DIMACS Technical Report 96-30
July 1996

Computing similarity between RNA strings [1]

by

V. Bafna,[2]
DIMACS Center
P.O. Box 1179
Piscataway, NJ 08855
bafna@dimacs.rutgers.edu

S. Muthukrishnan
Dept. of Computer Science
Univ. of Warwick
Coventry, CV47AL, UK
muthu@dcs.warwick.ac.uk

R. Ravi
Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15217
ravi+@andrew.cmu.edu

[1] Work done while the authors were postdoctoral fellows at DIMACS and supported under NSF contract STC-8809648. The second author's research was partly supported by ALCOM IT.
[2] Postdoctoral fellow

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

Ribonucleic acid (RNA) strings are strings over the four-letter alphabet $\{A, C, G, U\}$ with a secondary structure of base-pairing between $A-U$ and $C-G$ pairs in the string. Edges are drawn between two bases that are paired in the secondary structure and these edges have traditionally been assumed to be noncrossing. The noncrossing base-pairing naturally leads to a tree-like representation of the secondary structure of RNA strings.

In this paper, we address several notions of similarity between two RNA strings that take into account both the primary sequence and secondary base-pairing structure of the strings. We present efficient algorithms for exact matching and approximate matching between two RNA strings. We define a notion of alignment between two RNA strings and devise algorithms based on dynamic programming. We then present a method for optimally aligning a given RNA sequence with unknown secondary structure to one with known sequence and structure, thus attacking the structure prediction problem in the case when the structure of a closely related sequence is known. The techniques employed to prove our results include reductions to well-known string matching problems, allowing wild cards and ranges, and speeding up dynamic programming by using the tree structures implicit in the secondary structure of RNA strings.

1 Introduction

A variety of string matching problems are motivated by the analysis of DNA or protein sequences. An example is the problem of computing the similarity between two sequences such as the edit distance to transform one into another using insertions, deletions and substitutions of characters [13, 20, 21]. When comparing RNA sequences, usually much more is known about the secondary structure of base-pairing between nucleotides in the sequence. A bonded pair of bases is usually represented as an *edge* between the two complementary bases involved in the bond. Traditional models of RNA secondary structure[27] assume that every base participates in at most one such pair, and that the edges representing the paired bases are noncrossing along the length of the string. The secondary structure can be represented by a nesting tree whose nodes correspond to edges of the pairing and parenthood in the tree represents the immediate enclosure relation between the two edges (see fig. 1).

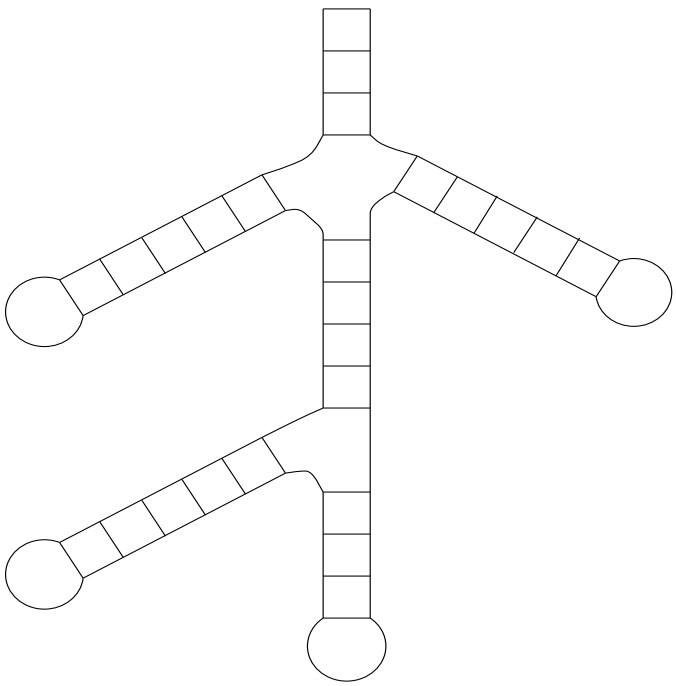

Figure 1: Secondary structure of an RNA sequence

Consequently, much of the work on comparing the secondary structures of two RNA strings have been modeled as problems of comparing two trees [7, 22, 25, 26]. In this paper, we study several problems in computing the similarity between two RNA strings that take into consideration *both* the primary sequence and secondary base-pairing information provided with the strings. We also investigate the problem of inferring the secondary structure of an RNA sequence when a closely related RNA string is given. We reserve the term RNA string when the sequence as well as the structure is given and use RNA sequence to denote only the primary sequence.

	Problem	Running time
String Matching	Symmetric Exact Matching (SEM)	$O(n+m)$
	Containment Exact Matching (CEM)	$O(n \text{ polylog } m)$
	Symmetric K-mismatches problem (SKM)	$O(nm^{2/3} \text{ polylog } m)$
	Containment K-mismatches problem (CKM)	$O(n\sqrt{m} \text{ polylog } m)$
Alignment	Global similarity or RNA edit distance	$O(n^2m^2)$
	Longest Common RNA Substring (RLCS)	$O(n^2m^2)$
	Shortest Common RNA Superstring (RSCS)	$O(n^2m^2(n^2+m^2))$
Structure	Inferring structure through alignment	$O(n^2m^2+nm^3)$

Table 1: Summary of results.

Standard notions of similarity between two sequences have been formulated as problems of exact and approximate string matching, finding a longest common subsequence (LCS) or a shortest common supersequence (SCS) of the sequences, and computing optimal alignments under general scoring functions. We formulate the corresponding versions of these problems between two RNA strings and present efficient algorithms for computing them. In the context of RNA strings, we would like to match symbols as well as edges in the two strings. An edge in one of the strings is said to be matched if the two symbols in the other string that are aligned to its endpoints are connected by an edge in the second string. Two variants of exact and approximate matching problems arise, depending on whether edges in both the pattern and text strings or only the edges in the pattern are required to be matched: we call these variants the *symmetric* and *containment* variants respectively. We observe that the action-at-a-distance effect of aligning edges significantly increases the complexity of RNA string similarity algorithms. Furthermore, problems like LCS and SCS that are computationally identical for sequences turn out to be quite different in the context of RNA strings.

Finally, we solve the problem of inferring the secondary structure of an RNA sequence when a closely related RNA string is given by aligning the sequence with the given RNA string. The alignment not only maximizes the common characters as in traditional sequence alignment but also favors the alignment of the endpoints of an edge in the related string with complementary base-pairs in the input string.

Results

Table 1 describes the list of problems addressed in this paper and the time complexity of the algorithms we propose for solving them. In the table, n and m represent the sizes of the two input strings.

String matching. The first four entries of the table are analogous to exact and approximate matching problems in standard strings. In such problems, a pattern RNA string (of length m) and a typically longer text RNA string (of length n) are given, and the task is to compute

the positions in the text where the pattern occurs with at most k (≥ 0) mismatches. We obtain our bounds by reducing these RNA string matching problems to well-known string matching problems allowing wild cards and ranges.

While character mismatches clearly account as mismatches, the notion of an edge mismatch can be defined in many ways. Consider the alignment of the pattern beginning at position i in the text. We shall insist that an edge in the pattern string must always be matched by an edge in the text or it creates an edge mismatch. An edge in the text string which has exactly one endpoint within the range formed by the projection of the pattern in the text cannot be considered an edge mismatch, since the other endpoint is out of the span of the pattern in the text. However, two variants arise from the consideration of whether an edge in the text within the span of the pattern must be deemed to match with an edge in the pattern. The *Symmetric* variant counts a text edge within the span of the pattern that is not matched as a mismatch, while the *Containment* version does not count this as a mismatch. (Formal definition of these problems are in Section 2).

Alignment. The global similarity between two RNA strings is defined as a weighted sum of sequence similarity and structural similarity. Our algorithm for computing global similarity can be used to compute an edit distance between two RNA strings that is a weighted sum of sequence edit cost and edge-mismatch costs.

The next two entries in Table 1 are analogous to the LCS and SCS problems.

Informally, an RNA-subsequence of an RNA-string is a subsequence of the string and has an edge only if the corresponding base-pairs have an edge in the RNA-string. A longest common RNA subsequence (RLCS) of two RNA strings is an RNA-subsequence of the two given strings with the maximum number of characters plus edges. This definition, as well as our algorithm, can be extended to maximize an arbitrary weighted sum of the number of characters and the number of edges. The shortest common RNA supersequence (RSCS) is an RNA string of which both the given strings are RNA subsequences such that the total number of characters plus edges in the supersequence is minimum. Again, the extension to weighting number of edges and number of characters differently is straightforward. Somewhat surprisingly, RLCS and RSCS seem to be computationally very different problems. We can reduce RLCS to computing an optimal alignment by defining the scoring function appropriately. On the other hand our algorithm for computing the RSCS has running time that is two orders of magnitude more.

Structure. The problem of inferring structure through alignment, takes as input an RNA sequence and an RNA string and computes an alignment of the sequence with the string to maximize a weighted sum of the sequence similarity and the number of edges in the RNA string whose endpoints are aligned with complementary base-pairs in the RNA sequence. This alignment can be used as a measure of commonality between the RNA string and the unknown RNA sequence. As more and more RNA secondary structures become available, our formulation of the prediction problem appears increasingly relevant.

Related work

Comparison methods. First we review work on comparison methods developed to estimate distances between RNA secondary structures. Since secondary structures can be represented as trees, there are several papers [7, 22, 25, 26] addressing comparisons of trees. Tree edits are discussed and efficient algorithms are derived in [22, 25, 26] while a new notion of tree alignment is proposed and algorithms developed in [7]. Even though these comparison methods compute distances only between secondary structures, the worst-case running time of estimating this distance (tree edits or alignments) is quadric; the running time of our alignment algorithm that takes in to account both sequence and structure compares favorably with this estimate.

Tree based comparison methods do not appear to generalize when comparing both the secondary structure as well as the underlying sequence of two RNA strings. The difficulty seems to be in extending the tree model for RNA strings to include characters. However, our algorithms use a combination of sequence edit computation and a tree-based computation.

Prediction methods. Prediction of RNA secondary structure of a single RNA molecule from its sequence has been widely studied in the past [3, 4, 10, 14, 23, 24, 28, 29]. Most of this work use adaptations of dynamic programming to solve variants of the problem; For instance, one could take into account different energy assignments for the different secondary structure primitives such as stacked pairs, hairpins, multiloops, interior loops and bulges[1], and compute a structure that minimizes the total energy. Sankoff [17] considers prediction of secondary structure common to two RNA strings also taking into account alignment of the sequences. The difference from our approach is that he does not assume that the secondary structure of either of the sequences is given, but instead computes an alignment and a most likely folding that is common to both sequences. Sankoff's algorithm more carefully models the energy functions for different kinds of loops in the structure (such as stacked pairs, multiloops etc.); the running time of his algorithm for two sequences is two orders of magnitude higher. In contrast to the above work, our version of the prediction problem assumes that more information is available. In particular, we assume that we are given both the sequence and the secondary structure of an RNA string that is closely related to one whose structure is to be predicted.

Another related line of work is [6, 15, 16] that uses stochastic context-free grammars (with probabilities attached to production rules) to model a family of related RNA strings. These papers also describe procedures for building such grammars given a set of unaligned RNA sequences with no secondary structure known for any member in the set. In this way, this problem is more closely related to that considered by Sankoff in [17] and described in the previous paragraph.

Other related work appears in [11, 18, 19, 8]. The approach in the first paper [11] addresses a single RNA sequence. Multiple secondary structures are computed from the given sequence and a consensus structure is output. The second article [18] considers the problem of aligning several RNA strings, each with a given secondary structure. The paper proposes

[1]For a formal definition of these terms, refer to [29].

a syntactic parenthetical representation of the RNA secondary structure that is interleaved with the primary RNA sequence. The paper then uses conventional multiple-sequence alignment methods on these modified sequences to infer a similar substring. However, the author does not rigorously specify what function of these modified sequences is being optimized to derive the final alignment and common substring; Our algorithms for exact and approximate matching of two RNA strings superficially resembles this method in that an edge between two positions in a string is encoded by the offset to the other endpoint (rather than parentheses as in [18]), and traditional string matching techniques are then employed. The third article [19] uses edit distance data between several trees representing RNA secondary structures to build a phylogenetic tree illustrating the relationship between these structures. Finally, [8] considers query primitives for data that is tree-structured that are less restricted than exact tree-pattern matching and analyzes the complexity of implementing such primitives. Thus, this work is more closely related to work on distances between trees.

In the next section, we describe our results on RNA string matching; We then turn to alignment of RNA strings and its variants in Section 3. In Section 4, we describe our algorithm for structure prediction given a related RNA string. We conclude with some open questions.

2 RNA string matching

In this section, we explore string matching problems in the context of RNA strings. We formally define the variants of string matching problems that arise when we deal with RNA strings.

2.1 Problem definitions

The first problem we consider is the basic problem of exact matching.

Symmetric Exact Matching (SEM). Given a text RNA string t and a pattern RNA string p of length n and m respectively, determine all those positions where p occurs in t, that is, all locations i in t where,

1. *Strings match*, that is, $t[i] \cdots t[i+m-1] = p[1] \cdots p[m]$, and
2. *Secondary structures are identical*, that is, for any bonding pair $(i+j-1, i+k-1)$ in t, $0 \leq j < k \leq m-1$, (j,k) is a bonding pair in p as well and vice-versa. Note that the existence of some bonding pair $(i+j, i+k)$ in t, where $0 \leq j < k$ and $k > m-1$ does not affect p occurring at i in t. Similarly for bonding pairs $(i+j, i+k)$ in t, where $0 \leq k \leq m-1$ and $j < 0$.

Containment Exact Matching (CEM). It is defined just as SEM except that the secondary structure of p and that of the substring of t of length m beginning at i need not be identical. It is only required that the secondary structure of p be contained in the latter. Formally, p occurs in t at i if strings match as before and

2. For any bonding pair (j,k) in p, $1 \leq j < k \leq m$, $(i+j-1, i+k-1)$ in t is a bonding pair as well, but not necessarily vice-versa.

The other basic problem we consider is the k-mismatches problem for the RNA strings defined as below.

Symmetric K-mismatches problem (SKM). Given a text RNA string t and a pattern RNA string p of length n and m respectively, and a parameter K, determine all those positions where p occurs in t with at most K mismatches. We say that p occurs at i in t with at most K mismatches if there exist integers K_1 and K_2 such that $K_1 + K_2 \leq K$ and in addition,

1. *There are K_1 string mismatches.* That is, $t[i] \cdots t[i+m-1]$ and $p[1] \cdots p[m]$ differ in K_1 positions.

2. *There are K_2 secondary structures mismatches.* That is, there are K_2 pairs of text and pattern positions $i+j-1, i+k-1$, $0 \leq j < k \leq m-1$ and j,k where precisely one of $(i+j-1, i+k-1)$ or (j,k) is a bonding pair in the corresponding string. Note again that the existence of some bonding pair $(i+j, i+k)$ in t, where $0 \leq j < k$ and $k > m-1$ does not affect the number of mismatches between the secondary structure of p and that of $t[i] \cdots t[i+m-1]$. Similarly for bonding pairs $(i+j, i+k)$ in t, where $0 \leq k \leq m-1$ and $j < 0$.

Containment K-mismatches problem (CKM). The CKM problem is defined the same way as SKM except for the mismatches in the secondary structure. Here we only consider the number of bonding pairs in p which mismatch, that is, those that do not fall on bonding pairs in t; the bonding pairs in t that do not fall on a bonding pair in p are not counted as mismatches.

2.2 Exact Matching with RNA strings

In this section, we sketch our results for SEM and CEM problems.

Theorem 2.1 *The CEM problem can be solved in $O(n \operatorname{polylog} m)$ time.*

Proof: Our algorithm works as follows. First we perform standard string matching (using, say [9]) to find all locations where p does not occur in t since the strings mismatch. This takes $O(n+m)$ time.

Next we look for mismatches in the secondary structure. We construct an instance of string matching with wild cards to solve this problem. Note that the RNA strings are strings drawn from $\{A, C, G, U\}$; the strings we generate for string matching with wild cards will contain symbols from $\{A, C, G, U\}$ as well as integers. We generate a text string t' from t by replacing $t[i]$ by the integer $k-i$ if (i,k) or (k,i) is a bonding pair in t. Similarly, we

generate p' from p by replacing each position i by $k-i$ if (i,k) or (k,i) is a bonding pair in p, and by the wild card ϕ otherwise.

Every position i such that p occurs at i in t and p' occurs at i in t' corresponds to a position where the RNA-string p is contained in the RNA-string t. The algorithm takes time $O(n \text{ polylog } m)$ using the bound in [5] for exact string matching with wildcards. □

Theorem 2.2 *The SEM problem can be solved in $O(n+m)$ time.*

Proof: We prove that a simple modification of the Knuth-Morris-Pratt algorithm [9] suffices to solve the SEM problem. Let the match operation between two strings as defined for the SEM problem be denoted by $\cong$.

Lemma 2.3 *The $\cong$ relation is transitive, that is, if $t[i]\cdots t[i+k-1]$ matches $p[1]\cdots p[k]$, and some prefix $p[1]\cdots p[j]$, $j<k$, matches the suffix of $t[i]\cdots t[i+k-1]$, then the string $p[1]\cdots p[j]$, matches the suffix of $p[1]\cdots p[k]$.*

Proof: Follows from the definition of the $\cong$ relation. □

Now we modify the comparisons in the KMP algorithm to perform SEM. Note that in standard KMP, comparing two locations is merely testing for character equality. The following subroutines for Compare($p[i]$,$t[j]$) for $j \geq i$, and Compare($p[i]$,$p[j]$) for $i \leq j$ implement our $\cong$ relation.

Compare($p[i]$,$t[j]$)
  if $p[i] \neq t[j]$ return *unequal*
  if $p[i] = t[j]$, $(i-k,i)$ is a bonding pair in t, $0 \leq k \leq i$ but $(j-k,j)$ is not a bonding pair in p then return *unequal*
  if $p[i] = t[j]$, $(j-k,j)$ is a bonding pair in p, $0 \leq k \leq i$ but $(i-k,i)$ is not a bonding pair in t then return *unequal*
  else return *equal*
end

The Compare($p[i]$,$p[j]$) for $i \leq j$ operation is implemented the same way. That completes the description of the algorithm.

The proof of correctness relies on the valid implementation of the failure function in the KMP algorithm by the new Compare operations. However, this is straightforward to verify, given Lemma 2.3 and that in the preprocessing of the pattern in the algorithm using the $\cong$ operation, the following is maintained: The largest prefix of $p[1]\cdots p[i+1]$ that matches its suffix under $\cong$ is the largest prefix $p[1]\cdots p[k]$ of $p[1]\cdots p[i]$ that matches $p[i-k+1]\cdots p[i]$ and also answers Compare($p[i+1]$,$p[k+1]$) equal. It follows that the entire algorithm works in $O(n+m)$ time. □

2.3 K-mismatches with RNA strings

We now describe algorithms for RNA string matching with mismatches. Our algorithms use known results from generalized string matching.

String matching with mismatches: Given a pattern string p of length m and a text string t of length n, for each position $i, 1 \leq i \leq n-m$, find the number of mismatches when p is positioned at i in t.

Theorem 2.4 *[1] The string matching problem with mismatches can be solved in* $O(n\sqrt{m}\text{polylog}m)$ *time.*

String matching with sets of ranges: Given a pattern string p of length m and a text string t of length n where t contains symbols from an ordered alphabet Σ and each position of p matches (a fixed number of) sets of ranges of the symbols from Σ, for each position $i, 1 \leq i \leq n-m$, find the number of mismatches when p is positioned at i in t.
An example. Let $p = a\langle[a-c]+[f-g]+[z]\rangle c$ and $t = ababcda$. The second position in p from the left matches symbols a, b, c, f, g, z. Therefore, the number of mismatches between p and t at the leftmost position is 1 and that at the 5th position from the left is 3.

Theorem 2.5 *[12] The string matching problem with sets of ranges can be solved in* $O(nm^{\frac{2}{3}}\text{polylog}m)$ *time.*

Theorem 2.6 *The CKM problem can be solved in* $O(n\sqrt{m}\,\text{polylog}\,m)$ *time.*

Proof: As usual we solve the version of the problem in which for each text location we return the number of mismatches (in the string and the secondary structure) between p and t. The problem of counting character mismatches can be solved in $O(n\sqrt{m}\,\text{polylog}\,m)$ time using Theorem 2.4. Next we compute the number of mismatches in the secondary structure. Recall the reduction in the proof of Theorem 2.1. We generate t' and p' (with wild cards) as described there. Observe that the number of mismatches of p' at a location i in t' is *exactly twice* the number of edge mismatches of p at i in t. Thus, our problem is reduced to solving the problem of counting the mismatches between p' and t'. This problem in turn can be solved in $O(n\sqrt{m}\,\text{polylog}\,m)$ time as above. Finally, we can combine the number of mismatches in the two parts above in linear time and detect all locations in t where the total number of mismatches is at most K. □

Theorem 2.7 *The SKM problem can be solved in* $O(nm^{\frac{2}{3}}\,\text{polylog}\,m)$ *time.*

Proof: Suppose without loss of generality that $n \leq 2m$. We will again determine for each position the number of mismatches between p and t. Doing this considering p and t as sequences of characters can be done in $O(nm^{\frac{1}{2}}\,\text{polylog}\,m)$ time. Hence we only consider counting edge mismatches. We give a reduction from this problem to string matching with sets of ranges. We generate t' from t by replacing each position i by $k-i$ if (i,k) or (k,i) is a bonding pair in t, and by a special character character X_t if there is no bonding pair involving i. We generate p' from p by replacing each position i by $k-i$ if (i,k) or (k,i) is a bonding pair in p, and by

$$\langle[(m-i)\cdots(2m)]+[(-2m)\cdots(-i)]+X_t\rangle$$

if there is no bonding pair involving i.

Suppose for a given placement of the pattern on the text, the character in position i of the pattern is not involved in a base-pair while the corresponding character in the text is. Two cases arise. If the base-pair in t lies within the span of p, then this indicates an edge mismatch. On the other hand, if the other endpoint of the base-pair in t is outside the span of p, then there is no edge mismatch at position i in p. In the first case, the position in t' corresponding to position i in p' contains an integer in the range $\{-(i-1), \ldots, m-i-1\}$. This is outside the chosen range for $p'[i]$ and results in a mismatch. In the second case, $p'[i]$ matches with the corresponding character in t'. Thus, p' occurs at i in t' with $2k$ mismatches if and only if p occurs in t with k edge mismatches. By Theorem 2.5, the running time is $O(nm^{\frac{2}{3}} \text{ polylog } m)$. □

3 Computing Alignment for RNA strings

In this section, we look at sequence-alignment problems in the context of RNA strings. Specifically, we will consider variants of the edit-distance, longest-common-subsequence and shortest-common-supersequence problems.

Following Zuker [27, 28, 29], assume a model in which there are no *knots* in the secondary structure. In other words, the edges representing base-pairs in the secondary structure must be noncrossing along the length of the string. A secondary structure is denoted by the set S of all base-pairs which have formed bonds. For $(i,j) \in S$, h is accessible from (i,j) if $i < h < j$, and there is no pair $(k,l) \in S$, s.t. $i < k < h < l < j$. Define $(i,j) \in S$ as the *parent* of $(k,l) \in S$ if k, l are accessible from (i,j). Observe that each $(i,j) \in S$ has at most one parent due to noncrossing edges, implying a forest on the elements of S. The definitions of sibling, child and leaf follow naturally.

Let $s[1 \ldots n]$ and $t[1 \ldots m]$ be two RNA strings over the alphabet $\sum = \{A, C, G, U\}$ with structure S_1 and S_2 respectively. For technical reasons, let $s[0] = t[0] = \text{`}-\text{'}$. An *alignment* of $s$ and $t$ is defined by a $2 \times m'$ matrix $A$, in which the first row contains $s$ and the second contains $t$, possibly interspersed with spaces, and for all columns $j$, either $A[1,j] \neq \text{`}-\text{'}$ or $A[2,j] \neq \text{`}-\text{'}$. For $i \in \{1,2\}$, define

$$gap[i,j] = \begin{cases} j & \text{if } A[i,j] = \text{`}-\text{'} \\ |\{l < j \text{ s.t. } A[i,l] = \text{`}-\text{'}\}| & \text{otherwise} \end{cases}$$

Intuitively, if $A[i,j] \neq \text{`}-\text{'}$, then $gap[i,j]$ is the number of gaps that were inserted in the $ith$ string till position $j$ in alignment $A$. Following standard terminology, position $i$ in $A$ has a *match* if $A[1,i] = A[2,i] \neq \text{`}-\text{'}$, an *insertion* if $A[1,i] = \text{`}-\text{'}$, a *deletion* if $A[2,i] = \text{`}-\text{'}$ and a *mismatch* otherwise. Additionally, for RNA strings a bonding pair occurs at positions i, j if $(i - gap[1,i], j - gap[1,j]) \in S_1$ and $(i - gap[2,i], j - gap[2,j]) \in S_2$. Intuitively, we would like to compute an alignment which maximizes both symbol and base-pair matches.

Formally, for elements $u, v \in \sum \cup \{\text{`}-\text{'}\}$, define $\gamma(u,v)$ to be the similarity score associated with aligning u against v. For $1 \leq i < j \leq m$ and $1 \leq k < l \leq n$, let $\delta(i,j,k,l)$ be the similarity score associated with aligning base-pairs (i,j) with (k,l).

Definition 3.1 *The Global Alignment problem for RNA strings is defined as follows: Given RNA strings s and t, compute an alignment A (and the associated function gap) that maximizes*

$$\sum_{1\le i\le m'} \gamma(s[i-gap[1,i]], t[i-gap[2,i]]) \quad + \\ \sum_{1\le i<j\le m'} \delta(i-gap[1,i], j-gap[1,j], i-gap[2,i], j-gap[2,j])$$

Theorem 3.2 *Algorithm AlignRNA (fig. 2) computes the score of an optimum global alignment for Two RNA strings in $O(n^2m^2)$ time.*

The proof of correctness of the theorem follows from the observation that the last column in an alignment either participates in an edge match or not. If not, this column represents a character insertion, deletion or substitution. The algorithm chooses the one with maximum score among these four possible cases. Since we examine all pairs of intervals drawn from the two strings, the running time is as claimed. Standard techniques can be used to retrieve an alignment of maximum similarity using the same algorithm.

Note that edit-distance is the inverse problem of computing an alignment with a minimum number of insertions, deletions, mismatches and bonding pair mismatches. It follows that edit-distance can be computed in $O(n^2m^2)$ time.

Procedure *AlignRNA*
begin
for intervals (i_1, j_1), $1 \le i_1 < j_1 \le m$
and (i_2, j_2), $1 \le i_2 < j_2 \le n$
(* Assume that the intervals are examined in lexicographically
increasing order of widths *)

$$Align[i_1, j_1, i_2, j_2] = \max \begin{cases} Align[i_1, j_1 - 1, i_2, j_2] + \gamma(s[j_1], \text{`}-\text{'}) \\ Align[i_1, j_1, i_2, j_2 - 1] + \gamma(\text{`}-\text{'}, t[j_2]) \\ Align[i_1, j_1 - 1, i_2, j_2 - 1] + \gamma(s[j_1], t[j_2]) \end{cases}$$

if there exist $i_1 \le k_1 < j_1, i_2 \le k_2 < j_2$
s.t. $(k_1, j_1) \in S_1, (k_2, j_2) \in S_2$

$$Align[i_1, j_1, i_2, j_2] = \max \begin{cases} Align[i_1, j_1, i_2, j_2], \\ Align[i_1, k_1 - 1, i_2, k_2 - 1] + \\ Align[k_1 + 1, j_1 - 1, k_2 + 1, j_2 - 1] \\ +\delta(k_1, j_1, k_2, j_2) + \gamma(s[k_1], t[k_2]) + \gamma(s[j_1], t[j_2]) \end{cases}$$

end

Figure 2: Computing optimal alignment for RNA strings

3.1 LCS and SCS of RNA strings

For sequences, LCS and SCS can easily be deduced from an alignment in which no mismatches occur. The LCS is simply the sequence formed by concatenating matched non-space

symbols from such an alignment, while the SCS is the concatenation of all non-space symbols that appear in the alignment. It is also easy to see that for strings of length m and n, if l is the length of the longest common subsequence and s is the length of the shortest common supersequence, then $l = m + n - s$. Therefore for sequences, the solutions to both problems are identical.

The notion of a subsequence and supersequence can be extended naturally to RNA strings as follows:

Definition 3.3 *Let s and t be two RNA strings with structure S_1 and S_2 respectively. s is an RNA-supersequence of t if there exists an alignment A of s and t, such that*

- *for all i, $A[1,i] = s[i]$,*
- *for all j, either $A[2,j] = \text{`}-\text{'}$ or $A[1,j] = A[2,j]$ (i.e., s is a supersequence of t), and*
- *for all i,j $(i - gap[2,i], j - gap[2,j]) \in S_2$ implies $(i,j) \in S_1$.*

s is an RNA-subsequence of t if t is an RNA-supersequence of s.

Definition 3.4 *Let s be an RNA string with structure S. Define the 'length' of s, denoted by $len(s)$, to be $|s| + |S|$.*

From this, the definition of a shortest common RNA-supersequence (RSCS) and a longest common RNA-subsequence (RLCS) of two or more RNA sequences follows.

Theorem 3.5 *Let s,t be two RNA strings over $\sum$ with structure S_1 and S_2 respectively. RLCS of s,t can be computed in $O(n^2m^2)$ time.*

Proof: For elements $u, v \in \sum \cup \{\text{`}-\text{'}\}$, define

$$\gamma(u,v) = \begin{cases} 1 & u = v \\ 0 & u = \text{`}-\text{'} \text{ or } v = \text{`}-\text{'} \\ -\infty & \text{otherwise} \end{cases}$$

and

$$\delta(i,j,k,l) = \begin{cases} 1 & s[i] = t[k] \wedge s[j] = t[l] \\ & \wedge (i,j) \in S_1 \wedge (k,l) \in S_2 \\ 0 & s[i] = \text{`}-\text{'} \vee s[j] = \text{`}-\text{'} \vee \\ & t[k] = \text{`}-\text{'} \vee t[l] = \text{`}-\text{'} \\ -\infty & \text{otherwise} \end{cases}$$

Algorithm AlignRNA with these definitions of γ and δ computes the length of RLCS of two strings. The claim on running time follows from Theorem 3.2. □

Note that the alignment obtained in the LCS computation does not always yield a common RNA supersequence. In fact, it is possible that for the alignment obtained, there exist indices $1 \leq i < j < k < l \leq m + n$ such that $(i - gap[1,i], k - gap[1,k]) \in S_1$

and $(j - gap[2,j], l - gap[2,l]) \in S_2$. Clearly a common RNA supersequence must have both base-pairs but they cannot be interleaved because of our assumption of an unknotted structure.

Our algorithm for computing RSCS must therefore enforce this condition. For an RNA-string s with structure S, define a spanning-interval $[i,j]$ as a substring $s[i \ldots j]$, such that no pair $(k,l) \in S$ satisfies $i \leq k \leq j < l$ or $k < i \leq l \leq j$. We compute the RSCS for each pair of spanning interval $[i_1, j_1]$ and $[i_2, j_2]$ in the two strings. If j_1 or j_2 do not form bonds with another base-pair, then

$$RSCS[i_1,j_1,i_2,j_2] = \min \begin{cases} RSCS[i_1,j_1-1,i_2,j_2]+1 \\ RSCS[i_1,j_1,i_2,j_2-1]+1 \\ RSCS[i_1,j_1-1,i_2,j_2-1]+\gamma(s[j_1],t[j_2]) \end{cases}$$

where γ is defined as in RLCS. Otherwise, let $(k_1, j_1) \in S_1$ and $(k_2, j_2) \in S_2$. Then we have different cases depending whether (k_1, j_1) and (k_2, j_2) match or not in the RSCS of the two strings. (k_1, j_1) and (k_2, j_2) are matched only if $s[k_1] = t[k_2]$, and $s[j_1] = t[j_2]$. In that case,

$$\begin{aligned} RSCS[i_1,j_1,i_2,j_2] \;=\; & RSCS[i_1,k_1-1,i_2,k_2-1] + \\ & RSCS[k_2+1,j_1-1,k_2+1,j_2-1] + 3 \end{aligned}$$

If (k_1, j_1) and (k_2, j_2) are not matched to each other then either (k_1, j_1) is aligned against an interval $[x, y]$ in $[i_2, j_2]$ or (k_2, j_2) is aligned against an interval (x, y) in $[i_1, j_1]$. In both cases, for the edges of the superstring to be noncrossing, the interval $[x, y]$ must be a spanning interval of its corresponding string. Thus in the first case,

$$\begin{aligned} RSCS[i_1,j_1,i_2,j_2] \;=\; \min_{x,y} \; & \{RSCS[i_1,k_1-1,i_2,x-1] + \\ & RSCS[k_1,j_1,x,y] + len(t[y+1 \ldots j_2])\} \end{aligned}$$

where the minimization is over all $i_2 \leq x < y \leq j_2$ such that $[x, y]$ is a spanning interval of t. Similarly, in the second case,

$$\begin{aligned} RSCS[i_1,j_1,i_2,j_2] \;=\; \min_{x,y} \; & \{RSCS[i_1,x-1,i_2,k_2-1] + \\ & RSCS[x,y,k_2,j_2] + len(s[y+1 \ldots j_1])\} \end{aligned}$$

where $i_1 \leq x < y \leq j_1$ and $[x, y]$ is a spanning interval of s. Clearly, for each pair of spanning intervals, the complexity of computing RSCS is $O(n^2 + m^2)$, which implies theorem 3.6.

Theorem 3.6 *Let s, t be two RNA strings over $\sum$ with structure S_1 and S_2 respectively. An RSCS of s, t can be computed in $O(n^2m^2(n^2 + m^2))$ time.*

4 Inferring RNA structure via Alignment

Given two sequences $s[1 \ldots n]$ and $t[1 \ldots m]$, where s has a known structure S_1, we infer the structure of t by aligning the two sequences. This approach is useful if we know that the two

sequences are functionally related and have similar structure. For all $1 \leq i < j \leq m+n$, base-pairs $i - gap[2,i]$ and $j - gap[2,j]$ form a bond in t only if they are complementary and $(i - gap[1,i], j - gap[1,j]) \in S$. We would like to find an alignment that maximizes the sequence similarity score as well as the number of bonds formed in t. The algorithm for computing RNA alignment can be easily modified to accomplish this.

Consider the algorithm *AlignRNA* (fig. 2). Note that when we are comparing intervals $[i_1, j_1]$ and $[i_2, j_2]$, and there exists $(k_1, j_1) \in S$, there may be $\Omega(m)$ complementary pairs (k_2, j_2) that (k_1, j_1) can align against, and we need to pick the pair that gives the best alignment. Therefore, the naive algorithm has complexity $O(n^2m^3)$.

We improve the running time by reducing the number of intervals for which the computation of the score takes time $\Omega(m)$. First note that in aligning intervals $[i_1, j_1]$ and $[i_2, j_2]$, we can consider the cases where the leftmost or the rightmost column is involved only in character operations (insertions, deletions or substitutions). This leaves only the cases where both i_1 and j_1 are endpoints of edges in S. Even so, the number of such intervals $[i_1, j_1]$ is quadratic. However, we can use the tree structure on S to show that it suffices to carry out the more expensive computation for only a linear number of such intervals $[i_1, j_1]$. This leads to an algorithm with a running time of $O(n^2m^2 + nm^3)$.

Consider the case when the endpoints of the interval $[i_1, j_1]$ form an edge in S. Moreover, $s[i_1]$ must be matched against $t[i_2]$ and $s[j_i]$ against $t[j_2]$, else we are in a case of character operations addressed above. This step can be computed in constant time.

Next we consider the case when the endpoints of the interval $[i_1, j_1]$ participate in different edges in S. For this case, we will *binarize* the forest structure on S to yield a linear number of such intervals $[i_1, j_1]$ with the following property: the score of an optimal alignment of any such interval with t can be obtained by considering alignments of such subintervals with intervals in t.

Consider the forest defined by the elements of S and the function *parent*. We binarize the forest by introducing additional base-pairs in S to get S', so that each node in S' has at most 2 children. Procedure Binarize (fig. 3) accomplishes this, for a tree rooted at $(i, j) \in S$.

For the algorithm *InferStructure* (fig. 4), we assume that we already have the sets S and S' for s. Functions γ and δ are the costs for aligning bases and base-pairs respectively, as defined for RNA Alignment.

Observe that $|S'| = O(|S|)$, which implies that $|S'| = O(n)$. The first two conditional statements each take constant time and are executed for all possible $O(n^2m^2)$ intervals. The last conditional statement takes time $O(m)$, but is executed $O(|S'|m^2) = O(nm^2)$ times. This gives the following theorem.

Theorem 4.1 *The algorithm InferStructure computes the score of an optimal alignment for inferring structure in $O(m^2n^2 + nm^3)$ time.*

Procedure *Binarize*(i, j)
(* Assume that $(i, j) \in S$ has k children $\{(i_1, j_1), \ldots, (i_k, j_k)\}$ *)
begin
    $S' = \emptyset$.
    for $1 \leq u < k$ do
        Binarize(i_u, j_u)
        $S' = S' \cup \{(i_1, j_u)\}$
        if $(u > 1)$
            parent$((i_1, j_{u-1})) = (i_1, j_u)$
            parent$((i_u, j_u)) = (i_1, j_u)$
    parent$(i_1, j_k) = (i, j)$
end

Figure 3: Binarizing an RNA structure tree

Procedure *InferStructure*
begin
 for intervals (i_1, j_1), $1 \leq i_1 < j_1 \leq n$
    and intervals (i_2, j_2), $1 \leq i_2 < j_2 \leq m$
(* Assume that the intervals are examined in lexicographically increasing order of widths*)

$$Align[i_1, j_1, i_2, j_2] = \max \begin{cases} Align[i_1 + 1, j_1, i_2, j_2] + \gamma(s[i_1], `-') \\ Align[i_1, j_1, i_2 + 1, j_2] + \gamma(`-', t[i_2]) \\ Align[i_1 + 1, j_1, i_2 + 1, j_2] + \gamma(s[i_1], t[i_2]) \\ Align[i_1, j_1 - 1, i_2, j_2] + \gamma(s[j_1], `-') \\ Align[i_1, j_1, i_2, j_2 - 1] + \gamma(`-', t[j_2]) \\ Align[i_1, j_1 - 1, i_2, j_2 - 1] + \gamma(s[j_1], t[j_2]) \end{cases}$$

    if $(i_1, j_1) \in S$ and
        $t[i_2]$ and $t[j_2]$ are complementary base-pairs

$$Align[i_1, j_1, i_2, j_2] = \max \begin{cases} Align[i_1, j_1, i_2, j_2], \\ \delta(i_1, j_1, i_2, j_2) + \gamma(s[i_1], t[i_2]) \\ +\gamma(s[j_1], t[j_2]) + Align[i_1 + 1, j_1 - 1, i_2 + 1, j_2 - 1] \end{cases}$$

    else if $(i_1, j_1) \in S` - S$ and
        $(k, j_1)) = rightchild(i_1, j_1)$

$$Align[i_1, j_1, i_2, j_2] = \max \begin{cases} Align[i_1, j_1, i_2, j_2], \\ \max_{i_2 < l < j_2} \{Align[i_1, k - 1, i_2, l - 1] + Align[k, j_1, l, j_2]\} \end{cases}$$

end

Figure 4: Inferring structure of an RNA string

5 Conclusions

We have formulated and provided efficient algorithms for a number of problems on computing similarity between RNA strings. Among possible extensions, we mention two: careful modeling of energies of secondary structure primitives such as multiloops, bulges etc., and defining models that capture more general structures such as proteins. It is worthwhile to note that our algorithms for RNA string matching extend to structures that allow crossing edges as long as every base forms at most one bond.

6 Acknowledgements

We thank Martin Vingron for help with formulating some of these problems.

References

[1] K. Abrahamson. Generalized string matching. *SIAM J. Comp.*, 1987, 1039-1051.

[2] A. Amir and M. Farach. Efficient 2-dimensional Approximate Matching of Non-rectangular Figures. *Proc of 2nd Ann ACM Symp on Discrete Algorithms*, 1991, 212-222.

[3] D. Eppstein, Z. Galil, R. Giancarlo, and G.F. Italiano, "Sparse dynamic programming I: Linear cost functions," *JACM*, Vol. 39, No. 3, 1992, 519-545.

[4] D. Eppstein, Z. Galil, R. Giancarlo, and G.F. Italiano, "Sparse dynamic programming II: Convex and concave cost functions," *JACM*, Vol. 39, No. 3, 546-567 (1992).

[5] M. Fischer and M. Paterson. String Matching and other Products. *SIAM-AMS Proceedings*, Vol. 7, 113-125, 1974.

[6] L. Grate, M. Hebster. R. Hughey, D, Haussler, I. S. Mian and H. Noller, "RNA modeling using Gibbs sampling and stochastic context free grammars," *Second Intl. Conf. on Intelligent Systems for Molecular Biology* (1994).

[7] T. Jiang, L. Wang and K. Zhang, "Alignment of trees - an alternative to tree edit," *Proc. Combinatorial Pattern Matching Conf. 94*, LNCS 807, 75-86 (1994).

[8] P. Kilpeläinen and H. Mannila, "Query primitives for tree-structured data," *Proc. Combinatorial Pattern Matching Conf. 94*, LNCS 807, 213-225 (1994).

[9] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Computing*, 6:323–350, 1977.

[10] L. L. Larmore and B. Schieber, "On-line dynamic programming with applications to the prediction of RNA secondary structure," *Prof. First ACM-SIAM Symp. on Discrete Algorithms*, 503-512 (1990).

[11] S-Y Le, J. Owens, R. Nussinov, J-H. Chen, B. Shapiro and J. V. Maizel, "RNA secondary structures: comparison and determination of frequently recurring substructures by consensus," *CABIOS* Vol. 5, No. 3, 205-210 (1989).

[12] S. Muthukrishnan. New results and open problems related to non-standard stringology. *Proc. 6th Combinatorial Pattern Matching Conference*, 1995, 298-317.

[13] S. E. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino-acid sequences of two proteins," *J. Mol. Bio.*, 48, 443-453 (1970).

[14] R. Nussinov, G. Pieczenik, J. R. Griggs and D. J. Kleitman, "Algorithms for loop matchings," *SIAM J. Appl. Math.*, 35, 68-82 (1978).

[15] Y. Sakakibara, M. Brown, I. S. Mian, R. Underwood, and D. Haussler, "Stochastic context free grammars for modeling RNA," *Proc. the Hawaii Intl. Conf. on System Sciences*, IEEE Computer Society Press, Los Alamitos, CA, (1994).

[16] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood and D. Haussler, "Recent methods for RNA modeling using stochastic context-free grammars," *Proc. Combinatorial Pattern Matching Conf.*, LNCS 807, 289-306 (1994).

[17] D. Sankoff, "Simultaneous solution of the RNA folding, alignment and protosequence problems," *SIAM J. Appl. Math.* Vol. 45, No. 5, 810-825 (1985).

[18] B. A. Shapiro, "An algorithm for comparing multiple RNA secondary structures," *CABIOS*, Vol. 4, No. 3, 387-393 (1988).

[19] B. A. Shapiro and K. Zhang, "Comparing multiple RNA secondary structures using tree comparisons," *CABIOS* Vol. 6, No. 4, 309-318 (1990).

[20] T. F. Smith and M. S. Waterman, "The identification of common molecular subsequences," *J. Mol. Biol.* 147, 195-197 (1981).

[21] T. F. Smith and M. S. Waterman, "Comparison of biosequences," *Adv. in App. Math.* 2, 482-489 (1981).

[22] K-C Tai, "The tree to tree correction problem," *JACM*, Vol. 26, No. 3, 422-433 (1979).

[23] M. S. Waterman, "Secondary structure of single-stranded nucleic acids," *Studies in Foundations and Combinatorics, Advances in Mathematics supplementary studies* VOl. 1, Academic press, New York, 167-212 (1978).

[24] M. S. Waterman and T. F. Smith, "RNA secondary structure: a complete mathematical analysis," *Math. Biosci.* 42, 257-266 (1978).

[25] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems, *SIAM J. Comput.* 18, 1245-1262 (1989).

[26] K. Zhang, R. Statman, and D. Shasha, "On the editing distance between unordered labeled trees," *Inform. Proc. Lett.* 42, 133-139 (1992).

[27] M. Zuker, "On finding all suboptimal foldings of an RNA molecule," *Science*, 244 7, 48-52 (1989).

[28] M. Zuker and D. Sankoff, "RNA secondary structures and their prediction," *Bull. Math. Biol.* 46, 591-621 (1984).

[29] M. Zuker and P. Stiegler, "Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information," *Nucleic Acid Res.* 9, 133-148 (1981).