the pattern found in the first RNA which is matched to the same pattern in the second RNA. Given an element $i \in \text{ran}_1(M)$, we denote $M(i)$ as the uniquely determined element j with $(i,j) \in M$. Similarly, given an element $j \in \text{ran}_2(M)$, we denote $M^{-1}(j)$ as the uniquely determined element i with $(i,j) \in M$.

The first two points of the definition can be easily written as a matching predicate between two bases at positions i and j : $\text{match}(i,j) = [S(i) = S(j)] \wedge [\text{lbd}_{P_1}(i) \leftrightarrow \text{lbd}_{P_2}(j)] \wedge [\text{rbd}_{P_1}(i) \leftrightarrow \text{rbd}_{P_2}(j)]$. The bond condition provides a structure conserving requirement based on base-pairs. It can be extended by a bond checking such that the predicate of $\text{match}(i \overset{P_1}{\textbf{—}} i', j \overset{P_2}{\textbf{—}} j')$ is given as

$$[i \overset{P_1}{\textbf{—}} i'] \wedge [j \overset{P_2}{\textbf{—}} j'] \wedge [S_1(i) = S_2(j)] \wedge [S_1(i') = S_2(j')]$$

The matching conditions are applied to single bases or base-pairs so far. Now, we want to merge bases and base-pairs such that special relations among them are fulfilled. They provide a definition for matchings. We make use of a *transition type* function on two positions i and i' which is $+1$, -1 or 0 depending on whether $i = i' + 1$, $i = i' - 1$ or $i \overset{P}{\textbf{—}} i'$. A path in an RNA is a sequence of positions $i_1 \dots i_k$, such that the bases $S(i_l)$ and $S(i_{l+1})$ for $l = 1, \dots, k-1$ are connected due to the bond conditions or due to the backbone of this RNA.

Definition 3 (Matching/Matched Path). *Let* (S_1, P_1) *and* (S_2, P_2) *be two RNAs and M a matching between them. An* M-matching path *is a list of pairs* $(i_1, j_1) \dots (i_k, j_k) \in M$ *such that 1.)* $i_1 \dots i_k$ *is a path in* (S_1, P_1)*; 2.)* $j_1 \dots j_k$ *is a path in* (S_2, P_2)*; and 3.) for each* $1 \leq l < k$ *the transition types of* (i_l, i_{l+1}) *and* (j_l, j_{l+1}) *are equal. A matching is* connected *if there is a* M*-matching path between any two pairs in* M*. A path in only one of the RNAs consisting of only matched bases is called* M-matched path.

Note the difference between *matching* paths and *matched* paths. A matched path is a path occurring in one structure, but there must not be necessarily a corresponding path in the other structure. Furthermore, the restriction of *matching* paths to some structure clearly produces a *matched* path. But the contrary is not true. There are matched paths, where the image of the path (under the matching) is not a path in the other structure. To clarify this, consider

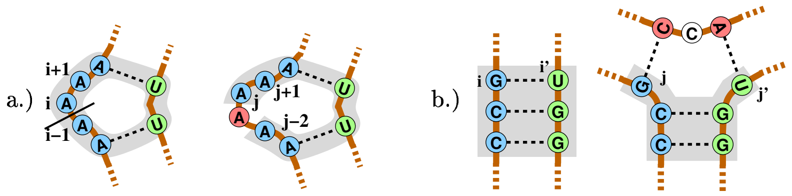

Figure 3. Unpreserved bonds (backbone and secondary). a.) the backbone bonds $i-1, i$ is not preserved. b.) the bond $i \overset{P_1}{\boldsymbol{-\!\!-}} i'$ is not preserved. The matching is indicated by blue and green nodes. In both cases, the the corresponding bases in the second structure are connected with nodes (in red) that are not part of the matching.

the simplest matched paths, which are edges (backbone connections or bonds) between matched bases. By definition, they are matched paths, but there might not be a matching path associated with. This happens for bases which mark the "ends" of the matching. The two cases for backbone edges and bond edges are shown in Figure 3.

From the definitions of matchings, it is not clear whether they respect the backbone order, i.e. $i < i'$ implies $M(i) < M(i')$. One can show that this holds for connected matchings. Since we will restrict ourself to matchings that preserve bonds later, and the proofs are simpler for these kind of matchings, we omit the proof for the general case here. We treat only the simple case for preserving the stacking order for general connected matchings.

Proposition 1. *Let $(i_1, j_1) \dots (i_k, j_k) \in M$ be a matching path. Then the path preserves the relative stacking order, i.e. for all $1 \le r \le k$ we have $stord_{P_1}(i_1) - stord_{P_2}(j_1) = stord_{P_1}(i_r) - stord_{P_2}(j_r)$.*

4 Bond Preserving Matching

As Figure 3b indicates, a matched bond $i \overset{P_1}{\boldsymbol{-\!\!-}} i'$ which does not correspond to a matching path only occurs if we have a stem in the first structure that is matched to a multiple loop in the second structure (or vice versa). This is biologically unwanted, since it is very unlikely that this pattern could have been generated by evolution. For that reason, we are interested in matchings that preserve bonds.

Definition 4 (Bond-Preserving Matching). *A connected matching M is said to be bond-preserving if every matched bond in P_1 or P_2 is also a matching path, i.e. if $\{(i, j), (i', j')\} \subseteq M$ and $i \overset{P_1}{\boldsymbol{-\!\!-}} i'$, then $j \overset{P_2}{\boldsymbol{-\!\!-}} j'$, and vice versa.*

In the following, we will consider bond-preserving matchings. We say that a connected, bond-preserving matching M is *maximally extended*, if there is no M' such that $M \subsetneq M'$. We are interested in finding all (non-overlapping) maximally extended matchings. For this purpose, we need to show some properties. We start with a proposition that allows us to decompose the problem of finding a maximally extended matching into subproblems of finding maximally extended loop matchings. The next proposition shows that the backbone order is respected. And the third proposition shows that if we do not exceed a loop, then maximally extended matchings (in this loop) are uniquely determined by one element.

Proposition 2. *Let $i, i' \in loop(r \overset{P_1}{\textbf{—}} s)$, and let M be a bond-preserving matching with $\{(i,j),(i',j')\} \subseteq M$. Then any shortest matching path between (i,j) and (i',j') uses only elements of $loop(r \overset{P_1}{\textbf{—}} s) \cup \{r,s\}$.*

Proposition 3 (Backbone Order). *Let M be a connected matching, and $(i,j),(i',j') \in M$. Then $i < i'$ if and only if $j < j'$.*

Proposition 4. *Let $i \overset{P_1}{\textbf{—}} i'$ and $j \overset{P_2}{\textbf{—}} j'$ be two bonds, and let $r \in loop(i \overset{P_1}{\textbf{—}} i')$ and $s = loop(j \overset{P_2}{\textbf{—}} j')$. Let M, M' with $i, i' \notin ran_1(M) \cup ran_1(M')$ and $(r,s) \in M \cap M'$. Then $ran_1(M) \cup loop(i \overset{P_1}{\textbf{—}} i') = ran_1(M') \cup loop(i \overset{P_1}{\textbf{—}} i')$ and $ran_2(M) \cup loop(j \overset{P_2}{\textbf{—}} j') = ran_2(M') \cup loop(j \overset{P_2}{\textbf{—}} j')$.*

5 Dynamic Programming Matrices

We want to find all non-overlapping, maximally extended, bond-preserving matchings. For overlapping matchings, we choose the one with maximal size. If there are overlapping matchings of the same size, then only one is selected.

We use a dynamic programming approach by filling a matrix $M(r,s)$, with the following interpretation. We define an order $\prec$ on elements as follows:

$$i \prec j \equiv \begin{cases} i < j & \text{if } \mathrm{stord}_{P_1}(i) = \mathrm{stord}_{P_1}(j) \\ \mathrm{stord}_{P_1}(i) < \mathrm{stord}_{P_1}(j) & \text{otherwise} \end{cases}$$

For pairs (r,s) and (k,l) we define $(r,s) \prec (k,l)$ if and only if $r \prec k$. Then

$$M(r,s) = \max\left\{ |M| \;\middle|\; \begin{array}{l} \text{M is a maximally extended matching} \\ \text{with } (r,s) \in M \text{ and there is no} \\ (r',s') \in M \text{ with } (r',s') \prec (r,s) \end{array} \right\}$$

contains the size of an maximal matching. For simplicity, we assume the maximum value over an empty set to be 0. Note that the size is stored only for the left-most, bottom-most pair (r,s) in M. For calculating $M(r,s)$, we will additionally need auxiliary matrices M^{r-end}, M^{bb} and M^{rb}, which are defined as follows.

Definition 5 (Auxiliary Matrices). *Let $R_1 = (S_1, P_1)$ and $R_2 = (S_2, P_2)$ be two RNAs. Let r (resp. s) be an element of $loop(i \overset{P_1}{\textbf{—}} i')$ (resp. $loop(j \overset{P_2}{\textbf{—}} j')$).*

Then $M^{loop}_{\prec}(r,s)$ is the size of the maximal matching within the loops that contain (r,s), and is extended to the right or above (r,s), i.e.

$$M^{loop}_{\prec}(r,s) = \max\left\{ |M| \;\middle|\; \begin{array}{l} M \subseteq [i..i'] \times [j..j'] \textit{ is a connected} \\ \textit{matching with } (r,s) \in M \textit{ and} \\ \forall (r',s') \in M \backslash \{(i,i'),(j,j')\} : (r,s) \preceq (r',s') \end{array} \right\}$$

In addition, we define for every i,j such that $i \overset{P_1}{\textbf{---}} i'$ and $j \overset{P_2}{\textbf{---}} j'$ the matrix element $M^{bb}(i \overset{P_1}{\textbf{---}} i', j \overset{P_2}{\textbf{---}} j')$ to be the maximal matching that matches the bonds $i \overset{P_1}{\textbf{---}} i'$ and $j \overset{P_2}{\textbf{---}} j'$, i.e.

$$M^{bb}(i \overset{P_1}{\textbf{---}} i', j \overset{P_2}{\textbf{---}} j') = \max\left\{ |M| \;\middle|\; \begin{array}{l} M \subseteq [i..i'] \times [j..j'] \textit{ is a} \\ \textit{connected matching with} \\ (i,j) \in M \text{ and } (i',j') \in M \end{array} \right\}$$

In addition, we define $M^{rb}(i \overset{P_1}{\textbf{---}} i', j \overset{P_2}{\textbf{---}} j')$ to be the maximal matching containing the right partners i' and j' of the bonds only, i.e.

$$M^{rb}(i \overset{P_1}{\textbf{---}} i', j \overset{P_2}{\textbf{---}} j') = \max\left\{ |M| \;\middle|\; \begin{array}{l} M \in [i+1..i'] \times [j+1..j'] \\ \textit{is a connected matching} \\ \textit{with } (i',j') \in M \end{array} \right\}$$

The first procedure calculates $M^{loop}_{\prec}(r,s)$ for a matching of two loops associated with the bonds $i \overset{P_1}{\textbf{---}} i'$ and $j \overset{P_2}{\textbf{---}} j'$, given that $M^{loop}_{\prec}$, M^{bb} and M^{rb} is already calculated for all bonds that are contained in the two loops. For calculating $M^{bb}(i \overset{P_1}{\textbf{---}} i', j \overset{P_2}{\textbf{---}} j')$, we use additional auxiliary variables. The variable $RDist$ stores the loop distance to the right-end of the loop. Thus, for given $RDist$, we consider elements r and s which have distance $RDist$ to i' and j', respectively. Looking from the right end (i',j') of the loop this implies that $r = \text{left}^{RDist}_{R_1}(i')$ and $s = \text{left}^{RDist}_{R_2}(j')$.

First, we need to know whether there is a matching connecting (r,s) with the right ends of the loop (i',j'):

$$Reach^{r-end}(RDist) = \begin{cases} \text{true} & \text{if } \exists \text{ connected matching } M \subseteq [i..i'] \times [j..j'] \\ & \text{with } (r,s) \in M \text{ and } (i',j') \in M \\ \text{false} & \text{otherwise} \end{cases} \qquad (1)$$

Since we don't need the matrix entries any further, we only store the current value in the variable $Reach$. In addition, we store the size of the matching that used in the definition of $Reach^{r-end}(RDist)$. If $Reach^{r-end}(RDist)$ is false, then we use the size of the last entry $Reach^{r-end}(RDist')$ with $RDist' < RDist$ and $Reach^{r-end}(RDist') = true$. Technically, this is achieved by an array $M^{r-end}(RDist)$ with

$$M^{r-end}(RDist) = \max\left\{ |M| \;\middle|\; \begin{array}{l} M \subseteq [i..i'] \times [j..j'] \text{ is connected matching} \\ \text{with } (i',j') \in M \text{ and for all } (r',s') \text{ in} \\ M \backslash \{(i,i'),(j,j')\} \text{ we have } (r,s) \preceq (r',s') \end{array} \right\} \qquad (2)$$

6 Recursion Equations

The auxiliary matrices and arrays can be easily calculated via the following recursion equations. For $M_{\prec}^{loop}(r,s)$ we have

$$M_{\prec}^{loop}(r,s) = \tag{3}$$

$$\begin{cases} M^{bb}(r,s) + M_{\prec}^{loop}(r'+1, s'+1) & \text{if match}(r \overset{P_1}{\rule[0.5ex]{1.5em}{0.15em}} r', s \overset{P_2}{\rule[0.5ex]{1.5em}{0.15em}} s') \\ M^{rb}(r,s) + M_{\prec}^{loop}(r+1, s+1) & \text{else if } \text{rbd}_{P_1}(r) \wedge \text{rbd}_{P_2}(s) \wedge \text{match}(r,s) \\ 1 + M_{\prec}^{loop}(r+1, s+1) & \text{else if match}(r,s) \\ 0 & \text{otherwise} \end{cases}$$

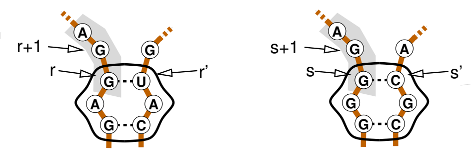

Figure 4. Extension to next loop.

Note that if r and s are the left ends of the bonds $r \overset{P_1}{\rule[0.5ex]{1.5em}{0.15em}} r' \wedge s \overset{P_2}{\rule[0.5ex]{1.5em}{0.15em}} s'$, but the bonds are not matchable, then this case is covered by the third case. Here, $r+1$ and $s+1$ are *not* in the same loop as r, s. Therefore, we consider the case where the maximal matching extends to the next loop via the left ends of two bonds. This case is depicted in Figure 4. r and s do match, whereas the bonding partners r' and s' do not match. The currently considered loop is encircled. Since $r+1$ and $s+1$ in the contiguous loop do match, we know that we can calculate $M_{\prec}^{loop}(r,s)$ recursively by calculating $M_{\prec}^{loop}(r+1, s+1)$.

The next step is to define the auxiliary arrays $Reach^{r\text{-}end}(RDist)$ and $M^{r\text{-}end}(RDist)$ for a given loop. $RDist$ is the distance to the right end of the closing bond. Consider the case where we want to match two loops associated with the bonds $i \overset{P_1}{\rule[0.5ex]{1.5em}{0.15em}} i'$ and $j \overset{P_2}{\rule[0.5ex]{1.5em}{0.15em}} j'$. Let len be the minimum of the two loop lengths, and $0 \leq RDist < len$. Then

$$Reach^{r\text{-}end}(0) = \begin{cases} true & \text{if match}(i', j') \\ false & \text{otherwise} \end{cases} \quad \text{and } M^{r\text{-}end}(0) = \begin{cases} 1 & \text{if match}(i', j') \\ 0 & \text{otherwise} \end{cases}$$

For $1 \leq RDist \leq len_{min}$, let $r = \text{left}_{R_1}^{RDist}(i')$ and $s = \text{left}_{R_2}^{RDist}(j')$ be the two positions with distance $RDist$ to the right end of the considered loops. Then we obtain $Reach^{r\text{-}end}(RDist) \quad = Reach^{r\text{-}end}(RDist-1) \wedge \text{match}(r,s)$

$$M^{r\text{-}end}(RDist) = \begin{cases} M_{\prec}^{loop}(r,s) & \text{if } Reach^{r\text{-}end}(RDist) \\ M^{r\text{-}end}(RDist-1) & \text{otherwise.} \end{cases}$$

The matrix $M^{rb}(i \overset{P_1}{\rule[0.5ex]{1.5em}{0.15em}} i', j \overset{P_2}{\rule[0.5ex]{1.5em}{0.15em}} j')$ then is simply $\max\limits_{0 \leq Rdist < len_{min}} \left\{ M^{r\text{-}end}(RDist) \right\}$.

For the M^{bb} matrix, there are two different cases as shown in Figure 5. In the first case a.), the extensions from the initial matching (i, i') to the right, and the extension from (j, j') to the left do not overlap, whereas they do overlap in the second case b). For the second case, we do not know exactly how to match the overlapping part. Hence, we have to consider all possible *cuts* in the smaller loop,

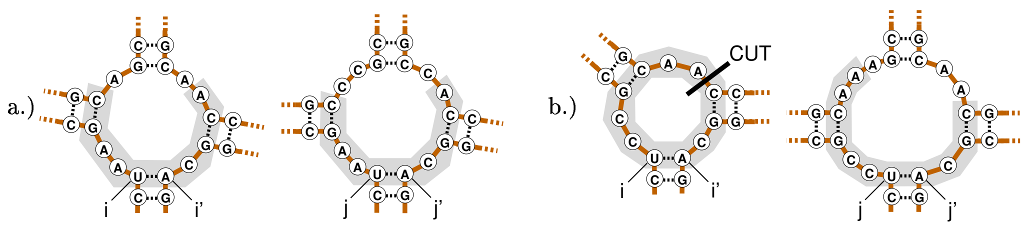

Figure 5. The two possible cases for $M^{rb}(i \overset{P_1}{\rule{2em}{2pt}} i', j \overset{P_2}{\rule{2em}{2pt}} j')$

marking the corresponding ends of the extensions from the left ends and from the right ends of the loop. The extensions from the right ends are already calculated in the M^{r_end} matrix. Only for the definition of the recursion equation, we define $M^{l\_end}(LDist)$ and $Reach^{l\_end}(LDist)$ analogously to equations (2) and (1), respectively. For the implementation, we need to store only the current values M^{l_end} and $Reach^{l\_end}$.

Now let $len_{i,i'}$ (resp $len_{j,j'}$) be $|\text{loop}(i \overset{P_1}{\rule{2em}{2pt}} i')|$ (resp. $|\text{loop}(i \overset{P_1}{\rule{2em}{2pt}} i')|$), and let $len_{min} = \min\{len_{i,i'}, len_{j,j'}\}$. Then we have

$$M^{bb}(i \overset{P_1}{\rule{2em}{2pt}} i', j \overset{P_2}{\rule{2em}{2pt}} j') = \max_{\substack{0 \leq LDist < Len_{Min} \\ \text{with right}_{P_1}^{LDist}(i) \text{ is not} \\ \text{a left end of a bond}}} \left\{ M^{l\_end}(LDist) + M^{r\_end}(RDist) \right\} \tag{4}$$

where $RDist = len_{i,i'} - LDist$ if $len_{min} = len_{i,i'}$, and $len_{i,i'} + (len_{i,i'} - len_{j,j'}) - LDist$ otherwise. The condition *right*$_{P_1}^{LDist}(i)$ *is not a left end of a bond* guarantees that we do not cut in the middle of a bond, which is excluded since we are considering bond-preserving matchings only. The term $(len_{i,i'} - len_{j,j'})$ in the second part of the definition of $RDist$ is to compensate for the longer length of the first loop[1].

Finally, we consider the $M(r,s)$ entries. Let r and s again be two bases of the loops defined $i \overset{P_1}{\rule{2em}{2pt}} i'$ and $j \overset{P_2}{\rule{2em}{2pt}} j'$ with distance $RDist$ to the right loop ends i' and j', respectively. The values of $M(r,s)$ and $Mloop(r,s)$ are equal for all entries $M(r,s) \neq 0$. $M(r,s)$ is zero if there is some $(r',s') \prec (r,s)$ that is matchable. This leads to the following equation: $M(r,s) = 0$ if $\neg\text{match}(r,s)$ or $\text{match}(\text{left}_{R_1}(r), \text{left}_{R_2}(s))$ or $Reach^{r\_end}(RDist)$, and $M_{\prec}^{loop}(r,s)$ otherwise.

7 Pseudo-Code

The main procedure consists of two for-loops, each calling a base-pair from the first and second RNA, and performs the pattern search from inner to outer loops. It calls the procedure START-LOOP-WALKING which initiates the calculation of all matrices except $M^{bb}(i \overset{P_1}{\rule{2em}{2pt}} i', j \overset{P_2}{\rule{2em}{2pt}} j')$ for two bonds $i \overset{P_1}{\rule{2em}{2pt}} i'$ and

[1] In the case that $i \overset{P_1}{\rule{2em}{2pt}} i'$ is the smaller loop, then overlapping of the left and right match extensions is already excluded by definition, and we do not need to compensate for it

```
 1: procedure START-LOOP-WALKING($i, i', j, j'$)
 2:     $reach$ = INIT-LOOP-MATRICES($i', j', i', j'$)
 3:     $(loop\_size, loop\_dist)$ := LOOP-WALKING($i', j', i, j, i', j', reach, true$)
 4:     $k := i'$
 5:     while $k > i + 1$ do
 6:         $k := \text{left}_{R_1}(k)$
 7:         INIT-LOOP-MATRICES($k, j', i', j'$)
 8:         LOOP-WALKING($k, j', i, j, i', j', false, false$)
 9:     end while
10:     $l := j'$
11:     while $l > j + 1$ do
12:         $l := \text{left}_{R_2}(l)$
13:         INIT-LOOP-MATRICES($i', l, i', j'$)
14:         LOOP-WALKING($i', l, i, j, i', j', false, false$)
15:     end while
16:     return $(loop\_size, loop\_dist)$
17: end procedure
```

Figure 6. Starting points of loop walking

$j \overset{P_2}{\boldsymbol{-\!\!-}} j'$, assuming that all matrix entries for loops above are already calculated. In addition, it calculates the loop length of the smaller loop and the distance of the two loop lengths (which is done in sub-procedure CALC-REMAIN-LOOP-LEN).

The real calculation of these matrices is done in the sub-procedure LOOP-WALKING, which traverses the loop from right to left (via the application of $\text{left}_{\cdot}(\cdot)$ function). The function LOOP-WALKING has two modes concerning whether we started the loop-traversal with both right ends i, i' or not. In the first mode (initiated in line of START-LOOP-WALKING), we calculate also the array M^{r_end}, and move the $M(r, s)$ down to (i', j') for all (r, s) where $Reach^{r\_end}$ is true. This part is done by the subprocedure LOOP-REACH. In the second mode, when LOOP-WALKING is called with only one right end (lines 8 and 14 of START-LOOP-WALKING), then we know the right ends cannot be in any matching considered there. Hence, we may not calculate the M^{r_end} array.

The subprocedure MLOOP-RECURSION is just an implementation of recursion Equation (3) for $M_{\prec}^{loop}$. The sub-procedure INIT-LOOP-MATRICES just initializes the matrices for the starting points. In most case, the initial values are 0 (since we cannot have a match if we do not start with the right-ends due to the structure condition). The only exception is if we start with both right ends, and these rights ends do match. In this case, we initialize the corresponding matrix entries with 1. The sub-procedure INIT-LOOP-MATRICES is listed in the appendix.

The next step is to calculate $M^{bb}(i \overset{P_1}{\boldsymbol{-\!\!-}} i', j \overset{P_2}{\boldsymbol{-\!\!-}} j')$, which is done by the procedure LOOP-MATCHING. LOOP-MATCHING is called *after* START-LOOP-WALKING is finished. In principle, this is just an implementation of the recursion equation (4). Since we do not want want to maintain another array $M^{l\_end}(LDist)$, we store only value for the current $LDist$ in the variable M^{l_end}. The procedure maintains three neighbouring cells (r^l, s^l), (r, s) and (r^r, s^r). (r^l, s^l) correspond to $LDist - 1$, and (r, s) to $LDist$. The cut will be between (r, s) and (r^r, s^r). The sub-procedure MLEND-RECURSION is in principle only an

```
procedure LOOP-WALKING($r, s, i, j, i', j', reach, right\_ends$)
    $RDist = 0$
    while $r > i \wedge s > j$ do
        $r' := r$;   $s' := s$;   $r := \text{left}_{R_1}(r')$;   $s := \text{left}_{R_2}(s')$;   $RDist = RDist + 1$
        if BASE-MATCH($r, s$) $\vee$ BOND-MATCH($r^r, r, s^r, s$) then
            MLOOP-RECURSION($r^r, r, s^r, s'$)
            $M(r, s) := M_{\prec}^{loop}(r, s)$;   $M(r', s') := 0$
            If $right\_ends$ then LOOP-REACH($r, s, i, j, i', j', reach, RDist$)
        else
            $M_{\prec}^{loop}(r, s) := 0$;   $M(r, s) := 0$;   $reach := false$
            If $right\_ends$ then $M^{r}\text{-}^{end}(RDist) := M^{r}\text{-}^{end}(RDist - 1)$
        end if
    end while
    If $right\_ends$ then return CALC-REMAIN-LOOP-LEN($r, s, i, j, RDist$)
end procedure
```

Figure 7. The procedure loop walking is going from one base to the next

implementation of the recursion equation for $M^{l}\text{-}^{end}$ under the condition that that $Reach^{l}\text{-}^{end}$ is true. As it can be seen from the definition of $M^{r}\text{-}^{end}$ in Equation (2), the recursion equation under this condition is in principle analogous to the recursion equation for $M_{\prec}^{loop}$ given in Equation (3).

```
procedure LOOP-MATCHING($i, i', j, j', i\_i'\_lens, lens\_dist$)
    $LDist := 0$
    if BOND-MATCH($i, i', j, j'$) then
        $M^{l}\text{-}^{end} := 0$;   $Reach^{l}\text{-}^{end} := true$
        $r^r := i$;   $r := i$;   $r^l := i$;   $s^r := j$;   $s := j$;   $s^l := j$
        while $r^r < i' \wedge s^r < j' \wedge Reach^{l}\text{-}^{end} := true$ do
            $r^l := r$; $r := r^r$; $r^r := \text{right}_{R_1}(r^r)$; $s^l := s$; $s := s^r$; $s^r := \text{right}_{R_2}(s^r)$;
            if BASE-MATCH($r, s$) $\vee$ BOND-MATCH($r^r, r, s^r, s$) then
                $M^{l}\text{-}^{end}$ = MLEND-RECURSION($r^l, r, r^r, s^l, s, s^r, M^{l}\text{-}^{end}$)
            else $Reach^{l}\text{-}^{end} := false$ endif
            if $Reach^{l}\text{-}^{end} \wedge \neg$BOND-MATCH($r^l, r, s^l, s$) then
                FILL-MBB($i, i', j, j', M^{l}\text{-}^{end}, LDist, i\_i'\_len, lens\_dist$)
            end if
            $LDist := LDist + 1$
        end while
    else $M^{bb}(i, j) := 0$ end if
end procedure
```

Figure 8. Calculation of M^{bb}

The maximally extended matchings are finally calculated from the $M(r, s)$ matrix by an usual traceback. The space complexity of the algorithm is $O(nm)$. The time complexity is $O(nm)$ for the following reason. Every pair (r, s) with $1 \leq r \leq |S_1|$ and $1 \leq s \leq |S_2|$ is considered at most twice in START-LOOP-WALKING and LOOP-WALKING, with an $O(1)$ complexity for calculating the corresponding matrix entries. Similarly, every pair (r, s) is considered at most twice in LOOP-WALKING. Since there are $O(nm)$ many pairs (r, s), we get a total complexity of $O(nm)$.

8 Conclusion

We have presented a fast dynamic programming approach in time $O(nm)$ and space $O(nm)$ for detecting common sequence/structure patterns between two RNAs given by their sequence and secondary structures. These patterns are derived from exact matchings and can be used for local alignments ([1]). The most promising advantage is clearly to investigate large RNAs of several thousand bases in reasonable time. Here, one can think of detecting local sequence/structure regions of several RNAs sharing the same biological function.

References

1. Rolf Backofen and Sebastian Will. Local sequence-structure motifs in RNA. *Journal of Bioinformatics and Computational Biology (JBCB)*, 2004. accepted for publication.
2. V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between rna strings. In *Proc. 6th Symp. Combinatorical Pattern Matching*, pages –16, 1995.
3. David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms & Applications*, 3(3):1–27, 1999.
4. P. Gendron, D. Gautheret, and F. Major. Structural ribonucleic acid motifs identification and classification. In *High Performance Computing Systems and Applications*. Kluwer Academic Press, 1998.
5. Gramm, Guo, and Niedermeier. Pattern matching for arc-annotated sequences. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 22, 2002.
6. I. L. Hofacker, B. Priwitzer, and P. F. Stadler. Prediction of locally stable RNA secondary structures for genome-wide surveys. *Bioinformatics*, 20(2):186–190, 2004.
7. Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in rna secondary structures. In *Proceedings of Computational Systems Bioinformatics (CSB 2003)*, 2003.
8. Tao Jiang, Guo-Hui Lin, Bin Ma, and Kaizhong Zhang. The longest common subsequence problem for arc-annotated sequences. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM2000)*, 2000.
9. Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–88, 2002.
10. B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–4, 1998.
11. J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, January 1976.
12. Juris Viksna and David Gilbert. Pattern matching and pattern discovery algorithms for protein topologies. In O. Gascuel and B. M. E. Moret, editors, *Proceedings of the First International Workshop on Algorithms in Bioinformatics (WABI 2001)*, number 2149, pages 98–111, 2001.
13. Jason Tsong-Li Wang, Bruce A. Shapiro, Dennis Shasha, Kaizhong Zhang, and Kathleen 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.