Distance Profiles of Optimal RNA Foldings*

J. Liu[1], I. Duan[1], S. Santichaivekin[1], and R. Libeskind-Hadas[2]

[1] Harvey Mudd College, Claremont CA 91711, USA
[2] Claremont McKenna College, Claremont, CA 91711, USA

Abstract. Predicting the secondary structure of RNA is an important problem in molecular biology, providing insights into the function of non-coding RNAs and with broad applications in understanding disease, the development of new drugs, among others. Combinatorial algorithms for predicting RNA foldings can generate an exponentially large number of equally optimal foldings with respect to a given optimization criterion, making it difficult to determine how well any single folding represents the entire space. We provide efficient new algorithms for providing insights into this large space of optimal RNA foldings and a research software tool, toRNAdo, that implements these algorithms.

Keywords: RNA folding · Nussinov Algorithm · Zuker Algorithm

1 Introduction

The secondary structure of RNA plays an important role in gene regulation and expression. For this reason, the problem of predicting RNA secondary structure, also called a *folding* of the RNA, has received considerable attention over more than four decades. Among the best-known algorithms for predicting RNA foldings are those due to Nussinov [10] and Zuker [15].

The Nussinov Algorithm finds a folding of the RNA sequence that maximizes the number of paired complementary bases (A-U, G-C, and, optionally, the weaker G-U "wobble" pairs) with no *pseudoknots*. A pseudoknot is a pair of "crossing" matched pairs; that is, matched pairs with indices (i, j) and (k, ℓ) where $i < k < j < \ell$. The Zuker Algorithm finds a folding that minimizes the free energy of the folding, again assuming no pseudoknots. While the structure of the Zuker and Nussinov dynamic programs are similar, the standard Nussinov Algorithm has time complexity $O(n^3)$ (which can be improved to $O(n^3/\log n)$ [13]), while the Zuker Algorithm's more complex computation of free energies associated with different loop structures results in time complexity $O(n^4)$.

The restriction that foldings do not contain pseudoknots is required for the optimal substructure property exploited by the Nussinov and Zuker dynamic programs. In fact, finding minimum energy foldings with pseudoknots permitted is known to be computationally intractable (NP-complete) [7]. While comparatively rare [1], pseudoknots can arise and some efforts have been made to consider them, including complicated and slow combinatorial algorithms as well as machine learning approaches that have their own limitations [12]. As a result, the Zuker Algorithm remains a standard and has been implemented in a number of the most widely-used software tools for RNA folding [6, 8, 16].

However, in general, the number of equally optimal foldings (e.g., maximum number of matched base pairs in Nussinov's formulation or minimum free energy in Zuker's formulation) grows exponentially with the length of the RNA sequence. For example, Kiirala *et. al* performed experiments on 23S rRNA sequences with 117 sequences of average length 2726 and found that, on average, the number of Nussinov-optimal solutions was approximately 6×10^{130} [5].

An important and well-recognized challenge, therefore, is understanding the potentially large and diverse space of possible optimal foldings induced by a given optimization criterion. One approach is based on sampling a relatively small subset of optimal foldings and seeking to cluster them with

* This work was funded by the U.S. National Science Foundation under Grant Number IIS-1419739. The authors also thank Harvey Mudd College for use of lab resources where this work was conducted.

respect to a given distance metric [2, 9]. The medoids of those clusters can then be selected as good representatives of the space of optimal foldings [3]. An alternative approach to understanding the space of optimal foldings seeks to identify all of the base pair matches that are common to *all* foldings in a given set; for the sets of optimal foldings found by dynamic programming algorithms such as the Nussinov and Zuker algorithms, this can be done in polynomial-time [5].

Another approach for understanding the space of optimal foldings is based on computing statistics on the space with respect to a given distance metric. For example, the diameter of the space (the maximum distance between two foldings in the space) and the distribution of pairwise distances between all pairs of foldings are both potentially informative statistics and have been used successfully for other problems with large solution spaces [4, 11].

In many studies, a single optimal folding, R, is selected and it is therefore desirable to indicate the extent to which all other optimal foldings differ from R. A relatively simple measure is the maximum distance between R and any other optimal folding with respect to a given distance metric. A more general and informative statistic is the distribution of the distances between all optimal foldings and R. In this paper we show that both of these statistics can be computed exactly (i.e., without using sampling methods) in polynomial time for a family of distance metrics, in spite of the fact that the space of optimal foldings is, in general, exponentially large. In particular, we show how this can be done for both the maximum base pairing (Nussinov Algorithm) and the minimum free energy (Zuker Algorithm) objectives. Specifically, the contributions of this paper are:

1. An efficient exact algorithm for finding the maximum distance from a given optimal folding to all optimal foldings found by the Nussinov and Zuker Algorithms with respect to the well-known symmetric difference distance metric;
2. A generalization of this algorithm for computing the distribution of distances from a given optimal folding to all optimal foldings; and
3. A freely available software tool, toRNAdo , that implements these algorithms and demonstrates the efficacy of our approach. (`https://github.com/iisaduan/toRNAdo`)

2 Background

This section provides background required for the next section where our new algorithms are presented. Since there are a number of variations in the formulations of the Nussinov and Zuker Algorithms, and our new distance algorithms are formulation-dependent, we describe those algorithms here.

Throughout this paper, let S denote a given RNA string of length n, indexed from 1 to n. Let $S(i, j)$ denote the substring of S from indices i to j, inclusive. If $j < i$, we define $S(i, j)$ to be the empty string.

Let $\mathbf{comp}(i, j)$ be a Boolean-valued function that is True if the bases at indices i and j are complementary and False otherwise. Note that A-U and G-C are always assumed to be complementary and some formulations also allow G-U "wobble pairs" to be complementary.

A *folding* of string $S(i, j)$ is a set of ordered pairs $\{(i_1, j_1), \ldots, (i_m, j_m)\}$ such that $i \leq i_k < j_k \leq j$ for $1 \leq k \leq m$; each pair (i_k, j_k), called a *matched pair*, satisfies $\mathbf{comp}(i_k, j_k)$; and each index appears in at most one matched pair. We restrict our attention to foldings with no pseudoknots, that is, foldings where there exists no $1 \leq k, \ell \leq m$ such that $i_k < i_\ell < j_k < j_\ell$. Also note that when referring to an ordered pair of indices (i, j), we do not assume that they correspond to the indices of a matched pair in the RNA string unless explicitly stated.

2.1 The Nussinov Algorithm

The Nussinov Algorithm [10] seeks to maximize the total number of matched base pairs in a folding. Let $\mathbf{N}(i, j)$ denote the maximum number of matched base pairs in a folding of $S(i, j)$ for $1 \leq i, j \leq n$. We compute and store these values in a $n \times n$ dynamic programming table $\mathbf{N}$. When $j \leq i$, no matches are possible, so $N(i, j) = 0$. When $j \geq i + 1$, we consider two cases: either i is unpaired in an optimal folding of $S(i,j)$, or i is paired with a complementary base at some index k, $i < k \leq j$. Therefore,

$$\mathbf{N}(i,j) = \max \begin{cases} \mathbf{N}(i+1,j) & i \text{ unpaired} \\ \max\limits_{\substack{i<k\leq j, \\ \mathbf{comp}(i,k)}} 1 + \mathbf{N}(i+1,k-1) + \mathbf{N}(k+1,j) & i \text{ paired with } k \end{cases}$$

Note that each entry in this recurrence relies on entries with a greater value of i. Thus, we compute the entries in the dynamic programming table by decreasing values of i. We can reconstruct the optimal solutions by recording the optimal choices at each step. Specifically, the traceback information is stored in a table **opt**. Each entry $\mathbf{opt}(i,j)$ stores the set of all k such that $i < k \leq j$, and the base at index i is paired with the base at index k in some Nussinov-optimal folding for $S(i,j)$. Additionally, if i is unmatched in some Nussinov-optimal solution for $S(i,j)$, then i is included in $\mathbf{opt}(i,j)$ as well. The asymptotic time complexity of this algorithm, including the construction of the table **opt**, is $O(n^3)$, since there are $O(n^2)$ entries and computing each entry takes at most $O(n)$ steps.

2.2 The Zuker Algorithm

In contrast to the Nussinov Algorithm, which seeks to maximize the number of matched base pairs, the Zuker Algorithm seeks to minimize the free energy of a folding. There are a number of different formulations of the dynamic program for the Zuker Algorithm, all yielding the same set of optimal solutions. In this section we describe an adaptation of the formulation from [14] with the additional important property that when multiple cases are considered to compute the set of optimal foldings for $S(i,j)$, the sets of foldings for each of these cases are disjoint. This property is necessary when we compute the distribution of distances from one optimal folding to the set of all optimal foldings, in order to avoid overcounting solutions.

The Zuker Algorithm proceeds by identifying different types of RNA substructures, as each type of substructure may make a different contribution to the free energy. These structures are called *hairpin loops*, *stacking loops*, *internal loops*, and *multiloops*. Each of these loops, other than internal loops, is represented by a matched pair of indices (i,j), $i < j$, where the loop begins and ends; internal loops are parameterized by four indices (i,j,i',j'), $i < i' < j' < j$, to indicate that the loop has two ends, one at matched pair (i,j) and the other at matched pair (i',j'). The details of these loop topologies are not necessary for what follows, but the interested reader is referred to [14] as well as the Supplement (`shorturl.at/ELQX0`) which explicates the relationship between the original Zuker formulation and the equivalent one used here.

The energy functions of a hairpin loop (i,j), a stacking loop (i,j), and an internal loop (i,j,i',j') are given and denoted by $eH(i,j)$, $eS(i,j)$, and $eL(i,j,i',j')$, respectively [14]. Multiloops are computed as a function of three given positive constants a, b, and c; the free energy computation for multiloops is described below.

Now, we describe the formulation of the Zuker Algorithm. The equivalence of this formulation and the standard formulation [14] is easily verified. We calculate four $n \times n$ tables: **W**, **V**, **WM**, and **WM2**. Let $\mathbf{W}(i,j)$ denote the minimum energy of a folding of $S(i,j)$. Note that the minimum energy for a folding of the entire RNA string is given by $\mathbf{W}(1,n)$. Let $\mathbf{V}(i,j)$ denote the minimum energy of a folding of $S(i,j)$ such that (i,j) is a matched pair. The terms $\mathbf{WM}(i,j)$ and $\mathbf{WM2}(i,j)$ are used to compute the minimum energy of foldings that are part of a multiloop.

We first compute $\mathbf{V}(i,j)$. A positive integer m is used to specify the minimum size of a permitted loop. In the base case where $j - i < m$ or $\mathbf{comp}(i,j)$ is False, we have $\mathbf{V}(i,j) = \infty$ because no solution that matches the bases at indices i and j can exist. Otherwise, we take the minimum over the following four cases, corresponding to the four possible types of loops closed by (i,j) in an optimal folding:

$$\mathbf{V}(i,j) = \min \begin{cases} eH(i,j) & \text{Hairpin loop } (i,j) \\ eS(i,j) + \mathbf{V}(i+1,j-1) & \text{Stacking loop } (i,j) \\ \displaystyle\min_{\substack{i<i'<j'<j \\ (i',j')\neq(i+1,j-1)}} eL(i,j,i',j') + \mathbf{V}(i',j') & \text{Internal loop } (i,j,i',j') \\ a + \mathbf{WM2}(i+1,j-1) & \text{Multiloop} \end{cases}$$

Next, we compute $\mathbf{W}(i,j)$. When $j \leq i$, the optimal folding of $S(i,j)$ contains no matched base pairs, so $\mathbf{W}(i,j) = 0$. Otherwise, we consider the two cases, where i either is unpaired, or paired with k for some $i < k \leq j$. Thus,

$$\mathbf{W}(i,j) = \min \begin{cases} \mathbf{W}(i+1,j) & i \text{ is unpaired} \\ \displaystyle\min_{i<k\leq j} \mathbf{V}(i,k) + \mathbf{W}(k+1,j) & i \text{ is paired with } k \end{cases}$$

Next, we compute the minimum free energies due to multiloops. For $j \leq i$, we set $\mathbf{WM}(i,j) = \mathbf{WM2}(i,j) = \infty$ since no loop can exist on a substring of length 1 or 0. Otherwise,

$$\mathbf{WM}(i,j) = \min \begin{cases} \mathbf{WM}(i+1,j) + c & i \text{ is unpaired} \\ \displaystyle\min_{i<k\leq j} \mathbf{V}(i,k) + b + c(j-k) & i \text{ is paired with } k\text{, no pairings in } S(k+1,j) \\ \displaystyle\min_{i<k\leq j} \mathbf{V}(i,k) + \mathbf{WM}(k+1,j) + b & i \text{ is paired with } k\text{, more pairings in } S(k+1,j) \end{cases}$$

$$\mathbf{WM2}(i,j) = \min \begin{cases} \mathbf{WM2}(i+1,j) + c & i \text{ is unpaired} \\ \displaystyle\min_{i<k<j} \mathbf{V}(i,k) + \mathbf{WM}(k+1,j) + b & i \text{ is paired with } k \end{cases}$$

As in the Nussinov Algorithm, it is easy to record all optimal choices at each step so that we can reconstruct all the energy-minimizing solutions. We store this traceback information in a $4 \times n \times n$ table **opt**, where each entry $\mathbf{opt}(\mathbf{T},i,j)$ records the set of optimal choices for the entry $\mathbf{T}(i,j)$ where $\mathbf{T} \in \{\mathbf{W}, \mathbf{V}, \mathbf{WM}, \text{and } \mathbf{WM2}\}$; each optimal choice is a list of recursive calls in the form of $(\mathbf{T}', i', j')$. For example, suppose when computing the entry $\mathbf{WM2}(i,j)$, we find that there are three optimal choices (all yielding the same minimum value for that entry), which are: (1) not pairing i, (2) pairing i with k_1, and (3) pairing i with k_2. Then, for this example,

$$\begin{aligned} \mathbf{opt}(\mathbf{WM2},i,j) = \Big\{ & \{(\mathbf{WM2}, i+1, j)\}, \\ & \{(\mathbf{V}, i, k_1), (\mathbf{WM}, k_1+1, j)\}, \\ & \{(\mathbf{V}, i, k_2), (\mathbf{WM}, k_2+1, j)\} \Big\}. \end{aligned}$$

An optimal folding for $\mathbf{WM2}(i,j)$ can then be obtained by choosing any one of those three choices and continuing the traceback process from those choices. The resulting optimal folding is the set of matched pairs (k,ℓ) that appear in each tuple $(\mathbf{V},k,\ell)$ encountered in this traceback, since $\mathbf{V}(k,\ell)$ is invoked iff k is matched to ℓ.

As noted earlier and easily verified, this formulation has the property that each optimal choice (i.e. a list of recursive calls) recorded in $\mathbf{opt}(\mathbf{T},i,j)$ corresponds to a disjoint set of optimal solutions for $\mathbf{T}(i,j)$.

The asymptotic time complexity of the Zuker Algorithm, including the construction of the traceback table **opt**, is $O(n^4)$, since there are $O(n^2)$ entries to fill and computing each entry takes at most $O(n^2)$ time. The most costly computation occurs in computing the internal loop case for $V(i,j)$, which takes $O(n^2)$, while computing all the other entries takes $O(n)$ time.[3]

[3] A common heuristic bounds the interior loop size, which reduces the running time to $O(n^3)$.

2.3 Definitions and Notation

Recall that we use S to denote a given RNA string of length n indexed from 1 to n and $S(i,j)$ denotes the substring of S from indices i to j, inclusive. We use R to denote a given optimal folding (Nussinov- or Zuker-optimal). Let $R(i,j)$ denote the set of matched bases (i',j') in R where $i \le i' < j' \le j$.

For simplicity of exposition, we consider one distance metric on foldings, the *symmetric difference distance*, also known as the *BP distance* [2]: Given two foldings R_1 and R_2, the distance between the two foldings is equal to the number of matched pairs in R_1 that are not found in R_2 plus the number of matched pairs in R_2 that are not found in R_1. Henceforth, "distance" refers to this distance metric. We revisit distance metrics in Section 6.

The distribution of distances from a given optimal folding, R, to the set of all optimal foldings will be represented by a vector, where the i^{th} element of the vector indicates the number of optimal foldings at distance exactly i from the folding R under consideration. Throughout this work, vectors are understood to be over the non-negative integers. For a vector v, let $v[i]$ denote the element at index i. Given two vectors u and v, the *sum* of u and v, denoted $u+v$ is defined by $(u+v)[i] = u[i]+v[i]$. The *convolution* of u and v, denoted $u * v$ is defined by:

$$(u * v)[i] = \sum_{j=0}^{i} u[j]v[i-j]$$

The convolution of a sequence of vectors $V = (v_1, \ldots, v_n)$ is denoted $\mathop{\ast}_{v\in V} v$. (Note that this extension is well-defined since convolution is associative.) Given a vector v, σ^j, the j-place *shift* of v is defined by:

$$\sigma^j(v)[i] = \begin{cases} v[i-j] & : i \ge j \\ 0 & : i < j \end{cases}$$

Finally, we use the notation $\delta(X)$ to indicate the function that evaluates to 0 if the Boolean X is False and 1 otherwise.

3 Nussinov-Optimal Foldings

3.1 Nussinov Maximum Distance

We now describe a dynamic programming algorithm that, given an RNA sequence S and a Nussinov-optimal folding R of that sequence, calculates the maximum distance between R and all Nussinov-optimal foldings for S. A most distant Nussinov-optimal folding with respect to R can then be reconstructed through traceback.

The maximum distance algorithm begins by running the Nussinov Algorithm, which returns a traceback table **opt**. Recall from Section 2.1 that $\mathbf{opt}(i,j)$ stores the set of k, $i < k \le j$, such that the base at index i is paired with the base at index k in some optimal folding for the substring $S(i,j)$ and, if i is unmatched in some Nussinov-optimal solution for that substring, i is also included in this set. Additionally, given the Nussinov-optimal folding R, we can efficiently construct a table **numpairs**, where $\mathbf{numpairs}(i,j)$ is the number of matched pairs in $R(i,j)$ for $1 \le i,j \le n$. In other words, $\mathbf{numpairs}(i,j)$ is the number of pairs $(i',j') \in R$ such that $i \le i' < j' \le j$. We construct **numpairs**, as follows.

For the base cases, when $j \le i$, $\mathbf{numpairs}(i,j) = 0$ because any RNA substring of R of length 1 or less contains no matches. When $j \ge i+1$, we compute the entries in increasing lengths of the substring, that is, by increasing the value of $j-i$. These entries are computed as follows:

$$\mathbf{numpairs}(i,j) = \mathbf{numpairs}(i+1,j) + \begin{cases} 1 & \text{if } (i,k) \in R \text{ for some } i < k \le j \\ 0 & \text{otherwise} \end{cases}$$

The **numpairs** table allows us to efficiently compute the number of matches (i', j') in $R(i, j)$ that either use or "cross" an index k, meaning that $i \leq i' \leq k$ and $k \leq j' \leq j$. Specifically, the number of such crossings, denoted by **numcrossings**(i, j, k), can be calculated as:

$$\textbf{numcrossings}(i, j, k) = \textbf{numpairs}(i, j) - \textbf{numpairs}(i, k-1) - \textbf{numpairs}(k+1, j)$$

Now, we can calculate the maximum distance between R and all Nussinov-optimal foldings of S by computing a $n \times n$ table **maxdistance**. Each entry **maxdistance**(i, j) represents the maximum distance between $R(i, j)$, the set of matched bases (i', j') in R where $i \leq i' < j' \leq j$, and any Nussinov-optimal folding of $S(i, j)$, for all $1 \leq i, j \leq n$.

Base case: When $j \leq i$, the substring $S(i, j)$ has length 1 or 0, so both $R(i, j)$ and any Nussinov-optimal solution for $S(i, j)$ contain no matched bases. Therefore, for all $j \leq i$:

$$\textbf{maxdistance}(i, j) = 0$$

Recursive Case: When $j \geq i + 1$, we compute the entries **maxdistance**(i, j) in increasing values of $j - i$. For each entry, we consider two cases: (1) there exists k' such that $(i, k') \in R(i, j)$, that is, i is matched to k' in R for some $i < k' \leq j$, (2) there doesn't exist such k', which means i is either unmatched in R or matched with some ℓ where $\ell < i$ or $\ell > j$.

Case 1: i *is matched to* k' *in* $R(i, j)$ *for some* $i < k' \leq j$.

In this case, we must consider all the ways that i is matched or unmatched in the set of Nussinov-optimal solutions for $S(i, j)$. Recall that **opt**(i, j) stores the set of indices k, $i < k \leq j$, such that (i, k) is matched in some optimal solution for $S(i, j)$. Additionally, $i \in$ **opt**(i, j) if i is unmatched in some optimal solution for $S(i, j)$. We partition the set of optimal solutions for $S(i, j)$ into two parts, depending on whether i is matched in the solution or not. We consider each part separately and then we calculate **maxdistance**(i, j) by taking the maximum of the two parts.

In **part1**, we compute the maximum distance between $R(i, j)$ and all Nussinov-optimal solutions for $S(i, j)$ where i is matched to some k, $i < k \leq j$:

$$\begin{aligned}\textbf{part1} = \max_{\substack{k \in \textbf{opt}(i,j) \\ k \neq i}} \{ \; & \textbf{maxdistance}(i+1, k-1) + \textbf{maxdistance}(k+1, j) \\ & + \textbf{numcrossings}(i+1, j, k) + 2\delta(k \neq k')\}\end{aligned}$$

Note that for each k under consideration, the maximum distance for the substring $S(i, j)$ consists of the maximum distance for the substrings $S(i+1, k-1)$ and $S(k+1, j)$ (first line), the number of matched pairs in $R(i+1, j)$ that cross index k since these matched pairs are found in $R(i, j)$ but not in any optimal folding for $S(i, j)$ that matches i and k (second line), and the distances contributed by the matched pairs (i, k') and (i, k) (second line). If $k \neq k'$, then (i, k') and (i, k) are two different matched pairs and contribute 2 to the distance. If $k = k'$, then (i, k') and (i, k) are the same matched pair and make no contribution to the distance.

In **part2**, we compute the maximum distance between $R(i, j)$ and all Nussinov-optimal solutions for $S(i, j)$ where i is unmatched. If $i \notin$ **opt**(i, j), such optimal solutions do not exist, so we set **part2** $= -\infty$. If $i \in$ **opt**(i, j), then **part2** $=$ **maxdistance**$(i+1, j) + 1$. Here, the matched pair $(i, k') \in R(i, j)$ contributes 1 to the distance because $R(i, j)$ matches i with k', where $i < k' \leq j$, but the Nussinov-optimal solutions under consideration do not match i. Thus,

$$\textbf{part2} = \begin{cases} \textbf{maxdistance}(i+1, j) + 1 & \text{if unmatched } i \text{ is in } \textbf{opt}(i, j) \\ -\infty & \text{otherwise.} \end{cases}$$

Case 2: i is unmatched in $R(i,j)$.

Similar to the previous case, we partition the set of Nussinov-optimal solutions into two parts. In **part1**, we compute the maximum distance between $R(i,j)$ and the set of Nussinov-optimal solutions for $S(i,j)$ where i is matched to some k, $i < k \leq j$:

$$\mathbf{part1} = \max_{\substack{k \in \mathbf{opt}(i,j) \\ k \neq i}} \{ \mathbf{maxdistance}(i+1, k-1) + \mathbf{maxdistance}(k+1, j) + \mathbf{numcrossings}(i+1, j, k) + 1\}$$

The calculation is similar to the calculation for **part1** in Case 1 except for the last term: since i is unmatched in $R(i,j)$, any matching involving i in a Nussinov-optimal solution will contribute 1 to the distance. In **part2**, we again compute the maximum distance between $R(i,j)$ and all Nussinov-optimal solutions for $S(i,j)$ where i is unmatched:

$$\mathbf{part2} = \begin{cases} \mathbf{maxdistance}(i+1, j) & \text{if unmatched } i \text{ is in } \mathbf{opt}(i,j) \\ -\infty & \text{otherwise.} \end{cases}$$

This calculation differs from **part2** in Case 1 because neither $R(i,j)$ nor the Nussinov-optimal foldings under consideration match i.

Finally, in both Case 1 and Case 2, we find the maximum distance from all optimal solutions by

$$\mathbf{maxdistance}(i,j) = \max\{\mathbf{part1}, \mathbf{part2}\}$$

The asymptotic time complexity for the maximum distance algorithm is $O(n^3)$. The precomputation of **opt** takes time $O(n^3)$ and **numpairs** takes time $O(n^2)$. Filling the **maxdistance** DP table takes $O(n^3)$ as there are $O(n^2)$ cells to fill, and filling each cell takes $O(n)$ steps since we need to iterate through $O(n)$ possible values in $\mathbf{opt}(i,j)$.

3.2 Nussinov Distance Vector

The maximum distance algorithm can be extended to compute the *distance vector* $x = (x_0, x_1, \ldots, x_n)$ where x_i denotes the number of Nussinov-optimal foldings whose distance from R is exactly i. Note that $x[0] = 1$ since R is the only Nussinov-optimal folding with distance 0 from R, and that the maximum distance between any two Nussinov-optimal foldings is upper-bounded by n since a folding of string S of length n can have at most $\lfloor n/2 \rfloor$ matched pairs and thus two such foldings can differ in at most $2 \times \lfloor n/2 \rfloor \leq n$ matched pairs. Henceforth, we assume that all distance vectors have length $n+1$, with indices 0 through n, by appending zeros to the end if necessary. Recall that the notation $*$ denotes the convolution operator and $+$ denotes vector addition when its arguments are vectors.

We now describe the dynamic programming algorithm for computing $\mathbf{distanceVector}(i,j)$, the distance vector between $R(i,j)$ and the set of all Nussinov-optimal foldings for $S(i,j)$.

Base case: When $j \leq i$, the substring $S(i,j)$ has length 1 or 0, so there is only one Nussinov-optimal solution for $S(i,j)$, which is the trivial solution. This trivial solution is distance 0 from $R(i,j)$ since $R(i,j)$ also contains no matched bases. Therefore, for all $1 \leq i \leq n, j \leq i$:

$$\mathbf{distanceVector}(i,j) = (1, 0, 0, \ldots, 0)$$

Recursive case: When $j \geq i+1$, we compute the entries of **distanceVector** in increasing values of $j - i$, as we did for **maxdistance**. As before, we calculate the entries $\mathbf{distanceVector}(i,j)$ by cases.

Case 1: i is matched to k' in $R(i,j)$ for some $i < k' \leq j$.

In this case, $\mathbf{distanceVector}(i,j)$ is calculated by considering two parts in the partition of the set of Nussinov-optimal solutions. In **part1**, we consider the distribution of distances formed by $R(i,j)$ and the set of Nussinov-optimal solutions for $S(i,j)$ where i is matched to some $i < k \leq j$:

$$\mathbf{part1} = \sum_{\substack{k \in \mathbf{opt}(i,j) \\ k \neq i}} \sigma^{c_{i+1,j,k} + 2\delta(k \neq k')}(\mathbf{distanceVector}(i+1,k-1) * \mathbf{distanceVector}(k+1,j))$$

where $c_{i+1,j,k} = \mathbf{numcrossings}(i+1,j,k)$. Here, each term in the sum corresponds to the distance vector between $R(i,j)$ and the set of Nussinov-optimal solutions for $S(i,j)$ that contain the match (i,k). Consider a Nussinov-optimal solution for $S(i,j)$, denoted $Q(i,j)$, that contains the match (i,k) and has a distance of exactly d from $R(i,j)$. That distance, d, is made up of the distance between $Q(i+1,k-1)$ and $R(i+1,k-1)$, the distance between $Q(k+1,j)$ and $R(k+1,j)$, plus the distance contributed by matches in $Q(i,j)$ and $R(i,j)$ not contained in either substrings $S(i+1,k-1)$ and $S(k+1,j)$. To calculate how many Nussinov-optimal solutions are of this type, we take the convolution of the $\mathbf{distanceVector}(i+1,k-1)$ and $\mathbf{distanceVector}(k+1,j)$ and shift the result by the number of matches in $R(i+1,j)$ that cross k and any distance contributed by the matches (i,k) and (i,k').

In **part2**, we compute the distribution of distances between $R(i,j)$ and all Nussinov-optimal solutions for $S(i,j)$ where i is unmatched. If $i \notin \mathbf{opt}(i,j)$, such optimal solutions do not exist, so we set $\mathbf{part2} = \mathbf{0}$, the zero vector. Therefore, we have:

$$\mathbf{part2} = \begin{cases} \sigma^1(\mathbf{distanceVector}(i-1,j)) & \text{if unmatched } i \text{ is in } \mathbf{opt}(i,j) \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

The shift on $\mathbf{distanceVector}(i-1,j)$ accounts for the loss of the pairing $(i,k') \in R(i,j)$.

Case 2: i is unmatched in $R(i,j)$.

For the optimal solutions where i is matched to some k, we have:

$$\mathbf{part1} = \sum_{\substack{k \in \mathbf{opt}(i,j) \\ k \neq i}} \sigma^{c_{i+1,j,k}+1}(\mathbf{distanceVector}(i+1,k-1) * \mathbf{distanceVector}(k+1,j))$$

where $c_{i+1,j,k}$ is defined as in Case 1 above. This calculation is identical to the one in Case 1 except that we shift by 1 to account for the fact that i is unmatched in $R(i,j)$ but matched in the optimal solutions for $S(i,j)$ under consideration.

For the optimal solutions where i is unmatched, we have:

$$\mathbf{part2} = \begin{cases} \mathbf{distanceVector}(i+1,j) & \text{if unmatched } i \text{ is in } \mathbf{opt}(i,j) \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Finally, in both Case 1 and Case 2, we sum together **part1** and **part2** to account for all optimal solutions for $S(i,j)$:

$$\mathbf{distanceVector}(i,j) = \mathbf{part1} + \mathbf{part2}$$

The asymptotic time complexity of the distance vector algorithm is $O(n^4 \log n)$, since there are $O(n^2)$ cells to fill, filling each cell requires up to $O(n)$ convolutions, and each convolution takes $O(n \log n)$ time using Fast Fourier Transforms (FFTs).[4]

[4] Our implementation of the convolution operator in the accompanying toRNAdo software tool is not optimized and uses the naive $O(n^2)$ algorithm.

4 Zuker-Optimal Foldings

4.1 Zuker Maximum Distance

In this section, we describe a dynamic programming algorithm that computes the maximum distance from a given Zuker-optimal folding R to all other Zuker-optimal foldings for the given string S.

The distance algorithm begins by running the Zuker Algorithm which gives us the traceback table **opt**, where each entry $\mathbf{opt}(\mathbf{T}, i, j)$ records the set of optimal choices for obtaining the value at $\mathbf{T}(i, j)$, $\mathbf{T} \in \{\mathbf{W}, \mathbf{V}, \mathbf{WM}, \text{and } \mathbf{WM2}\}$. Recall that each optimal choice is a list of recursive calls in the form of $(\mathbf{T}', i', j')$ and the optimal choices in $\mathbf{opt}(\mathbf{T}, i, j)$ form a partition of the set of optimal solutions for $\mathbf{T}(i, j)$. We also precompute the table **numpairs** for R, defined and calculated in Section 3.1.

Now we can compute the maximum distance between R and all Zuker-optimal foldings of S by computing a $4 \times n \times n$ table **maxdistance**. Each entry $\mathbf{maxdistance}(\mathbf{T}, i, j)$ computes the maximum distance between $R(i, j)$ and any folding that can be constructed through a traceback of the table entry $\mathbf{T}(i, j)$, i.e. any optimal solution for $\mathbf{T}(i, j)$. The final answer, the maximum distance between R, and all Zuker-optimal solutions is found in $\mathbf{maxdistance}(\mathbf{W}, 1, n)$ since the set of solutions constructed by tracing back the entry $\mathbf{W}(1, n)$ is the set of all Zuker-optimal solutions.

Base Case: When $j \leq i$, the substring $S(i, j)$ has length 1 or 0. A traceback of the entry $\mathbf{T}(i, j)$ will yield the trivial solution for $\mathbf{T} = \mathbf{W}$, and no solution for $\mathbf{T} = \mathbf{V}, \mathbf{WM}, \mathbf{WM2}$. Thus,

$$\mathbf{maxdistance}(\mathbf{T}, i, j) = \begin{cases} 0 & \text{if } \mathbf{T} = \mathbf{W} \\ -\infty & \text{otherwise} \end{cases}$$

Recursive cases: We calculate the remaining entries of **maxdistance** in increasing order of $j - i$. For each (i, j) we calculate the cells in the order $T = \mathbf{V}, \mathbf{W}, \mathbf{WM}, \mathbf{WM2}$. We compute the entry $\mathbf{maxdistance}(\mathbf{T}, i, j)$ by considering the maximum distances between $R(i, j)$ and the different parts of the set of optimal solutions for $\mathbf{T}(i, j)$, partitioned by which optimal choice they correspond to in $\mathbf{opt}(\mathbf{T}, i, j)$. The maximum distance between $R(i, j)$ and the set of optimal solutions that make the choice e is given by:

$$\begin{aligned} \mathbf{choice}(e) = &\sum_{(\mathbf{T}', i', j') \in e} \mathbf{maxdistance}(\mathbf{T}', i', j') \\ &+ \mathbf{numpairs}(i, j) - \sum_{(\mathbf{T}', i', j') \in e} \mathbf{numpairs}(i', j') \\ &+ \lambda(\mathbf{T}, i, j) \end{aligned}$$

where

$$\lambda(\mathbf{T}, i, j) = \begin{cases} 0 & \text{if } \mathbf{T} \neq \mathbf{V} \\ -1 & \text{if } \mathbf{T} = \mathbf{V} \text{ and } (i, j) \in R(i, j) \\ 1 & \text{if } \mathbf{T} = \mathbf{V} \text{ and } (i, j) \notin R(i, j) \end{cases}$$

Consider a choice e that contains a set of recursive calls, each of the form $(\mathbf{T}', i', j')$. The optimal solutions for $\mathbf{T}(i, j)$ that use this choice e comprise optimal solutions for each of these entries $\mathbf{T}'(i', j')$. The first line sums up the maximum distances between $R(i', j')$ and optimal solutions for $\mathbf{T}'(i', j')$ for each recursive call $(\mathbf{T}', i', j') \in e$. The second line accounts for the matched pairs that are in $R(i, j)$ but not in the optimal solutions under consideration, analogous to the **numcrossings** term in the Nussinov distance algorithm. Finally, the term $\lambda(\mathbf{T}, i, j)$ adjusts the distance depending on whether the set of optimal solutions under consideration matches (i, j), which happens when $\mathbf{T} = \mathbf{V}$. If $\mathbf{T} = \mathbf{V}$ and (i, j) is also in $R(i, j)$, then we have overcounted the distance by 1 when we count (i, j) as a difference in the second line, so in this case $\lambda(e) = -1$. If $\mathbf{T} = \mathbf{V}$ but (i, j) is not in $R(i,j)$, then (i,j) is in all the optimal solutions under consideration but not in $R(i,j)$, and this difference has not been previously accounted for, so $\lambda(e) = 1$.

Finally, we maximize over all optimal choices:

$$\mathbf{maxdistance}(\mathbf{T}, i, j) = \max_{e \in \mathbf{opt}(\mathbf{T}, i, j)} \mathbf{choice}(e)$$

The asymptotic time complexity of this algorithm is $O(n^4)$, as there are $O(n^2)$ cells to fill, and each cell can take a maximum of $O(n^2)$ steps since there can be $O(n^2)$ optimal choices in $\mathbf{opt}(\mathbf{T}, i, j)$. (Recall that an interior loop (i, j, i', j') requires us to decide the optimal locations for both i' and j'. Therefore, there can be $O(n^2)$ optimal interior loops.)

4.2 Zuker Distance Vector

Finally, we extend the maximum distance algorithm for the Zuker Algorithm to compute the distance vector x where $x[i]$ denotes the number of Zuker-optimal foldings whose distance from R is exactly i. This extension is analogous to the one used to extend the Nussinov maximum distance algorithm to the Nussinov distance vector algorithm. As before, the maximum distance between any two Zuker-optimal foldings is n, the length of S, so we assume that all vectors have length $n+1$, with indices 0 through n. Let $\mathbf{distanceVector}(\mathbf{T}, i, j)$ denote the distance vector between $R(i,j)$ and all optimal solutions for $\mathbf{T}(i,j)$.

Base case: When $j \leq i$, the substring $S(i,j)$ has length 1 or 0, so there is only one folding for $S(i,j)$, the trivial folding. Thus we have one solution for $\mathbf{W}(i,j)$ at distance 0 from $R(i,j)$ and no solution for $\mathbf{T}(i,j)$ when $\mathbf{T} \neq \mathbf{W}$.

$$\mathbf{distanceVector}(\mathbf{T}, i, j) = \begin{cases} (1, 0, 0, \ldots, 0) & \text{if } \mathbf{T} = \mathbf{W} \\ (0, 0, 0, \ldots, 0) & \text{otherwise.} \end{cases}$$

Recursive cases: To calculate the remaining entries of **distanceVector**, we iterate through the dynamic programming table by increasing order of $j - i$. For each (i,j) we calculate the cells in the order $T = \mathbf{V}, \mathbf{W}, \mathbf{WM}, \mathbf{WM2}$. For each optimal choice $e \in \mathbf{opt}(\mathbf{T}, i, j)$, we find the distance vector between $R(i,j)$ and the set of optimal solutions that make the choice e, which is given by

$$\mathbf{choice}(e) = \sigma^{C_{\mathbf{T},i,j}(e) + \lambda(\mathbf{T},i,j)} \left(\mathop{\huge *}_{(\mathbf{T}', i', j') \in e} \mathbf{distanceVector}(\mathbf{T}', i', j') \right)$$

where

$$C_{\mathbf{T},i,j}(e) = \left(\mathbf{numpairs}(i,j) - \sum_{(\mathbf{T}', i', j') \in e} \mathbf{numpairs}(i', j') \right)$$

Finally, we add the vectors that result from different parts of the partition:

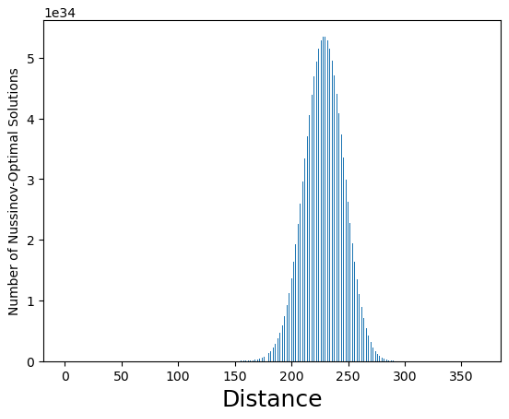
$$\mathbf{distanceVector}(\mathbf{T}, i, j) = \sum_{e \in \mathbf{opt}(\mathbf{T}, i, j)} \mathbf{choice}(e)$$

The asymptotic time complexity of this algorithm is $O(n^5 \log n)$; since there are $O(n^2)$ cells to fill, and filling each cell requires us to compute **choice** for each of the $O(n^2)$ optimal choices in $\mathbf{opt}(\mathbf{T}, i, j)$. Computing $\mathbf{choice}(e)$ takes only a constant number of convolutions because each e in any optimal solution makes at most two recursive calls, meaning we take the convolution of at most two terms. Therefore, the asymptotic time complexity of calculating $\mathbf{choice}(e)$ is limited by the time to compute a convolution, $O(n \log n)$.

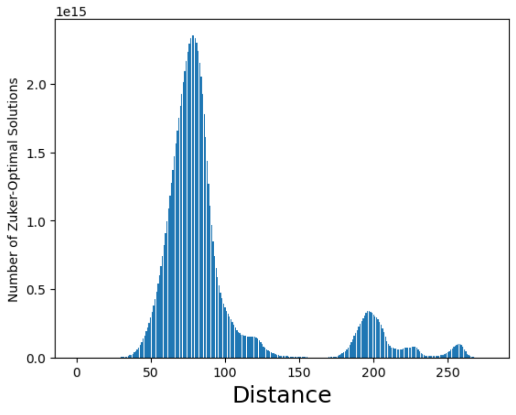
5 Algorithm Implementation

The algorithms described in the previous section have been implemented in a Python command line software tool called toRNAdo (`https://github.com/iisaduan/toRNAdo`). The software was tested for correctness by comparing results to those found by brute force for a large suite of small problem instances. This tool allows the user to specify the choice of Nussinov's or Zuker's Algorithm, provide an existing optimal folding or use a randomly selected optimal folding, among other command line options. The tool provides verbose text output as well as a histogram representing the distance vector.

As an example, we used this tool to fold a 517bp Arabidopsis thaliana (thale cress) partial 5S ribosomal RNA sequence (Accession: AF198208.1:1..517:rRNA). Under the Nussinov Algorithm, there were more than 10^{36} optimal foldings, and under the Zuker Algorithm, there were more than 10^{16} optimal foldings. Figure 1 shows the histograms representing the distance vectors for the two algorithms relative to randomly selected optimal solutions. These runs took 417 sec (Nussinov) and 4056 sec (Zuker) on a 3.2 GHz Apple M1 processor with 16GB RAM. Note that for the Nussinov Algorithm, the symmetric difference distances are always even since every pair of optimal foldings R_1 and R_2 has the same number of matched pairs and for any matched pair that is in $R_1 - R_2$ there must be a corresponding matched pair in $R_2 - R_1$. For the Zuker Algorithm, which minimizes free energy, two optimal solutions do not necessarily have the same number of matched pairs and thus distances are not necessarily even.



(a)



(b)

Fig. 1: Distance vector histograms for a 517bp Arabidopsis thaliana sequence relative to a randomly selected optimal solution for (a) Nussinov Algorithm (maximum distance: 366) and (b) Zuker Algorithm (maximum distance: 277).

6 Conclusion

In this paper we have shown that the maximum distance and, more generally, the distribution of distances between a given optimal RNA folding and the set of all optimal RNA foldings can be found in polynomial time, in spite of the fact that the number of optimal foldings can be exponentially large in the length of the given RNA sequence. This provides new insights into the space of optimal RNA foldings that can be particularly important when presenting and interpreting a folding for an RNA sequence.

Our results generalize in a number of ways. First, the "base" folding R needs not be optimal; our algorithms are able to find the maximum distance and the distance vector for non-optimal foldings. Second, we can trace back the dynamic programming tables to find an actual folding that is of maximum distance from R. (This feature is implemented in toRNAdo.) Third, in this paper we used the symmetric difference distance metric. The algorithmic results presented here are generalizable to other distance metrics that assign a positive (but not necessarily unit) cost when a matched pair occurs in one folding but not in another. However, there are many other possible distance metrics on foldings [9] and determining which metrics can be used with our approach is a direction for future research.

Finally, our work explores the distance from a given optimal folding to the set of all optimal foldings. Another approach would be computing the *diameter* of the space of foldings (the maximum distance between any two foldings) and, more generally, the distribution of distances between all pairs of foldings. This approach has been used to gain insights into the solutions spaces of other biological problems with exponentially large optimal solution spaces [4, 11]. It remains an open problem whether diameters and the distribution of pairwise distances can be computed efficiently for optimal RNA foldings.

References

1. Daniel P. Aalberts and Nathan O. Hodas. Asymmetry in RNA pseudoknots: observation and theory. *Nucleic Acids Research*, 33(7):2210–2214, 01 2005.
2. Phaedra Agius, Kristin P Bennett, and Michael Zuker. Comparing RNA secondary structures using a relaxed base-pair score. *RNA*, 16(5):865–878, 2010.
3. Ye Ding, Chi Yu Chan, and Charles E Lawrence. Clustering of rna secondary structures with application to messenger RNAs. *Journal of molecular biology*, 359(3):554–571, 2006.
4. Jordan Haack, Eli Zupke, Andrew Ramirez, Yi-Chieh Wu, and Ran Libeskind-Hadas. Computing the diameter of the space of maximum parsimony reconciliations in the duplication-transfer-loss model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(1):14–22, 2018.
5. Niko Kiirala, Leena Salmela, and Alexandru I Tomescu. Safe and complete algorithms for dynamic programming problems, with an application to RNA folding. In *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
6. Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Vienna RNA package 2.0. *Algorithms for molecular biology*, 6(1):1–14, 2011.
7. Rune B Lyngsø and Christian NS Pedersen. RNA pseudoknot prediction in energy-based models. *Journal of computational biology*, 7(3-4):409–427, 2000.
8. Nicholas R Markham and Michael Zuker. Unafold. In *Bioinformatics*, pages 3–31. Springer, 2008.
9. Vincent Moulton, Michael Zuker, Michael Steel, Robin Pointon, and David Penny. Metrics on rna secondary structures. *Journal of Computational Biology*, 7(1-2):277–292, 2000.
10. Ruth Nussinov, George Pieczenik, Jerrold R Griggs, and Daniel J Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied mathematics*, 35(1):68–82, 1978.
11. Santi Santichaivekin, Ross Mawhorter, and Ran Libeskind-Hadas. An efficient exact algorithm for computing all pairwise distances between reconciliations in the duplication-transfer-loss model. *BMC bioinformatics*, 20(20):1–11, 2019.
12. Jaswinder Singh, Jack Hanson, Kuldip Paliwal, and Yaoqi Zhou. RNA secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning. *Nature communications*, 10(1):1–13, 2019.
13. Balaji Venkatachalam, Dan Gusfield, and Yelena Frid. Faster algorithms for rna-folding using the four-russians method. *Algorithms for Molecular Biology*, 9(1):1–12, 2014.
14. Sebastian Will. Lecture notes from course 18.417, computational biology, MIT, fall 2011. `https://math.mit.edu/classes/18.417/Slides/rna-prediction-zuker.pdf`. Accessed: 2022-07-23.
15. Michael Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244(4900):48–52, 1989.
16. Michael Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic acids research*, 31(13):3406–3415, 2003.