A faster algorithm for RNA co-folding

Michal Ziv-Ukelson [1] [⋆], **Irit Gat-Viks**[2] [∗], **Ydo Wexler** [3] [∗], and **Ron Shamir**[4]

[1] Computer Science Department, Ben Gurion University of the Negev, Beer-Sheva.
[2] Computational Molecular Biology Department, Max Planck Institute for Molecular Genetics, Berlin, Germany.
[3] Microsoft Research, Microsoft Corporation, Redmond, WA.
[4] School of Computer Science, Tel Aviv University.

Abstract. The current pairwise RNA (secondary) structural alignment algorithms are based on Sankoff's dynamic programming algorithm from 1985. Sankoff's algorithm requires $O(N^6)$ time and $O(N^4)$ space, where N denotes the length of the compared sequences, and thus its applicability is very limited. The current literature offers many heuristics for speeding up Sankoff's alignment process, some making restrictive assumptions on the length or the shape of the RNA substructures. We show how to speed up Sankoff's algorithm in practice via non-heuristic methods, without compromising optimality. Our analysis shows that the expected time complexity of the new algorithm is $O(N^4\zeta(N))$, where $\zeta(N)$ converges to $O(N)$, assuming a standard polymer folding model which was supported by experimental analysis. Hence our algorithm speeds up Sankoff's algorithm by a linear factor on average. In simulations, our algorithm speeds up computation by a factor of 3-12 for sequences of length 25-250.
Availability: Code and data sets are available, upon request.

1 Introduction

Within the last few years non-coding RNAs (ncRNAs) have been recognized as a highly abundant class of RNAs that do not code for proteins but nevertheless are functional in many biological processes, including localization, replication, translation, degradation, regulation and stabilization of biological macromolecules [19]. Thus, the computational identification of functional RNAs in genomes is a major, yet largely unsolved, problem. It is well known that structural conservation implies potential function and thus comparative structure analysis is the gold standard for the identification of functional RNAs and the determination of RNA secondary structures [22].

Many of the comparative genomics methods for the identification of functional RNA structures require a sequence alignment as input [16, 13, 23]. However, an alignment based on primary sequence alone is generally not sufficient for the identification of conserved secondary structure [6]. This is due to the fact that functional RNAs are not necessarily conserved on their primary sequence level. Instead, the stem-pairing regions of the functional RNA structures

[⋆] These authors contributed equally to the paper.

evolve such that substitutions that maintain the bonding between paired bases are more likely to survive. The base-pair covariation in the structural stems includes multiple compensatory substitutions (*e.g.* $G{:}C \rightarrow A{:}U$) and compatible single substitutions (*e.g.* $G{:}C \rightarrow G{:}U$). In 1985 David Sankoff proposed an algorithm [4], which we shall call SA, designed for the simultaneous alignment and structure prediction of homologous structural RNA sequences (for brevity, we call that problem here the *co-folding* problem). SA merges the recursions of sequence alignment such as Smith and Waterman's algorithm [25] with those of RNA structure prediction algorithms [20, 27]. SA takes as input two sequences and finds a local alignment (two substrings) such that the entire configuration of structure and alignment is optimal. To combine both structure and alignment scores, SA aims to maximize the weighted sum of the predicted structure's base-pairing interactions and the alignment cost. Base-pairing interactions scores (e.g., Nussinov et al. [20] or Zuker and Stiegler's energy-based methods [27]) guide homologous base pairs to align correctly and thus base-pair covariation can be naturally included as a factor in the alignment solution. The alignment cost aims to punish for insertions/deletions but encourage compensatory mutations among base-paired characters. SA preserves a common branching configuration for the aligned structures, but allows variations in the sizes of the stems and the loops (see Fig. 1).

Unfortunately, SA is not practical for most realistic applications due to its prohibitive computational cost: the algorithm requires $O(N^6)$ time and $O(N^4)$ space, for a pair of sequences of length N. The high complexity of this algorithm on one hand, and the need for practical solutions on the other hand, motivated attempts to reduce its complexity heuristically. In recent years, many studies have suggested practical heuristic methods to reduce SA's computational cost [1, 3, 15, 14, 7, 21, 11, 10], highlighting the importance of this algorithm in current RNA comparative structure analysis. Here, in contrast to the above heuristic approaches, we obtain a non-heuristic speedup, which does not sacrifice the optimality of results.

Our algorithm extends the approach of Wexler, Zilberstein and Ziv-Ukelson [26], previously applied to speeding up the classical $O(N^3)$ *RNA secondary structure prediction* algorithms [20, 27]. The classical algorithms for RNA secondary structure prediction are based on dynamic programming (DP) and their time complexities are $O(N^3)$, where the bottleneck is due to the fact that the recursion for computing the optimal folding includes a term that takes into account $O(N)$ possible branching points (see Figure 1) which "compete" for the optimal score. The speedup suggested in [26] to the classical $O(N^3)$ algorithm computes an exact optimal folding. This is done by pruning the number of branch points that need to be considered from $O(N)$ down to $\psi(N)$, where $\psi(N)$ is shown to be constant on average under standard polymer folding models. The accelerated algorithm uses a candidate-list approach to utilize two observations: (a) the main (2D) DP matrix computed by the classical algorithm for RNA secondary structure prediction obeys the triangle inequality; (b) a classical thermodynamic argument indicates that the probability for base-pair formation between two bases q indices apart is bounded by b/q^c for some constants $b, c > 0$ [17, 18].

Similarly to the RNA folding algorithms, the time-complexity bottleneck of SA is also due to the computation of all the scores induced by competing branch points. However, in contrast to the RNA folding algorithms, in the SA case the branch point considerations scale up to two orders, as all possible combinations of pairs of branch point indices (*i.e.* all possible indices of sequence $A \times$ all possible indices of sequence B, where A and B are the two co-folded sequences) need to be considered. This adds a factor of $O(N^2)$ to the time complexity of SA which is due to branch point considerations. In this paper we extend the approach of [26] and apply it to speed up SA by reducing the number of branch points that need to be considered in the main recursion for the SA score computation. Although the method suggested in [26] is not directly applicable to the $4D$ DP matrix computed in the SA algorithm, we show how to interpret the terms in the SA algorithm so as to maintain a set of candidate lists that will reduce the amount of computations needed. The resulting algorithm is guaranteed to obtain the optimal solution to the problem. In order to do this, we first show that the DP table computed by the SA recursion also obeys a sort of triangle inequality, and that the main theorem of [26] regarding redundant candidate branch-point considerations extends to the branch-point pairs considered by SA. Based on these proofs, we give a new candidate-list variant of SA that exploits redundancies in the main SA DP table to speed it up.

Under the probabilistic model of self avoiding random walk [24], which has been verified experimentally for polymers [17, 18], the expected time complexity of the new algorithm is $O(N^4\zeta(N))$, where $\zeta(N)$ is the expected maximal size of a candidate list. In contrast to [26], where $\zeta(N)$ converges to a constant, we show that in fact here $\zeta(N)$ converges to $O(N)$. Thus, our algorithm provides a theoretical speedup over the original SA by a linear factor on average. This behavior of the co-folding algorithm is verified in an experimental analysis, which shows a linear growth of the candidate list size with increasing sequence length. The faster new algorithm was implemented as a filter, denoted fastCoFold, on top of a popular SA implementation of Havgaard et al. [15]. Our run-time benchmarks show that fastCoFold is indeed faster in practice when compared to the standard SA version of the same code, by a factor of 3-12 for sequences of length 25-250.

The rest of this paper proceeds as follows. Section 2 gives the basic preliminaries and definitions including an algorithmic background. Section 3 examines properties of the folding and co-folding DP algorithms which are later exploited by our pruning method. The new algorithm is described and analyzed in Section 4, and a comparative performance analysis of the algorithm is given in Section 5.

Due to space restrictions, all proofs are omitted. Proofs, as well as a biological application of SA to the analysis of functional tandem repeats in *C. elegans*, will appear in an extended journal version of the paper.

2 Preliminaries and Definitions

RNA is typically produced as a single stranded molecule, which then folds upon itself to form a number of short base-paired stems (Fig 1). This base-paired structure is called the *secondary structure* of the RNA. Paired bases almost always occur in a nested fashion in RNA secondary structure. Under the assumption that the structure does not contain pseudoknots, a model was proposed by

```
     Left Branch        ][   Right Branch
A-CGGCAAA---UUGGCCG-U   ][  GCUGCGUGCAAAGCGC
((((((.........))))))   ][  ...((((.....))))
AUCGCGAAAAAAUUGCGCGAU   ][  GCUG-GCGCAAAGC-C
```

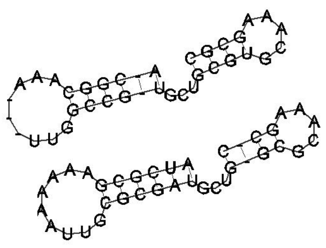


Fig. 1. Co-folding. Left: An example of a branching co-folding. Matched round parentheses indicate paired bases in the co-folding, and square brackets indicate a partition point. In the terminology of Section 2, the left branch is a co-terminus co-folding and the right side is a dangling co-folding. Right: The (aligned) secondary structures corresponding to the branching co-folding. Both branches consist of a stem (matched bases) and a loop.

Tinoco et al. [12] to calculate the stability (in terms of free energy) of a folded RNA molecule by summing all contributions from the stabilizing, consecutive base pairs from the loop-destabilizing terms in the secondary structure. This model has been widely investigated since and parameters for this model were experimentally collected *e.g.* by the Turner group [5]. Based on this model, DP algorithms for computing the most stable structures were proposed [20, 27].

The Sankoff co-folding algorithm [4] combines RNA folding with alignment, by taking into account both sequence and structure homology. This is formalized below in Definitions 1-4.

Definition 1 *An* **alignment** *of sequences R', T' is a pair of sequences R, T s.t. $|R| = |T|$, $R(T)$ is obtained from $R'(T')$ by adding '-'s such that for no index i, $r_i = t_i =$ '-'.*

The **alignment score** *is $\sum_i \sigma(R_i, T_i)$. Here, $\sigma(x, y)$ is the score of substituting x and y. The indel score $\sigma(x, -)$ is often indicated by $\gamma(x)$.*

Definition 2 *For a sequence F over the alphabet $\{(,), .\}$, where the parentheses are nested, define* $\mathbf{P(F)} = \{(\pi, \overline{\pi}) \mid \pi, \overline{\pi}$ *are paired parenthesis positions in $F\}$.*

Definition 3 *For a sequence T over the RNA alphabet $\{A, U, C, G, -\}$, a* **folding** *$F(T)$ is a series over the alphabet $\{(,), .\}$, annotated with the letters of T, such that (1)$|F(T)| = |T|$, (2) $F(T)$ is nested, and (3) for each paired parentheses positions $(\pi, \overline{\pi}) \in P(F)$, $\{T_\pi, T_{\overline{\pi}}\} \in \{\{A, U\}, \{G, C\}, \{G, U\}, \{-, -\}\}$.*

The **folding score** *is $\sum_{P(F)} \beta^T_{\pi\overline{\pi}}$, where β^T_{ij} denotes the score for the stability contribution of a base-pair between the characters in indices i and j of T.*

We note that some SA implementations relax requirement (3) in the above definition, and instead set β^T_{ij} to a score penalty in those cases when $\{T_i, T_j\}$

$\notin \{\{A,U\},\{G,C\},\{G,U\}\}$. Furthermore, the implementation of the term β^T_{ij} depends on the structure prediction algorithm employed by the specific SA variant. For example, in variants that are based on McCaskill's algorithm, such as PMCOMP [14], β^T_{ij} is the probability for the character at index i of T to form a base-pair with the character at index j of T. Alternatively, in SA variants that are based on Zuker's MFE approach, such as Foldalign [15], the score β^T_{ij} is in general computed dynamically depending on the structural context, making use of Turner parameters such as e.g. loop energy rules [27].

Definition 4 *A* **co-folding** *of two RNA sequences R', T' is the triplet R, T, F such that R, T is an alignment of R', T' and F is a folding of R and of T. The* **co-folding score** *is $\sum\limits_i \sigma(R_i, T_i) + \sum\limits_{P(F)} (\beta^T_{\pi\overline{\pi}} + \beta^R_{\pi\overline{\pi}} + \tau(S_\pi, R_{\overline{\pi}}, T_\pi, T_{\overline{\pi}}))$, where the term $\tau(R_\pi, R_{\overline{\pi}}, T_\pi, T_{\overline{\pi}})$ is a score that takes into account compensatory mutations and substitutions.*

Given a pair of RNA sequences A and B, the **local co-folding problem** *is to find two substrings $A[i \ldots j]$ and $B[k \ldots \ell]$ such that their co-folding has maximum score. The* **global co-folding problem** *is to find a co-folding of $A[1 \ldots N]$ and $B[1 \ldots N]$ of maximal score.*

Sankoff's algorithm solves the co-folding problem by DP. All of the current algorithms that are based on SA employ variants of the same basic DP recursion. We now demonstrate the recursions in FoldalignM [7] and Foldalign [15]. Given two RNA sequences A and B, let $S[i,j;k,\ell]$ hold the score of the best co-folding of the subsequences $A[i \ldots j]$ and $B[k \ldots \ell]$.

$$S[i,j\,;\,k,l] = \max \begin{cases} (1)\ S[i+1,j;k,\ell] \ + \gamma(A_i), \\ (2)\ S[i,j;k+1,\ell] \ + \gamma(B_k), \\ (3)\ S[i,j-1;k,\ell] \ + \gamma(A_j), \\ (4)\ S[i,j;k,\ell-1] \ + \gamma(B_\ell), \\ (5)\ S[i+1,j;k+1,\ell] \ + \sigma(A_i, B_k), \\ (6)\ S[i,j-1;k,\ell-1] \ + \sigma(A_j, B_l), \\ (7)\ S[i+1,j-1;k,\ell] + \beta^A_{ij} \ + \gamma(A_i) \ + \gamma(A_j) \\ (8)\ S[i,j;k+1,\ell-1] + \beta^B_{kl} \ + \gamma(B_k) \ + \gamma(B_\ell) \\ (9)\ S[i+1,j-1;k+1,\ell-1] + \beta^A_{ij} \ + \beta^B_{kl} \ + \tau(A_i, A_j, B_k, B_l), \\ (10)\ \max\limits_{i<m<j, k<n<l} \{\ S[i,m;k,n] \ + S[m+1,j;n+1,\ell]\ \} \end{cases} \tag{1}$$

where all entries of S are initialized to 0. Terms $(1)-(4)$ account for gaps in one of the two sequences. Terms (5) and (6) describe the extension of both subsequences with an unpaired position. Terms (7) and (8) describe the extension of only one of the subsequences with a base-pair. These two terms are conditional and are only applied in certain contexts: term (7) can only be applied if, in the co-folding corresponding to $S[i+1,j-1;k,\ell]$, A_{i+1} is base paired with A_{j-1}, and term (8) can only be applied if in the co-folding corresponding to $S[i,j;k+1,\ell-1]$ B_{k+1} is base paired with $B_{\ell-1}$. The constraints on terms 7 and 8 are only necessary in order to preserve the common branching configuration for the two aligned structures. Term (9) describes the extension of both sequences by a base-pair match, and Term (10) describes a branching event, as demonstrated in Figure 1. The following two definitions are also exemplified in Figure 1.

Definition 5 (co-terminus folding and co-folding) *A folding of a sequence $s_i \dots s_j$ is a* **co-terminus** *folding if s_i pairs with s_j. Otherwise it is called a* **dangling** *folding.*

Similarly, a **co-terminus co-folding** *over a pair of sequences $A[i, j]$ and $B[k, \ell]$ is a co-folding in which A_i pairs with A_j and B_k pairs with B_ℓ. Otherwise it is called a* **dangling** *co-folding.*

Definition 6 *A* **partition point** *in a given co-folding of $A[i, j]$ versus $B[k, \ell]$ is an index pair (p_1, p_2), such that there is no co-terminus folding over $A_x \dots A_y$ in this co-folding, where $i \leq x \leq p_1$ and $p_1 \leq y \leq j$, and in addition there is no co-terminus folding over $B_z \dots B_w$ in this co-folding, where $k \leq z \leq p_2$ and $p_2 < w \leq \ell$.*

Time and Space Complexity Analysis of SA**.**
Let $N = max\{|A|, |B|\}$. Computing the matrix S requires $O(N^4)$ space. For each entry $S[i, j, k, l]$ to be computed in this matrix, the algorithm employs the recursion of Eq. 1. The bottleneck of Eq. 1 is term (10), which considers $O(N^2)$ competing sums of pairs. Therefore, the time complexity is $O(N^6)$.

3 Properties of RNA co-folding

In this section we generalize the triangle inequality and polymer zeta properties, which were used for speeding up the folding algorithm [26], to the RNA Co-Folding problem. We start with a short review of the quadrangle inequality and the triangle inequality in the context of speeding up dynamic programming. Let M be an $n \times n$ matrix in which each entry $M(i, j)$, such that $i \leq j$, is computed by the following formula:

$$M(i, j) = \min_{i < i' \leq j} \{M(i, i') + M(i' + 1, j)\}$$

The well-known inverse quadrangle inequality property [9] is defined as follows.

Definition 7 *A matrix M obeys the* **inverse quadrangle inequality** *condition iff*

$$\forall \;\; i < i' < j < j' \qquad M(i, j') \leq M(i, j) + M(i', j') - M(j', j)$$

Both the quadrangle and the inverse quadrangle inequalities have previously been used to speed up dynamic programming [2, 9]. However, both the quadrangle inequality and the inverse quadrangle inequality are strong constraints on the input behavior, and do not apply to the matrix S computed by SA. However, a special weaker case of the classical inverse quadrangle inequality, the *triangle inequality* property, which is much more common in practice in various applications, will be extended in this paper and used to speed up RNA folding prediction.

Definition 8 *A matrix M obeys the* **triangle inequality** *property iff*

$$\forall \;\; i < j < j' \qquad M(i,j') \leq M(i,j) + M(j+1,j').$$

The matrix S is four dimensional and therefore the standard triangle inequality does not apply. However, we consider the following "extended" inverse triangle inequality property of a 4D matrix.

Definition 9 *A four dimensional matrix M obeys the* **4D inverse triangle inequality** *property iff*

$$\forall \;\; i < j < j' and \;\; \forall \;\; k < \ell < \ell'$$

$$M[i,j';k,\ell'] \geq M[i,j;k,\ell] + M[j+1,j';\ell+1,\ell']$$

The next claim is immediate from Definition 9 and Eq. 1.

Claim 1 *The matrix S, as computed by Eq. 1, obeys the 4D inverse triangle inequality.*

We next turn to review the polymer zeta property in the context of RNA folding.

Definition 10 *Consider the space θ of all possible foldings for a given RNA string $s_i \dots s_j$ under a given folding model Λ. Let $P(i,j)$ denote the probability for a folding in θ to be a co-terminus folding, and let $j - i = q$. We say that Λ has the* **polymer-zeta property** *if $P(i,j) \leq b/q^c$ for some constants $b, c > 0$.*

Experimental work has shown that RNA folding obeys the polymer-zeta property, namely, the probability that a co-terminus folding is formed over the subsequence, pairing two positions at distance q monomers apart, is $P(q) = b/q^c$ where $b = 1$ and $c > 1$ [17, 18]. This fact is explained by modeling the folding of a polymer chain as a self-avoiding random walk (SAW) in a 2D lattice [24].

The theoretical exponent for the 2D SAW model is known to be $c = 1.5$ [8]. In this paper we assume that the secondary structures corresponding to SA co-foldings obey the polymer zeta property. This assumption is supported by our computational results on real data, described in Section 5.

4 A Candidate List Filter for Computing the Matrix S

In this section we describe an alternative approach to the computation of S, which prunes redundant computations in the bottleneck term (10) of Eq. 1 without sacrificing optimality of results. The algorithm saves operations by filling the $O(N^4)$ matrix S in a specific order, avoiding certain computations that are suboptimal. For each combination of subsequence start points and end points, the original algorithm requires $O(N^2)$ sums in term (10). Instead, our algorithm will identify certain endpoint combinations for which a fraction of the sums is already guaranteed not to yield the optimal score, thereby saving from the $O(N^2)$ time needed for computing all sums.

We start by describing the order in which we traverse and fill the matrix S. Recursion 1 requires the availability of both values $S[i+1,j,k,\ell]$ and

$S[i,j,k+1,\ell]$ during the consideration of terms (1) and (2), correspondingly, in the computation of $S[i,j,k,\ell]$. Symmetrically, it also requires the availability of $S[i,j-1,k,\ell]$ during the computation of term (3) and of $S[i,j,k,\ell-1]$ during the computation of term (4). Theorem 1, which is given later in this section, shows that, for a given row in S, some partition points (with smaller m and n values) dominate others (with greater m and n values) in the computation of term (10), and that this dominance holds for all other entries in the row such that $j > m$ and $\ell > n$. In order to exploit this dominance property, as well as maintain the precedence order necessary for the application of Recursion 1, we will processes the entries of S in decreasing i index order and in decreasing k index order first (in other words: we compute S row-by-row, in decreasing row index order). For each index-pair (i,k), defining a specific row in S, we will compute the entry values in increasing ℓ index order first and then in increasing j index order (in other words: we maintain a left-to-right cell-traversal order within each row). Therefore, let *beam(i,k)* denote the ordered series of entries $S[i,j;k,\ell]$, for $j = i \dots N, \ell = k \dots N$, first in increasing ℓ index, and then in increasing j index. For simplicity of presentation, we will refer to each entry by its order of traversal within its beam, *i.e.* entry $S[i,j;k,\ell]$ will be denoted *entry* (j,ℓ) *of* $beam(i,k)$. Clearly, there are $O(N^2)$ possible beams in S and each beam covers $O(N^2)$ entries, left-to-right.

Note that each sum in term (10) is defined by its start points (i,k), its end points (j,ℓ) and its partition point (m,n). However, when considering all the $O(N^4)$ sums computed per beam, we note that a specific partition point (m,n) participates in the sums applied per computations for all end indices (m',n') such that $m \leq m' \leq j$ and $n \leq n' \leq \ell$. Therefore, in this section we view term (10) of Recursion 1 as a competition between partition points (m,n), $m = i+1 \dots j-1, n = k+1 \dots \ell-1$ for the branching event that yields the best score for $S[i,j;k,\ell]$. The term $S[i,m;k,n]$ of Recursion 1.(10) will be called the *left branch*, while the other term will be called the *right branch*. For each entry traversed by a beam, the naive SA computes the sums corresponding to $O(N^2)$ partition points in Recursion 1.(10). The following lemma and theorem show that some of these partition points are dominated by others (*i.e.* there are other partition points that yield equal or better score) and can thus be excluded from the computation.

Lemma 1. *Without loss of optimality, Recursion 1 can be constrained so that the left branch in term (10) is always a co-terminus co-folding.*

Naively, after constraining all left branches in term (10) to co-terminus co-foldings, there are still $O(N^2)$ partition points that compete for the optimal score in term (10), and thus altogether $O(N^4)$ sums of pairs computed per beam. However, the next theorem exposes a dominance relationship among the competing partition points, based on the 4D inverse triangle inequality property of S.

Theorem 1. *Suppose* $S[i,j;k,\ell] \leq S[i,m;k,n]+S[m+1,j;n+1,\ell]$ *for some* $i < m < j$ *and* $k < n < \ell$. *Then,* $\forall\ j' > j, \ell' > \ell \qquad S[i,j;k,\ell] + S[j+1,j';\ell+1,\ell'] \leq S[i,m;k,n] + S[m+1,j';n+1,\ell']$.

Theorem 1 exposes redundancies in the repeated computation of term (10) throughout the beam entry traversal, redundancies which could be avoided by maintaining a list of only those candidate partition points that are not dominated by others.

Definition 11 (candidate) *A partition point* (m, n) *is a* **candidate** *during the computation of all entries* $(j, \ell) \in beam(i, k)$ *such that* $m < j$, $n < \ell$, *iff*
1. $S[i, m; k, n]$ *corresponds to a co-terminus co-folding.*
2. $S[i, m; k, n] > S[i, m'; k, n'] + S[m' + 1, j; n' + 1, \ell]$ $\forall\ i < m' < m, k < n' < n$.

The above definition can be applied to speed up the computation of $S(i, j; k, \ell)$, as follows: rather than considering all possible $O(N^2)$ partition points, one could query the list that contains only partition points that satisfy the candidacy criteria above. This can be done by further reformulating term (10) as follows,

$$(10) \max_{\substack{\forall (m,n) \in candidate\_list: \\ m<j \wedge n<l}} \{\ S[i, m; k, n] + S[m + 1, j; n + 1, \ell]\ \} \tag{2}$$

The pseudocode for the new algorithm, denoted fastCoFold, is given below. *Procedure ComputeBeam* in Algorithm fastCoFold replaces the original term (10) in Recursion 1 with its constrained reformulation as Eq. 2. This is implemented as a candidate list that is empty at the start of each beam traversal, and is extended throughout the left-to-right computation of the entries of the beam, by appending to the list only those partition points that are candidates by Definition 11.

Each partition point (m, n) is considered for candidacy once per procedure call, when entry $S[i, m; k, n]$ is reached by the beam traversal, and will join the candidate list only if, at this point, the value of this entry dominates all its preceding partition points on the list. Each entry traversed by $beam(i, k)$ is computed by the new algorithm as before, with the only difference being that the maximum of term (10) is taken only over preceding pairs (m, n) from the candidate list, as formalized in Eq. 2.

In the following theorem we assume that the 2D structures corresponding to SA co-foldings follow the RNA 2D SAW model (see Section 3), and that therefore the probability for a co-terminus co-folding follows the polymer-zeta property with $c > 1$. This assumption is supported by the experimental results in Section 5.

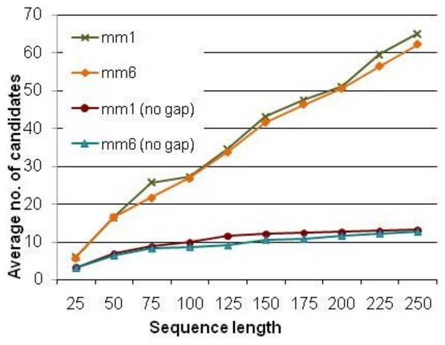
Theorem 2. *Algorithm* fastCoFold *improves* SA *by a linear factor on average.*

5 Performance Testing

To test the power of algorithm fastCoFold in practice, we implemented it as a filter on top of the FoldAlign SA program [15] in its version that computes an optimal solution with no heuristic shortcuts. We then compared the performance of fastCoFold with that of the original version of FoldAlign. For this, we

```
Algorithm FASTCOFOLD :
1  for each row $i := N$ to 1 do
2      for each row $k := N$ to 1 do
3          call Procedure ComputeBeam(i,k);

Procedure ComputeBeam(i,k):
1  $candidate\_list \leftarrow NULL$
2  for each column $j := i$ to $N$ do
3      for each column $l := k$ to $N$ do
4          $S\_dang[i,j;k,l] \leftarrow$
               $maximal\ score\ among\ terms$ (1) – (6) of Eq. 1
5          $S\_co\text{-}terminus[i,j;k,l] \leftarrow$
               $maximal\ score\ among\ terms$ (7) – (9) of Eq. 1
6          $S\_branch[i,j;k,l] \leftarrow$
               $\max_{\substack{\forall(m,n)\in candidate\_list:\\ m\le j\wedge n\le l}} \{S[i,m;k,n] + S[m+1,j;n+1,l]\}$
7          $S[i,j;k,l] \leftarrow \max\{S\_dang[i,j;k,l], S\_branch[i,j;k,l]\}$
8          if $(S\_co\text{-}terminus[i,j;k,l] > S[i,j;k,l])$ then
9              $S[i,j;k,l] \leftarrow S\_co\text{-}terminus[i,j;k,l]$
10             Append $(j,l)$ to the $candidate\_list$ for $(i,k)$
```



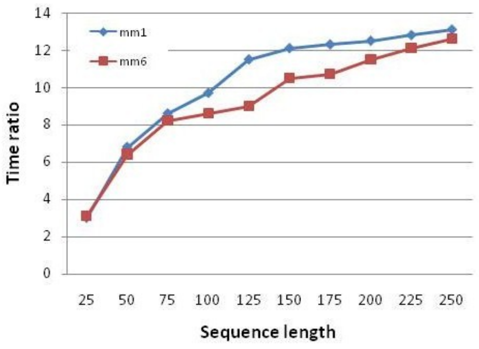


Fig. 2. Performance of FASTCOFOLD . (left) The average number of candidates in a list when running FASTCOFOLD. A "(no gap)" next to the benchmark name indicates that the parameter defining the allowed size difference between any two aligned subsequences was set to 0. (right) The average ratio between the run time of SA as implemented in Foldalign [15] and FASTCOFOLD (without any gap constraints), for different sequence lengths.

generated sequences of length 25,50,75 . . ., 250. Two datasets of 50 sequences each were generated for each length. Each pair of sequences of the same length in the same data set were co-folded. Altogether, we performed 24,500 co-foldings. The two sets were generated randomly according to a Markov model trained on RNA sequences randomly chosen from complete human mRNA sequences taken from the RefSeq database at NCBI `www.ncbi.nlm.nih.gov/RefSeq`. Sets "mm1" and "mm6" were generated using a Markov model of order 1 and 6, respectively. We ran FASTCOFOLD on each data set and measured running times and candidate list sizes. Two versions of FoldAlign and FASTCOFOLD were run: (1) allowing no gaps; (2) the full version allowing gaps of unbounded size. The results for FASTCOFOLD are shown in Figure 2(left). Reassuringly, in all runs the averages grow at most linearly with the length of the sequence. Figure 2(right) plots the average ratio between the run times of the two algorithms (both applied without any gap constraints) as a function of sequence length for both data sets. Runs were conducted on an Intel Xeon 2.8GHZ computer with 4GB RAM. The overall run-time for an all-against-all co-folding of a set of 50 sequences (including I/O time) varied from 2 seconds for the 25 bps sequences, to up to more than 12 hours for the 250 bps sequences. We suspect that the non-linear behavior of this graph is mostly due to exhausting memory resources in the benchmark computer.

Note that two heuristic constraints are used by Foldalign to reduce the time complexity. One constraint binds the total size of the allowed gaps, $(j-i)-(\ell-k)\leq\delta$. The other constraint, which can only be applied in the case of local alignments, binds the size of the compared subsequences $j-i\leq\lambda$ and $\ell-k\leq\lambda$. By applying both heuristics, a constrained version of SA is obtained with time complexities of $O(N^5)$ for global alignment with gaps bounded by a constant, and $O(N^4)$ for local alignment with both gaps and alignment sizes bounded by a constant. We observe that when δ is set to zero (no gaps allowed), ζ converges to a constant (see the "no gap" plot in Figure 2.a). Thus, using FASTCOFOLD, one can achieve an $O(N^4)$ SA that limits only the gaps and not the size of the alignment.

Acknowledgments We thank Gary Benson for helpful advice and for elaborate demonstrations of TRDB. We are also grateful to the anonymous WABI2008 referees for their helpful comments. RS was supported in part by the Wolfson Foundation and by the Raymond and Beverley Sackler Chair in Bioinformatics. A preliminary version of this study was performed when MZU was supported by a fellowship from the Edmond J. Safra Program in Bioinformatics at Tel Aviv University.

References

1. Uzilov AV., Keegan JM., and Mathews DH. Detection of non-coding RNAs on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7:173, 2005.
2. M. Crochemore, G. M. Landau, B. Schieber, and M. Ziv-Ukelson. Re-use dynamic programming for sequence alignment: An algorithmic toolkit. pages 19–60, 2005.
3. Mathews D. and Turner D. Dynalign: An algorithm for finding the secondary structure common to two RNA sequences. *Journal of Molecular Biology*, 317:191–203, 2002.
4. Sankoff D. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45:810–825, 1985.
5. Mathews DH., Burkard ME., Freier SM., Wyatt JR., and Turner DH. Predicting oligonucleotide affinity to nucleic acid target. *RNA*, 5:1458, 1999.

6. Rivas E. and Eddy SR. Secondary structure alone is generally not statistically significant for the detection of non-coding RNAs. *Bioinformatics*, 16:583–605, 2000.
7. Torarinsson E., Havgaard JH., and Gorodkin J. Multiple structural alignment and clustering of RNA sequences. *Bioinformatics*, 23(8):926–932, 2007.
8. M. E. Fisher. Shape of a self-avoiding walk or polymer chain. *J.Chem. Phys*, 44:616–622, 1966.
9. R. Giancarlo. *Dynamic Programming: Special Cases.* Oxford University Press, 1997.
10. Kiryu H., Tabei Y., Kin T., and Asai K. Murlet: a practical multiple alignment tool for structural RNA sequences. *Bioinformatics*, 23:1588–1598, 2007.
11. I. Holmes. Accelerated probabilistic inference of RNA structure evolution. *BMC Bioinformatics*, 6:73, 2005.
12. Tinoco I., Borer PN., Dengler B., Levine MD., Uhlenbeck OC., Crothers DM., and Gralla J. Improved estimation of secondary structure in ribonucleic acids. *Nature New Biology*, 246:40–41, 1973.
13. Hofacker IL., Fekete M., and Stadler PF. Secondary structure prediction for aligned RNA sequences. *Journal of Molecular Biology*, 319:1059–1066, 2002.
14. Hofacker IL., Bernhart S., and Stadler P. Alignment of RNA base pairing probability matrices. *Bioinformatics*, 20:2222–2227, 2004.
15. Havgaard JH., Lyngso RB., Stormo GD., and Gorodkin J. Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. *Bioinformatics*, 21(9):1815–1824, 2005.
16. Pederson JS., Bejerano G., Siepel A., Rosenbloom K., Lindblad-Toh K., Lander ES., Kent J., Miller W., and Haussler D. Identification and classification of conserved RNA secondary structres in the human genome. *PLOS Computational Biology*, 2:e33, 2006.
17. A. Kabakcioglu and A.L. Stella. A scale-free network hidden in the collapsing polymer. *ArXiv Condensed Matter e-prints*, September 2004.
18. Y. Kafri, D. Mukamel, and L. Peliti. Why is the dna denaturation transition first order? *Physical Review Letters*, 85:4988–4991, 2000.
19. Mandal M. and Breaker RR. Gene regulation by riboswitches. *Cell*, 6:451–463, 2004.
20. R. Nussinov and A.B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proc. Natl. Acad. Sci.*, 77(11):6309–6313, 1980.
21. Dowell RD. and Eddy S. Efficient pairwise RNA structure prediction and alignment using sequence alignment constraints. *BMC Bioinformatics*, 7:400, 2006.
22. Griffiths-Jones S. The microrna registry. *Nucleic Acids Research*, 32:D109–D111, 2003.
23. Washietl S. and Hofacker IL. Consensus folding of aligned sequences as a new measure for the detection of functional RNAs by comparative genomics. *Journal of Molecular Biology*, 342:19–30, 2004.
24. C. Vanderzande. *Lattice Models of Polymers (Cambridge Lecture Notes in Physics 11).* Cambridge University Press, 1998.
25. M.S. Waterman and T.F. Smith. Rapid dynamic programming algorithms for RNA secondary structure. *Adv. Appl. Math.*, 7:455–464, 1986.
26. Wexler Y., Zilberstein C., and Ziv-Ukelson M. A study of accessible motifs and the complexity of RNA folding. *Journal of Computational Biology*, 14(6):856–872, 2007.
27. M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, 1981.