Sparse RNA Folding: Time and Space Efficient Algorithms

Rolf Backofen[1], Dekel Tsur[2], Shay Zakov[2⋆], and Michal Ziv-Ukelson[2]

[1] Albert Ludwigs University, Freiburg, Germany
backofen@informatik.uni-freiburg.de
[2] Department of Computer Science, Ben-Gurion University of the Negev, Israel
{dekelts, zakovs, michaluz}@cs.bgu.ac.il

Abstract. The classical algorithm for *RNA single strand folding* requires $O(nZ)$ time and $O(n^2)$ space, where n denotes the length of the input sequence and Z is a sparsity parameter that satisfies $n \leq Z \leq n^2$. We show how to reduce the space complexity of this algorithm. The space reduction is based on the observation that some solutions for subproblems are not examined after a certain stage of the algorithm, and may be discarded from memory. This yields an $O(nZ)$ time and $O(Z)$ space algorithm, that outputs both the cardinality of the optimal folding as well as a corresponding secondary structure. The space-efficient approach also extends to the related *RNA simultaneous alignment with folding* problem, and can be applied to reduce the space complexity of the fastest algorithm for this problem from $O(n^2m^2)$ down to $O(nm^2 + \tilde{Z})$, where n and m denote the lengths of the input sequences to be aligned, and $\tilde{Z}$ is a sparsity parameter that satisfies $nm \leq \tilde{Z} \leq n^2m^2$.
In addition, we also show how to speed up the base-pairing maximization variant of RNA single strand folding. The speed up is achieved by combining two independent existing techniques, which restrict the number of expressions that need to be examined in bottleneck computations of these algorithms. This yields an $O(LZ)$ time and $O(Z)$ space algorithm, where L denotes the maximum cardinality of a folding of the input sequence.

Additional online supporting material may be found at:
http://www.cs.bgu.ac.il/~zakovs/RNAfold/CPM09_supporting_material.pdf

1 Introduction

The structure of RNA is evolutionarily more conserved than its sequence and is thus key to its functional analysis [1]. Unfortunately, although massive amounts of sequence data are continuously generated, the number of known RNA structures is still very limited since experimental methods, such as NMR and Crystallography, require expertise and long experimental time. Therefore, computational methods for predicting RNA structures are of great value [2–4].

⋆ To whom correspondence should be addressed.

RNA is typically produced as a single stranded molecule, which then folds upon itself to form a number of short base-paired stems. This base-paired structure is called the *secondary structure* of the RNA. The secondary structure almost always does not contain pseudoknots (i.e. crossing base pairs). Under the assumption that the structure does not contain pseudoknots, a model was proposed by Tinoco et al. [5] to calculate the stability (in terms of free energy) of a folded RNA molecule by summing all contributions from the stabilizing, consecutive base pairs and from the loop-destabilizing terms in the secondary structure. Based on this model, dynamic programming algorithms were suggested for computing the most stable structures [6–10], applying various scoring criteria such as the maximal number of base pairs [7] or the minimal free energy [8]. This optimization problem is formally denoted *RNA single strand folding*, and the time and space complexities of the classical algorithms for solving it are $O(n^3)$ and $O(n^2)$, respectively, where n denotes the length of the input RNA sequence. Recently, these were sped up to yield $O(nZ)$ time and $O(n^2)$ space [10] algorithms, where Z is a sparsity parameter that satisfies $n \leq Z \leq n^2$. We note that these algorithms are practical in the sense that the hidden constants are small. On a more theoretical front, Akutsu suggested an $O(D(n))$ algorithm for this problem [9], where $D(n)$ is the time for computing the distance product of two $n \times n$ matrices. The best current bound on $D(n)$ is $O(n^3 \log^3 \log n / \log^2 n)$ [11].

Another approach to RNA folding is the *simultaneous alignment with folding* (SAF for short) [12–16]. This approach consists of finding an optimal alignment between a set of RNA sequences, where an alignment score is evaluated with respect to some common folding of the input sequences. However, as stated in [17], even for the simple case where the input consists of only two sequences, this approach requires "extreme amounts of memory and space" with complexity of $O(n^2m^2)$ space and $O(n^3m^3)$ time, where n and m are the lengths of the input RNA sequences to be aligned. Thus, most existing practical implementations of this algorithm [13, 14, 16] use restricted versions of the original problem. Since these restrictions introduce another source of error, it is of utmost practical importance to the research on RNA to improve both the space and time complexities of the full version of SAF. A first non-heuristic speedup, which does not sacrifice the optimality of results, was recently described in [15]. This work extends the approach of [10] and yields an $O(nm\tilde{Z})$ time and $O(n^2m^2)$ space algorithm for the SAF problem, where $\tilde{Z}$ is a sparsity parameter that satisfies $nm \leq \tilde{Z} \leq n^2m^2$. However, experimental analysis of this algorithm indicates that the high memory requirements pose a major bottleneck in practice, both in constraining the lengths of the input sequences, as well as in exhausting the benchmark machine's memory, which in turn results in a page-fault slowdown.

Our contribution

(1.) Reducing the space requirements of RNA folding problems. In this work we focus on improving the space complexity of the base-pairing maximization variant of the RNA single strand folding problem [6, 7, 9]. The space requirement reduction is based on the observation that some solutions for subproblems are

	Previous results		New results	
	Time	Space	Time	Space
Single strand base-paring maximization	$O(n^3)$[7] $O(nZ)$[10] $O(D(n))$[9]	$O(n^2)$	$O(LZ)$	$O(Z)$
Single strand energy minimization	$O(n^3)$[8] $O(nZ)$[10]	$O(n^2)$	$O(nZ)$	$O(Z)$
Simultaneous alignment with folding	$O(n^3m^3)$[12] $O(nm\tilde{Z})$ [15]	$O(n^2m^2)$	$O(nm\tilde{Z})$	$O(nm^2+\tilde{Z})$

Table 1. Time and space complexities of RNA folding algorithms.

not examined after a certain stage of the algorithms, and may be discarded from memory. This yields an $O(nZ)$ time and $O(Z)$ space algorithm for this problem. In addition to the optimal folding cardinality computation, we show a trace-back procedure which outputs a corresponding secondary structure. Note that it is an interesting challenge on its own to recover an optimal folding within the time and space complexity bounds of the space-reduced algorithm, since due to the sparse representation only partial information is kept. The presented strategy may also be extended to the score computation of a family of RNA folding algorithms, which includes algorithms for the energy minimization variant of the single strand folding problem [8, 10] (improving the space complexity from $O(n^2)$ to $O(Z)$), as well as algorithms for SAF [12, 15] (improving the space complexity from $O(n^2m^2)$ to $O(nm^2+\tilde{Z})$).

(2.) A sparse RNA single strand folding algorithm. We also describe a fast algorithm for the base pairing maximization variant of RNA single strand folding that exploits an additional sparsity parameter, based on the cardinality of the optimal folding. This is achieved by combining two independent techniques, which were previously used to reduce the number of sub-instance pairs that need to be considered by the algorithm. This combination yields the simultaneous exploitation of two key properties emerging from the formal definitions of these folding problems: the triangle inequality property, previously exploited in [10] and [15], as well as the monotonicity and unit-step properties, previously utilized in [18] for a related problem. The result is an $O(LZ)$ time and $O(Z)$ space algorithm, where L denotes the maximum cardinality of a folding of the input sequence and $n \leq Z \leq n(L+1)$.

We note that the algorithms described here are practical in the sense that the hidden constants are small. In the context of practical contribution, we also point out that our space complexity improvements are more significant than the time complexity improvements, since while the expected value of L is $\Theta(n)$ (assuming uniform character distribution), both Z and $\tilde{Z}$ were experimentally shown to be significantly less than n^2 and n^2m^2, respectively [10, 15]. Furthermore, reducing the space complexity of the SAF problem is a key result in practice,

as in the previous results the space complexity was typically the computational bottleneck [17, 15].

Due to space constrains, figures, pseudocode and some omitted proofs are differed to an online supporting material document at `http://www.cs.bgu.ac.il/~zakovs/RNAfold/CPM09_supporting_material.pdf`.

2 Preliminaries

An RNA sequence is a sequence over the alphabet $\{A, C, G, U\}$. Each letter in an RNA sequence is also called a *base*. The bases A and U are called *complementary bases*, and so are the bases C and G [3]. For a base $\sigma \in \{A, C, G, U\}$, denote by $\overline{\sigma}$ the complementary base of σ. Fix henceforth an RNA sequence $S = s_1 s_2 \cdots s_n$. Denote by $S_{i,j}$ the subsequence $s_i \cdots s_j$ of S, where $S_{i,i-1}$ is defined to be an empty sequence.

Definition 1. *A* folding F *of a subsequence* $S_{i,j}$ *is a set of index pairs that satisfies the following:*

1. *For every* $(k,l) \in F$, $i \leq k < l \leq j$, *and* $s_l = \overline{s_k}$.
2. *There are no* $(k,l),(k',l') \in F$, *such that* $k \leq k' \leq l \leq l'$.

A pair $(k,l) \in F$ is called a *base-pair*. Say that index k is *paired* in a folding F if k appears in a base-pair in F, otherwise k is *unpaired* in F. Call an index q a *branch point* with respect to F if for all $(k,l) \in F$, either $l < q$ or $k \geq q$. We distinguish between two kinds of foldings of $S_{i,j}$: *co-terminus foldings* are foldings that include the base-pair (i,j), and *partitionable foldings* are those who do not include the base-pair (i,j). Note that for $j > i$, F is partitionable if and only if F has a branch point $i < q \leq j$. Denote by $|F|$ the *size* of a folding F, i.e. the number of base-pairs in F. The *single strand base-pairing maximization problem* was first addressed in [7]. The formal problem definition is given below.

Problem 1. Compute the maximum size of a folding of the instance sequence S.

Definition 2. *For a subsequence* $S_{i,j}$, *denote:*

1. $L(i,j)$ *is the maximum size of a folding of* $S_{i,j}$.
2. $L^p(i,j)$ *is the maximum size of a partitionable folding of* $S_{i,j}$.
3. $L^c(i,j)$ *is the maximum size of a co-terminus folding of* $S_{i,j}$, *or* $-\infty$ *if there is no such folding (if* $j \leq i$ *or* $s_j \neq \overline{s_i}$*).*

Call a folding F of $S_{i,j}$ for which $|F| = L(i,j)$, an *optimal* folding of $S_{i,j}$. In the rest of this paper, we use L instead of $L(1,n)$ whenever the context is clear.

[3] For the sake of clarity, we disregard the possible "wobble" pairing between G and U. All presented results may be easily extended to include $G-U$ pairing as well.

3 RNA Folding via base-pairing maximization

In this section we describe a recursive solution for the single strand base-pairing maximization problem, and present a technique for reducing its space complexity. This technique also extends to the single-strand RNA folding algorithms that are based on a thermodynamic model [2–4]. In addition, we suggest how to extend the space-reduction technique and apply it to the SAF problem [12, 15].

3.1 A recursive solution

For a subsequence $S_{i,j}$ such that $j \leq i$, the only possible folding is the empty folding, and therefore $L(i,j) = 0$. The following equations show how to recursively compute $L(i,j)$ when $j > i$:

$$L(i,j) = \max\left\{L^p(i,j), L^c(i,j)\right\}. \tag{3.1}$$

$$L^c(i,j) = \begin{cases} L(i+1,j-1)+1, & s_j = \overline{s_i}, \\ -\infty, & s_j \neq \overline{s_i}. \end{cases} \tag{3.2}$$

$$L^p(i,j) = \max_{i<q\leq j}\left\{L(i,q-1) + L(q,j)\right\}. \tag{3.3}$$

Note that the time complexity bottleneck in algorithms which implement the recursive computation of Equations 3.1 to 3.3 is due to the consideration of $O(n)$ branch points q in the computation of $L^p(i,j)$, according to Equation 3.3. In the rest of this section, as well as in Section 4, we describe techniques that reduce the number of branch points that need to be examined in this computation, and thus improve the time complexity of such algorithms. Due to Equations 3.1 and 3.3, the following (inverse) triangle inequality is sustained in the base-paring maximization problem:

Observation 1 (triangle inequality) *For every subsequence $S_{i,j}$ and for every $i < q \leq j$, $L(i,j) \geq L(i,q-1) + L(q,j)$.*

Based on the triangle inequality, Wexler et al. [10] observed that it is sufficient to examine only a subset of the branch points in order to compute $L^p(i,j)$. We present here a slightly different notation for the same concept.

Definition 3 (OCT). *A subsequence $S_{i,j}$ is* optimally co-terminus (OCT) *if $i = j$, or if every optimal folding of $S_{i,j}$ is co-terminus (that is, if $L(i,j) = L^c(i,j) > L^p(i,j)$).*

Call an index q for which $L^p(i,j) = L(i,q-1) + L(q,j)$ an *optimal branch point* with respect to $S_{i,j}$.

Lemma 1 (Wexler et al. [10]). *For every subsequence $S_{i,j}$, there is an optimal branch point q with respect to $S_{i,j}$ such that $S_{q,j}$ is an OCT.*

Define the following subset of branch points with respect to $S_{i,j}$:

$$Q_{i,j} = \{i < q \leq j : S_{q,j} \text{ is an OCT}\}.$$

The following equation restates Equation 3.3, based on Lemma 1, by restricting the branch points considered by the maximization term to those in $Q_{i,j}$.

$$L^p(i,j) = \max_{q \in Q_{i,j}} \{L(i,q-1) + L(q,j)\}. \tag{3.4}$$

We define the following sparsity measure of RNA sequences.

Definition 4. *For a subsequence $S_{i,j}$, $Z(i,j)$ is the number of subsequences of $S_{i,j}$ which are OCTs.*

In the rest of this paper, we use Z instead of $Z(1,n)$ whenever the context is clear. In the sparse case, only a small portion of the $O(n^2)$ subsequences of S are OCTs. In Section 4.1 we show that, in the base pairing maximization variant of the problem, Z is bounded by $n(L+1)$. For the minimum free energy problem variant, an estimation of the expected value of a parameter related to Z, based on a probabilistic model for polymer folding and measured by simulations, which shows that that Z is significantly smaller than $O(n^2)$, can be found in [10].

Previous algorithms for the base-pairing maximization problem were presented by Nussinov and Jacobson [7] and by Wexler et al. [10][4]. Both algorithms are dynamic programming algorithms that perform a bottom-up computation of the recurrence described in this section, where the Nussinov-Jacobson algorithm uses Equation 3.3 for the computation of $L^p(i,j)$, and the Wexler et al. algorithm improves it by using Equation 3.4. These algorithms compute the upper triangle of a table $M_{n \times n}$, where each cell $M[i,j]$ stores the value $L(i,j)$. The entries of M are traversed in an order which guarantees that all values that are needed for the computation of $M[i,j] = L(i,j)$, according to the recurrence formula, are computed and stored in M prior to the computation of $M[i,j]$. Upon termination, $M[1,n]$ holds the value L. The time complexity of the algorithm by Nussinov and Jacobson is $O(n^3)$, whereas that of the algorithm by Wexler et al. is $O(nZ)$. Both algorithms use $O(n^2)$ space.

3.2 A space efficient algorithm

Our space reduction strategy is based on the observation that some of the values stored by the algorithm of Wexler et al. [10] are not necessary throughout the complete run of the algorithm. In the following lemma we characterize the values that need to be maintained in memory for the computation of $L(i,j)$.

Lemma 2. *For a subsequence $S_{i,j}$, it is possible to compute $L(i,j)$ by examining only those values $L(a,b)$, where $i \leq a < b \leq j$ and $b - a < j - i$, which sustain that either $a = i$, $a = i+1$, or $S_{a,b}$ is an OCT.*

[4] [10] deals with the more realistic *energy minimization* variant of the problem. For clarity, we project their notions on the simpler *base-paring maximization* variant discussed here.

Proof. Immediate from Equations 3.1, 3.2 and 3.4. □

Consider a dynamic programming algorithm which fills the table M by traversing its entries row by row from bottom to top, and each row from left to right. Lemma 2 implies that at the stage where $M[i,j]$ is computed, it is sufficient to keep only the values in the currently computed i-th row, the values in the recently computed $(i+1)$-th row, and values in entries which correspond to OCT subsequences of S. Thus, there is no need to maintain the complete table M in memory, rather, at each stage, entries which are guarantied not to be further examined by the algorithm may be discarded. This yields a total space complexity of $O(n+Z) = O(Z)$. Note that the computation of each entry $M[i,j]$ requires $O(|Q_{i,j}|)$ operations, due to the consideration of the branch point set $Q_{i,j}$ (these sets are maintained as lists in order to allow an efficient traversal, as explained in [10]). Since $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}|Q_{i,j}| \leq \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}|Q_{1,j}| \leq \sum_{i=1}^{n-1} Z < nZ$, the running time of the algorithm is $O(nZ)$. Fig. 2 and Alg. 1 in the online supporting material illustrate and give the pseudo code of the above described algorithm. Its time and space complexities are summarized in the following lemma.

Lemma 3. *Given an RNA sequence S of length n, there is an algorithm which computes $L(1,n)$ in $O(nZ)$ time and $O(Z)$ space.*

3.3 Folding reconstruction

In addition to computing the optimal folding score of a given RNA sequence, it is often of interest to report at least one optimal folding. Some well known standard techniques for reporting an optimal folding apply trace-back procedures over the folding score matrix M, in $O(n^2)$ time [19]. In this section we show how to reconstruct one optimal folding, without exceeding the time and space complexities of our folding algorithm. Note that this is a challenging task, as the classical trace-back algorithm requires the availability of the full table M, while our algorithm stores only partial information.

Assume that the full table M is given, with annotated OCT subsequences. The basic recursive folding reconstruction algorithm [19] could be modified as follows to utilize the OCT subsequences:

1. For $j \leq i$, the only (optimal) folding of $S_{i,j}$ is the empty folding, and the algorithm halts without reporting any base-pair.
2. For $j > i$, if $S_{i,j}$ is an OCT, the algorithm reports the pair (i,j) and is called recursively on the subsequence $S_{i+1,j-1}$.
3. Otherwise, $S_{i,j}$ is partitionable, and therefore the algorithm finds an index $q \in Q_{i,j}$ for which $M[i,j] = M[i,q-1] + M[q,j]$ and then continues by computing an optimal folding of $S_{i,q-1}$ and of $S_{q,j}$. An optimal folding of $S_{i,q-1}$ is obtained by calling the algorithm recursively with the sub-instance $S_{i,q-1}$. As for computing an optimal folding of $S_{q,j}$, note that $S_{q,j}$ is an OCT, and consider the two cases, where either $q = j$ or $q < j$. If $q = j$, then there is no need for another recursive call. Otherwise $q < j$, and an

optimal folding of $S_{q,j}$ is obtained by first reporting the base-pair (q, j) and then calling the algorithm recursively with the sub-instance $S_{q+1,j-1}$.

Time complexity analysis of the trace-back algorithm on the full table M. When calling the above algorithm to compute the folding traceback of $S_{i,j}$, recursive calls with three different subsequences could be initiated at the top level: $S_{i-1,j-1}$, $S_{q+1,j-1}$ and $S_{i,q-1}$, thus index j is eliminated from further consideration as an end index. Therefore, each recursive call is performed with a different end index j, and altogether there are at most n recursive calls in the whole computation. For a recursive call in which the end index is j, at most $O\left(|Q_{1,j}|\right)$ operations are preformed in finding a $q \in Q_{i,j}$ for which $M[i,j] = M[i,q-1]+M[q,j]$. Since $\sum_{1 \le j \le n} |Q_{1,j}| \le Z$, the total running time is $O(Z)$.

We next turn to address the challenge of reconstructing an optimal folding from the sparse table M computed in Section 3.2. The above described algorithm cannot be applied directly in this case, due to the fact that when the algorithm needs to find $q \in Q_{i,j}$ for which $M[i,j] = M[i,q-1] + M[q,j]$, the values $M[i,j]$ and $M[i,q-1]$ may have been discarded from memory (while $M[q,j]$ is maintained in memory since $S_{q,j}$ is an OCT). In order to overcome this difficulty we adopt a similar approach as of the algorithm of Hirschberg [20], namely performing on-demand value re-computations of discarded entries. Thus, it remains to show how to recover such deleted entries.

Lemma 4. *Given the sparse table M that contains folding scores for OCT subsequences, there is an algorithm which recovers the set of entries $M[i,i+1], M[i,i+2], \ldots, M[i,j]$, for a pair of given indices i and j, in $O(Z)$ time.*

Proof. The entries of the form $M[i,j']$ which have been discarded from memory correspond to partitionable subsequences, where $L\left(i,j'\right) = L^p\left(i,j'\right)$, and thus may be recomputed based solely on Equation 3.4. Observe that this computation examines only entries of the form $M[i,q]$ for $q < j'$, and $M[q,j']$ for OCT subsequences $S_{q,j'}$. Re-computing the entries of the ith row from left to right guaranties that upon computing $M[i,j']$, all necessary values for the computation of $L^p\left(i,j'\right)$ are already stored in M. For each $i < j' \le j$, there are $O(|Q_{1,j'}|)$ operations performed along this computation, due to the consideration of branch points in the set $Q_{i,j'}$. As before, summing this expression over all $i < j' \le j$ accumulates to $O(Z)$. □

We next show that, throughout the full run of the algorithm, the process of restoring row entries is applied to $O(L)$ distinct start indices. Consider the case where the trace-back algorithm is applied on $S_{i,j}$ and assume that the set of entries $M[i,i+1], M[i,i+2], \ldots, M[i,j]$ was already previously restored. Note that a recursive call with a sub-instance of the form $S_{i,q-1}$ does not require the restoration of the entries $M[i,i+1], M[i,i+2], \ldots, M[i,q-1]$, as (by the assumption) they have already been restored and are maintained in M. The other two possible recursive calls with sub-instances of the form $S_{i+1,j-1}$ or $S_{q+1,j-1}$, do require re-computation of entries in M (in rows $i+1$ or $q+1$, correspondingly).

However, observe that each call of the latter kind is preceded by a detection of a base-pair. Since throughout the full run of the algorithm only L base pairs are detected, we get that the row entry recovery only needs to be executed L times (in addition to the recovery of $M[1,1], M[1,2], \ldots, M[1,n]$ during initialization). Thus, according to Lemma 4, the entry value recovery contributes an additional $O(LZ)$ factor to the total time complexity of the trace-back algorithm.

Furthermore, note that upon performing such a re-computation of an entry set, there is no need to further maintain the values in $M[i,i+1], M[i,i+2], \ldots, M[i,j]$ in the case where $S_{i,j}$ is co-terminus, nor to keep the values in $M[i,q], M[i,q+1], \ldots, M[i,j]$ in the case where $S_{i,j}$ is partitionable. This allows to discard these values from memory before the re-computation of the entry set for the corresponding sub-instance, guaranteeing that at each stage, at most n recovered entries are maintained in the sparse table M, in addition to the already existing OCT corresponding entries. Therefore, the space complexity of the trace-back algorithm remains $O(Z+n) = O(Z)$.

Alg. 2 in the online supporting material implements the efficient trace-back scheme.

Lemma 5. *There is an algorithm which, given the sparse table M that contains folding scores for all OCT subsequence of S, computes an optimal folding of S in $O(LZ)$ time and $O(Z)$ space.*

3.4 Extending the space reduction to Simultaneous Alignment with Folding

The goal of the SAF problem is to find a multiple sequence alignment and a common folding of the aligned sequences, which optimizes some score function. For simplicity, we assume the problem instance consists of two sequences. Similarly to single RNA strand folding algorithms, the basic dynamic programming algorithm for the SAF problem [12] computes the scores for all sub-instances of its input instance, and then combines these values to resolve the score of the full input instance. Given an instance of the problem - a pair of RNA sequences S and T, the algorithm maintains the scores of sub-instances $(S_{i,j}, T_{i',j'})$ in a four-dimensional table N (see Fig. 3). For $|S| = n$ and $|T| = m$, we depict N as an $n \times n$ "super table", in which each entry $N_{i,j}$ corresponds to an internal table of size $m \times m$, where the combined alignment-with-folding score of the sub-instance $(S_{i,j}, T_{i',j'})$ is stored in the entry $N_{i,j}[i',j']$. The time-complexity of the basic dynamic programming algorithm for the SAF problem is dictated by the need to compute all $O(n^2m^2)$ sub-instances, where each such computation involves the consideration of a set of $O(nm)$ competing branch point index pairs (*i.e.* all (q,q') such that $i < q \leq j$ and $i' < q' \leq j'$). This yields a total time complexity of $O(n^3m^3)$.

Recently, [15] extended the approach of [10] and applied it to speed up SAF by reducing the number of branch points that need to be considered in the main recursion for the SAF score computation. Similarly to the concept of OCT sequences, it is possible to define *OCT-aligned* sequence pairs, where the

pair $(S_{i,j}, T_{i',j'})$ is OCT-aligned if, in every optimal alignment-with-folding of $(S_{i,j}, T_{i',j'})$ the bases s_i and $t_{i'}$ are aligned to each other, the bases s_j and $t_{j'}$ are aligned to each other, and the common folding is co-terminus. Using this formulation to describe the results of [15], it was shown that it is sufficient to examine branch point pairs (q, q') such that the sequences $S_{q,j}$ and $T_{q',j'}$ are OCT-aligned, thus reducing the number of examined branch points and improving the running time of the algorithm. This extension yields an $O(nm\tilde{Z})$ time and $O(n^2m^2)$ space algorithm for the SAF problem, where $\tilde{Z}$ is the number of OCT-aligned sub-instances, and $nm \leq \tilde{Z} \leq n^2m^2$ (in practice, $\tilde{Z}$ is expected to be significantly smaller than $O(n^2m^2)$ [15]).

Applying an observation similar to Observation 2, an algorithm is suggested here which, upon the computation of entry $N_{i,j}[i', j']$, queries only those entries which correspond to OCT-aligned sub-instances, in addition to entries in rows i and $i+1$ of the "super table" N. The space complexity of SAF is thus reduced to $O(nm^2 + \tilde{Z})$ (w.l.o.g. $m \leq n$). In the extended version of this paper we describe in detail how to extend the space-reduction technique described in Section 3.2 to the four-dimensional matrix computed by the SAF algorithm [15]. An intuitive explanation can be found in Fig. 3.

Lemma 6. *There is an algorithm that computes the simultaneous alignment with folding of two RNA sequences S and T in $O(nm\tilde{Z})$ time and $O(nm^2 + \tilde{Z})$ space, where $n = |S|$, $m = |T|$, and w.l.o.g. $m \leq n$.*

4 Utilizing step characterization

In this section we take advantage of a step characterization of the single strand base-pairing maximization problem in order to improve the running time of the algorithms which compute it. Based on this approach, in Section 4.1 we describe an improvement to Alg. 1 which reduces its running time from $O(nZ)$ to $O\left(n^2 + LZ\right)$, and then in Section 4.2 we further reduce it to $O(LZ)$. Both algorithms have the same space complexity as Alg. 1, which is $O(Z)$.

Let $S_{i,j}$ be a subsequence of S. For $L' = L(i+1, j)$ or $L' = L(i, j-1)$, it is straightforward to show that $L' \leq L(i,j) \leq L' + 1$. Therefore, we get the following observation:

Observation 2 *For every $1 \leq k \leq n$, the sequence $L(k,k), L(k,k+1), \ldots, L(k,n)$, as well as the sequence $L(k,k), L(k-1,k), \ldots, L(1,k)$ are monotonically non-decreasing with unit steps in the range $0 - L$.*

The above observation implies a bound on Z, as follows.

Lemma 7. *The value of Z satisfies $n \leq Z \leq n(L+1)$.*

Proof. Every OCT subsequence $S_{i,j}$ satisfies that either $i = j$, or $L(i,j) > L(i+1, j)$. Hence, according to Observation 2, there are at most $L+1$ OCT subsequences that end with a given index j, and at therefore there are most $n(L+1)$ OCT subsequences of S. □

4.1 An $O(n^2 + LZ)$ Algorithm

Similarly to the previously presented technique for restricting the set of examined branch points in the computation of $L^p(i,j)$, we next show another dominance relation which can be utilized to further constrain the set of branch points examined in Equation 3.3.

Definition 5 (step sequence). *Call a subsequence $S_{i,j}$ a* step sequence *if in every optimal folding of $S_{i,j}$ the base i is paired.*

Observe that $S_{i,j}$ is a step sequence if and only if $q = i+1$ is not a branch point in any of the optimal foldings of $S_{i,j}$, i.e. $L(i,j) > L(i,i)+L(i+1,j) = L(i+1,j)$ (hence the term "step"). Also note that any OCT subsequence of length greater than 1 is a step sequence, though the opposite is not necessarily true. In the following Lemma we further restrict the branch points which need to be examined in a recursive computation of $L^p(i,j)$.

Lemma 8. *For any subsequence $S_{i,j}$ such that $j > i$, there is an optimal branch point q with respect to $S_{i,j}$ such that either $q = i+1$, or $S_{i,q-1}$ is a step sequence and $S_{q,j}$ is an OCT.*

Proof. If $q = i+1$ is an optimal branch point with respect to S, the lemma holds. Otherwise, $L^p(i,j) > L(i,i) + L(i+1,j) = L(i+1,j)$. According to Lemma 1, there is an optimal branch point $i+1 < q \leq j$ such that $S_{q,j}$ is an OCT. Therefore, $L(i,q-1)+L(q,j) = L^p(i,j) > L(i+1,j) \overset{\text{Obs. 1}}{\geq} L(i+1,q-1)+L(q,j)$. It follows that $L(i,q-1) > L(i+1,q-1)$, hence $S_{i,q-1}$ is a step sequence. □

Define the following subset of branch points with respect to $S_{i,j}$:

$$P_{i,j} = \{i+1\} \cup \{i+1 < q \leq j : S_{i,q-1} \text{ is a step sequence and } S_{q,j} \text{ is an OCT}\}.$$

The following equation restates Equation 3.4, based on Lemma 8.

$$L^p(i,j) = \max_{q \in P_{i,j}} \{L(i,q-1) + L(q,j)\}. \tag{4.1}$$

We next show a bottom-up algorithm that computes L according to Equations 3.1, 3.2, and 4.1. The presented algorithm is similar to Alg. 1, where a forward dynamic programming technique is applied in order to efficiently compute $L^p(i,j)$ (forward dynamic programming was also applied by Jansson et al. [18] to a related problem).

The new algorithm also scans and computes the entries of M in decreasing row index and increasing column index. It maintains the following invariant: upon reaching entry $M[i,j]$, the entry contains the value $L^p(i,j)$. Before computing row i in M, the entries $M[i,i-1]$ and $M[i,i]$ are initialized with zeros, and all entries $M[i,j]$ for $i < j \leq n$ are initialized with the corresponding values $M[i+1,j]$. This initialization is equivalent to examining the branch point

$q = i + 1$ in the computation of $L^p(i,j)$ according to equation 4.1 for all $j > i$ (the branching at $q = i+1$ is handled separately from other branch points in $P_{i,j}$ since it does not follow the step sequence-prefix-OCT-suffix rule as the rest of the group). Note that in this stage the invariant is sustained for the first entry in the row which is traversed by the algorithm - $M[i, i+1]$, since $P_{i,i+1} = \{i+1\}$.

Based on the invariant, upon reaching $M[i,j]$, the entry contains the value $L^p(i,j)$, and the value $L(i,j)$ can be computed by resolving the maximum between the current entry value and the value of $L^c(i,j)$, which is obtained from Equation 3.2. If $L^c(i,j) > L^p(i,j)$, $S_{i,j}$ is classified as an OCT. Then, if $M[i,j] > M[i+1,j]$, $S_{i,j}$ is classified as a step sequence, and the branch point $q = j+1$ is considered and forward-reflected to the computation of $L^p(i,j')$, for all $j' > j$ such that $S_{j+1,j'}$ is an OCT, by updating the value of $M[i,j']$ to be the maximum among its current value and that of $M[i,j] + M[j+1,j']$, thus accumulating the maximum according to Equation 4.1, and guaranteeing the maintenance of the invariant.

Alg. 5 in the online supporting material implements the forward dynamic programming approach described above, combined with the space-efficient approach described in Section 3.2. An illustration of its run is given in Fig. 4. The speedup obtained by this algorithm is due to the fact that branch points are examined by Equation 4.1 only if both the sequence prefix before the branch point is a step-sequence and its suffix, as from the branch point on, is an OCT. Note that, for each one of the Z OCT subsequences $S_{q,j}$ which are examined as suffices by Equation 4.1, Observation 2 shows that there are at most L sequences $S_{i,q-1}$ which may be corresponding step-sequence prefixes, and thus the total run-time contribution due to computation of values of the form $L^p(i,j)$ is $O(LZ)$. Since the table M has $O(n^2)$ entries, where for each entry $O(1)$ operations are performed in addition to the operations involved in the computations of $L^p(i,j)$, the total running time is $O(n^2 + LZ)$. The space complexity remains $O(Z)$, as the space complexity of Alg. 1.

Lemma 9. *Given an RNA sequence S of length n, there is an algorithm which computes $L(1,n)$ in $O(n^2 + LZ)$ time and $O(Z)$ space.*

4.2 An $O(LZ)$ Algorithm

In this section we further reduce the running time of the folding algorithm from $O(n^2 + LZ)$ to $O(LZ)$. We do so by applying a *step encoding* [21] to M, representing each of its rows by its $O(L)$ steps (see Fig. 5). Hence, in what follows we give corresponding step-encoding formulations, where a typical instance is composed of a suffix $S_{i,n}$ of S and a folding cardinality x, which will be denoted by the pair $(S_{i,n}, x)$. The goal is to compute the minimum index $i - 1 \leq j \leq n$ such that there is a folding of $S_{i,j}$ whose cardinality is x. The next definition gives the step-encoding equivalents of the entities $L(i,j), L^p(i,j)$, and $L^c(i,j)$.

Definition 6. *For $1 \leq i \leq n$, $1 \leq x$, and $\alpha \in \{\epsilon, p, c\}$ (where ϵ denotes the empty word), define $\beta^\alpha(i,x)$ to be the minimum index j such that $L^\alpha(i,j) \geq x$, or ∞ if there is no such j.*

Note the relation between the step-encoding formulation and the standard formulation, where $L(i,j)$ is the maximum x such that $\beta(i,x) \leq j$. Say that a sub-instance $(S_{i,n}, x)$ is a β-*OCT* if $\beta(i,x) = \beta^c(i,x) < \beta^p(i,x)$. The set $Y_{i,x}$ is the step-encoding equivalent of $P_{i,j}$:

$$Y_{i,x} = \{i+1\} \cup \left\{ i+1 < q \leq \beta(i+1, x-1) : \begin{array}{c} S_{i,q-1} \text{ is a step sequence, and} \\ (S_{q,n}, x - L(i,q-1)) \text{ is a } \beta\text{-OCT} \end{array} \right\}.$$

The following auxiliary function will be used in the computation of $\beta^c(i,x)$.

Definition 7. *For $\sigma \in \{A,C,G,U\}$ and $1 \leq r \leq n$, define $next(r,\sigma)$ to be the minimum index $r' > r$ such that $s_{r'} = \sigma$, or ∞ if there is no such index r'.*

We now convert Equations 3.1, 3.2 and 4.1 to their equivalent forms in the step encoding. For all $1 \leq i \leq n$ and $1 \leq x$:

$$\beta(i,x) = \min\{\beta^c(i,x), \beta^p(i,x)\}. \tag{4.2}$$

$$\beta^c(i,x) = next(\beta(i+1, x-1), \overline{s_i}). \tag{4.3}$$

$$\beta^p(i,x) = \min\left\{ \min_{q \in Y_{i,x}} \{\beta(q, x - L(i,q-1))\}, \beta^c(i,x) + 1 \right\}. \tag{4.4}$$

Formal proofs of the correctness of Equations 4.2 to 4.4, as well as the pseudocode of an algorithm that implements them, are included in the online supporting material. This algorithm, denoted Alg. 6, adopts a forward dynamic programming approach, similarly to that of Alg. 5. This allows for efficient computation of Equation 4.4, where the number of sub-instances, as well as the dimensions of the data structure that stores solutions for these sub-instances, is $O(Ln)$ (instead of $O(n^2)$).

Lemma 10. *Given an RNA sequence S of length n, there is an algorithm that computes $L(1,n)$ in $O(LZ)$ time and $O(Z)$ space.*

Acknowledgments. The work of Shay Zakov and Michal Ziv-Ukelson was partially supported by the Frankel Center for Computer Science at Ben Gurion University of the Negev. Rolf Backofen received funding from the German Research Foundation (DFG grant BA 2168/2-1 SPP 1258), and from the German Federal Ministry of Education and Research (BMBF grant 0313921 FRISYS).

References

1. Consortium, A.F.B., Backofen, R., Bernhart, S.H., Flamm, C., Fried, C., Fritzsch, G., Hackermuller, J., Hertel, J., Hofacker, I.L., Missal, K., Mosig, A., Prohaska, S.J., Rose, D., Stadler, P.F., Tanzer, A., Washietl, S., Will, S.: RNAs everywhere: genome-wide annotation of structured RNAs. Journal of Experimental Zoology Part B: Molecular and Developmental Evolution **308**(1) (2007) 1–25

2. Zuker, M.: Mfold web server for nucleic acid folding and hybridization prediction. Nucleic Acids Research (13) (2003) 3406–15
3. Hofacker, I.L.: Vienna RNA secondary structure server. Nucleic Acids Research (13) (2003) 3429–3431
4. Zuker, M.: Computer prediction of RNA structure. Methods Enzymol. **180** (1989) 262–288
5. Tinoco, I., Borer, P., Dengler, B., Levine, M., Uhlenbeck, O., Crothers, D., Gralla, J.: Improved estimation of secondary structure in ribonucleic acids. Nature New Biology **246** (1973) 40–41
6. Waterman, M., Smith, T.: RNA secondary structure: a complete mathematical analysis. Mathematical Biosciences **42** (1978) 257–266
7. Nussinov, R., Jacobson, A.B.: Fast algorithm for predicting the secondary structure of single-stranded RNA. PNAS **77**(11) (1980) 6309–6313
8. Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Research **9**(1) (1981) 133–148
9. Akutsu, T.: Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages. Journal of Combinatorial Optimization **3** (1999) 321–336
10. Wexler, Y., Zilberstein, C., Ziv-Ukelson, M.: A study of accessible motifs and RNA folding complexity. Journal of Computational Biology **14**(6) (2007) 856–872
11. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. 39th Symposium on the Theory of Computing (STOC). (2007) 590–598
12. Sankoff, D.: Simultaneous solution of the RNA folding, alignment and protosequence problems. SIAM Journal on Applied Mathematics **45**(5) (1985) 810–825
13. Mathews, D.H., Turner, D.H.: Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. Journal of Molecular Biology **317**(2) (2002) 191–203
14. Havgaard, J., Lyngso, R., Stormo, G., Gorodkin, J.: Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. Bioinformatics **21**(9) (2005) 1815–1824
15. Ziv-Ukelson, M., Gat-Viks, I., Wexler, Y., Shamir, R.: A faster algorithm for RNA co-folding. (2008) 174–185
16. Will, S., Reiche, K., Hofacker, I.L., Stadler, P.F., Backofen, R.: Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. PLOS Computational Biology **3**(4) (2007) e65
17. Gardner, P.P., Giegerich, R.: A comprehensive comparison of comparative RNA structure prediction approaches. BMC Bioinformatics **5** (2004) 140
18. Jansson, J., Ng, S.K., Sung, W.K., Willy, H.: A faster and more space-efficient algorithm for inferring arc-annotations of RNA sequences through alignment. Algorithmica **46**(2) (2006) 223–245
19. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis: Probabilistic models of proteins and nucleic acids. Cambridge University Press (1998)
20. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Communications of the ACM **18**(6) (1975) 341–343
21. Hirschberg, D.S.: Algorithms for the longest common subsequence problem. JACM **24** (1977) 664–675