RNA Structure Prediction Using Nussinov's Algorithm With Four Russians Speedup

Chan Tong Kou, Yves Chan, Andrew Taeyeon Kim, and Pranav Garg

***Abstract*—We implement and test two algorithms used to predict secondary structure of an RNA sequence. The four Russians technique used to speed up Nussinov's dynamic programming algorithm provides an improvement in time complexity but adds several preparation steps. Using an efficient implementation of this "Two-Vector" algorithm and the original method, we obtain structures for sequences of various sizes to see if the improvement in performance is evident for typical lengths. Our results indicate a two to five fold reduction in runtime for 500 - 5000 base sequences. The predicted structures, however, are very different from those known experimentally because the simple scoring model cannot capture all interactions between bases.**

I. Introduction

RNA molecules are crucial to some of the most fundamental life processes: gene regulation and protein synthesis. They are found in all domains of life, including viruses and many perform highly conserved functions. The secondary and tertiary structures of these molecules are central to their function, and often it is the folded structure that is conserved, not the actual sequence. Numerous methods have been developed to model and predict this folding process. Unlike proteins, where the native structure can be easily determined by methods such as X-ray crystallography and NMR, RNAs are much harder to work with. Not only are ribonucleases notoriously difficult to eliminate, the folded structures of RNA molecules are extremely sensitive to temperature and usually do not survive the processing phases of these experiments.

Fortunately, computational methods have become quite reliable in recent years. There are methods to predict RNA structures by sequence comparison with known structures. Others try to find *de novo* the most favourable configuration by minimising free energy. An early algorithm for *de novo* structure prediction was proposed by Nussinov and Jacobson [1]. This works by maximizing the number of non-crossing base pairs in a single RNA strand. For a sequence of length n, it takes $O(n^3)$ steps to deterministically find an optimal structure. In 2010, these algorithm was further improved by Frid and Gusfield [2] using the Four Russians technique where the solution to all possible subproblems (upto a certain size) are precomputed and stored. This "Two-Vector" algorithm is able to give the same solution as Nussinov but in $O(n^3/\log n)$ time. Parallelized versions of these algorithms, published by Venkatachalam, Gusfield and Frid [3], have a further n-fold speed-up each.

A theoretical improvement in time complexity is relevant for large datasets but sometimes due to the presence of large constants hidden in the big O notation, these algorithms actually take longer than simpler but "worse" algorithms. We test this effect on the Two-Vector and Nussinov algorithms using biological sequences of typical lengths. We also examine the resulting structures and see if they are similar to the known structures.

Submitted April 6, 2017.

II. Algorithms

A. *Nussinov's Algorithm*

Given an RNA sequence, this algorithm looks for a secondary structure which maximises the number of non-crossing complementary base pairs, *i.e*, `G–C` and `A–U`. The assumption of non-complementary base pairs is defined as follows: if in a sequence positions i and i' are paired respectively with j and j', then either $i < j < i' < j'$ (sequential pairs) or $i < i' < j' < j$ (nested pairs) — see Fig. 1. The algorithm proceeds by dynamic programming, calculating scores for larger sub-sequences by adding new residues on either side of a smaller sub-sequence.

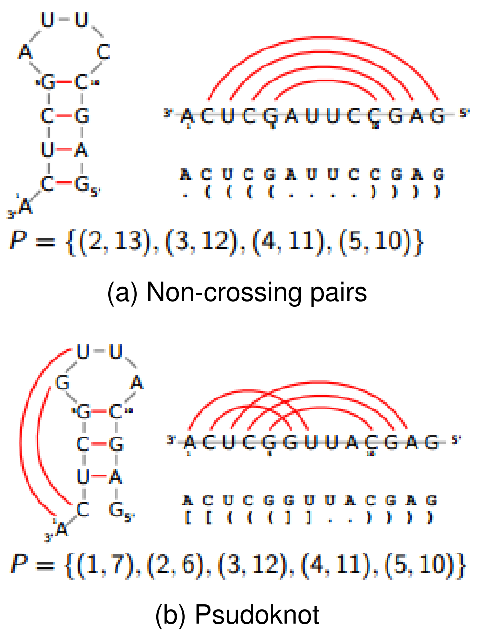
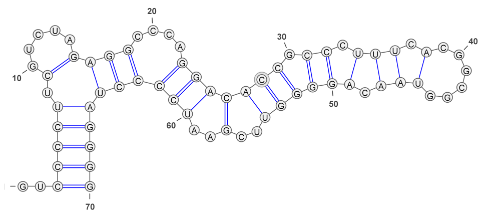
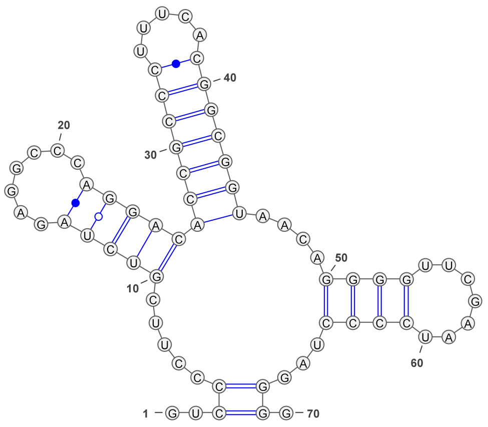

3' A C U C G A U U C C G A G 5'
A C U C G A U U C C G A G
. ((((. . . .))))
$P = \{(2,13),(3,12),(4,11),(5,10)\}$
(a) Non-crossing pairs
3' A C U C G G U U A C G A G 5'
A C U C G G U U A C G A G
[[((( ]] . .))))
$P = \{(1,7),(2,6),(3,12),(4,11),(5,10)\}$
(b) Psudoknot

Fig. 1. Example of non-crossing base pairing vs. psudoknots. P is the set of all pairs. Figure from [4]

Consider a two dimensional table D where $D(i,j)$ denotes the maximum number of non-crossing base pairings for the subsequence from position i to position j. After initialization, we only need to work only with the region $i < j$ of the table. Let length of the sequence be n.



(a)



(b)

Fig. 2. Stuctures of Glu-tRNA from *Escherichia coli* (PDB ID: `2DET`, RNA STRAND `PDB_00999`), as predicted by our folding algorithms (a) and known structure (b). Notice the presence of seven bulge loops in the predicted structure, four of size 1 at residues 10, 33, 37 and 48, two of size 2 from 54 to 60 and one of size 3 near residue 20. No such motifs exist in the known structure.

1) Initialization: For $-1 \leq j-i \leq M$:

$$D(i,j) = 0$$

where M is the minimum gap between paired nucleotides to allow space for bending the backbone. Usually $M = 4$.

2) Recurrence: For each i, j, $i < j$,

$$D(i,j) = \max \begin{cases} b(S(i),S(j)) + D(i+1,j-1), \\ \max\limits_{i \leq k < j} [D(i,k) + D(k+1,j)] \end{cases} \quad (1)$$

where $b(S(i),S(j))$ is the score for pairing the residues at positions i and j. It is 1 if $S(i)$ is a complement of $S(j)$, and 0 otherwise.

3) Termination and Traceback: The best score is $D(1,n)$. During recurrence, we should store how the best value was obtained in each step. This allows for traceback to reconstruct the pairings.

4) Complexity: Out of the two terms in the recurrence expression 1, the first term is calculated in constant time. The second term involves examining $O(n)$ values to find the maximum. Given there are $O(n^2)$ entries in the table, the total time complexity for this algorithm is $O(n^3)$.

B. Two-Vector

Dynamic programming algorithms typically proceed by filling out the answers to problems of increasing size in a table. The Four Russians Method is a tactic used to speed up calculations involved in filling up the table by pre-computing and storing answers to all possible pieces of the table. Imagine the table as being made up of blocks of some size. Each block is filled up using values stored in a boundary layer of cells, say top row and left column. If we can pre-compute what block values correspond to each set of boundary values we are able to speed up the table-filling process (provided we choose the block size appropriately).

This idea is ported over to our problem by speeding up the calculation of the second term in the recurrence (1) by breaking the i^{th} row and j^{th} column into vectors of a particular length q. If we pre-computed the answer for each possible row vector with each possible column vector, we could work vector by vector instead of cell by cell. This longer stride length gives a speed boost because we have far fewer terms to maximize over. To reduce the size of the lookup table though, we work with difference vectors which store the difference between consecutive cells. This condenses many possible vectors into one entry in the table. Thus, we write the values of cells in a row vector of length q starting at (i,l) as

$$D(i,l+p) = D(i,l) + V_p, \text{ where}$$
$$p \in \{0,...,q-1\} \text{ and } V_p = \sum_{i=0}^{p} v_i$$

and v is the horizontal difference vector in our lookup table. Entries in the horizontal difference vector are obtained as $D(i+i,j) - D(i,j)$ and an analogous vertical difference vector $\overline{v}$ is defined, whose values are $D(i,j+1) - D(i,j)$. The v vectors are binary because entries in a row can only increase by either 1 or 0 caused by at most one new matched pair introduced in each step. The same works for $\overline{v}$ except that it contains either 0 or -1.

1) Pre-computation: Let v and $\overline{v}$ be written in a decimal representation by treating the vectors as numbers and dropping negative signs.

For each $(v, \overline{v})$,

$$T(v,\overline{v}) = \max_{0 \leq k \leq q} \left(\sum_{i=0}^{k} v + \sum_{j=0}^{k} \overline{v} \right)$$

For traceback, we also store the $\arg\max$.

2) Initialization: This is same as before.

3) Recurrence: For each i, j, $i < j$,

Let $(j-i) = g \times q + s$ where $s < q$. g is number of blocks ("groups") and s is number of cells that don't fit into blocks ("stragglers").

$$D(i,j) = \max \begin{cases} b(S(i),S(j)) + D(i+1,j-1), \\ \max\limits_{i \leq k < s} [D(i,k) + D(k+1,j)], \\ \max\limits_{0 \leq k < g} T\left(v(i, i+s+k\cdot q)\right., \\ \qquad \left.\overline{v}(i+1+s+k\cdot q, j)\right) \end{cases} \quad (2)$$

where $b(S(i),S(j))$ is the score for pairing the residues at positions i and j. It is 1 if $S(i)$ is a complement of $S(j)$, and 0 otherwise.

4) Termination and Traceback: This is same as before.

5) Complexity: In the pre-computation step, the first entry in each difference vector is set to zero so we can omit those in the table. This gives us $O(2^{q-1})$ vectors to pre-compute, each taking $O(q)$ time. So pre-computing takes $O(q \times 2^{q-1})$ time In the recurrence step, we are now stepping q cells at a time, so the comparison will now be between $O(n/q)$ values. Since there are $O(n^2)$ cells, the recurrence takes $O(n^3/q)$ time.

The total complexity is $O(n^3/q) + O(q \times 2^{q-1})$. We can differentiate with respect to q to find the minimum for this expression. We get

$$2^{q-1} \times (q^4 - q^3 + q^2) = n^3$$

Keeping in mind that we are working with functional forms in O notation and not exact functions, we can drop the polynomial on the left to get that $q \sim \log(n)$ for optimal complexity which is $O(n^3/\log n)$.

III. Implementation and Testing

We implemented the two algorithms using C and C++. The source code is available on Github[1]. We included methods to compare runtimes, and to compare outputs of the two methods.

For testing, known RNA structures, as found in nature were obtained using the STRAND search page [5] in BPSEQ format. The sequences from these reference structures were input into our program. The resulting structures were visualized in VARNA [6]. In case of similar looking structures, simple file comparison could be used on the formatted outputs to look for differences.

We tested these algorithms on several sequences. The t-RNA sequences were of length ~ 70 and so their structures could be easily compared by eye against those from the database. Eight different sequences of varying lengths from 486 to 4381 were used for run-time analysis.

IV. Results

The two methods implemented produce exactly the same dynamic programming tables and therefore identical results, as expected.

The structure obtained for a t-RNA sequence is shown in figure 2. It is clear that there are large disparities between the natural observed structure and the predicted structure. This is a general feature of the outputs generated — the algorithm allows for some unrealistic features in the structures. The predicted structure clearly has more paired residues, which shows that it is not a flaw in the implementation.

Looking closer, we see that the predicted structure has long stem loops with little room for the backbone to bend in going from one to the other. It also has unaligned nucleotides to bulging outward from an aligned region. These bulge loops can actually be quite destabilizing for the structure in reality because they interrupt stacking of aromatic rings between adjacent residues. These features indicate that a better scoring model is needed to correctly penalise these features. The best way to account for such effects is to base the calculation on free energy contributions of each interaction between bases, as proposed by Zuker [7].

The runtime analysis is shown in table I and figures 3 and 4. The Two-Vector method outperforms the Nussinov algorithm in all cases. Runtime for short sequences was not available because it was too small to measure. The slopes in the log-log plots are close to 3 and clearly smaller for the Two-Vector algorithm which is consistent with our complexity analysis.

V. Conclusion

A variety of interactions contribute to the RNA secondary structure. The naïve scoring method used in Nussinov's algorithm is unable to take them all into account therefore giving structures that barely resemble those seen in nature. However, it was the first dynamic programming algorithm to address the problem of RNA secondary structure prediction. Many present day algorithms follow the same computation schemes and therefore any strategy to speed up this algorithm can be applied to present-day algorithms. We found that the Two-Vector method provides a two to five fold improvement in speed for sequences between 400 and 5000 bases long. This improvement could be valuable to Zuker's Algorithm which has to do many floating point operations in every iteration.

[1]https://github.com/Pranav-Garg/RNAfolding/tree/master/code

TABLE I
Runtime Data

STRAND ID	Length	Nussinov (s)	Two-Vector (s)
ASE_00102	486	0.09375	0.06250
CRW_00432	950	0.73438	0.29688
CRW_00253	1522	3.42188	1.34375
CRW_00330	1995	9.46875	3.31250
CRW_00522	2700	27.48438	6.76562
PDB_00796	3009	41.37500	9.35938
CRW_00524	3539	72.21875	17.17188
CRW_00528	4381	147.85938	32.54688

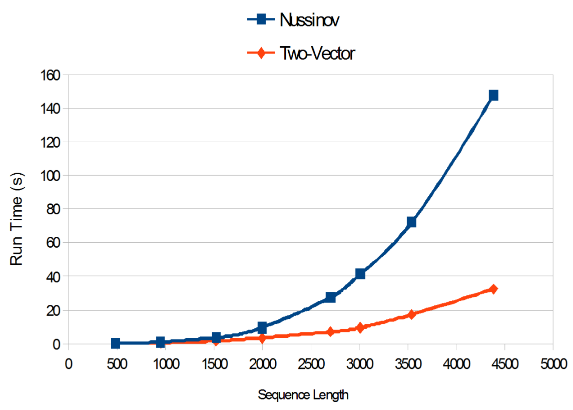


Fig. 3. Plot of runtime against sequence length.

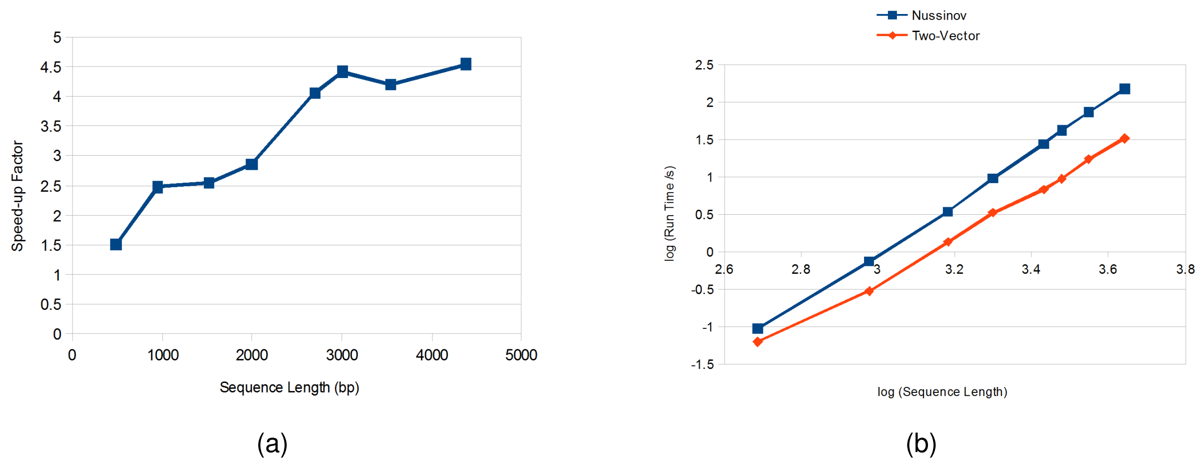


Fig. 4. Analysis of runtimes. The speed-up factor increases with length of sequence (a). The runtime scales roughly as the cube of the sequence length (b). The exponent is lower for the Two-Vector method. Slopes of best-fit lines (not shown) are 3.387 and 2.875 for Nussinov and Two-Vector algorithms respectively.

References

[1] R. Nussinov and A. B. Jacobson, "Fast algorithm for predicting the secondary structure of single-stranded RNA," *Proceedings of the National Academy of Sciences*, vol. 77, no. 11, pp. 6309–6313, 1980. [Online]. Available: http://www.pnas.org/content/77/11/6309.abstract

[2] Y. Frid and D. Gusfield, "A simple, practical and complete $O(n^3/\log(n))$-time algorithm for rna folding using the four-russians speedup," *Algorithms for Molecular Biology*, vol. 5, no. 1, p. 13, 2010. [Online]. Available: http://dx.doi.org/10.1186/1748-7188-5-13

[3] B. Venkatachalam, D. Gusfield, and Y. Frid, "Faster algorithms for RNA-folding using the Four-Russians method," *Algorithms for Molecular Biology*, vol. 9, no. 1, p. 5, 2014. [Online]. Available: http://dx.doi.org/10.1186/1748-7188-9-5

[4] S. Will, "RNA structure and RNA prediction," *MIT 18.417*, 2011. [Online]. Available: http://math.mit.edu/classes/18.417/Slides/rna-prediction-nussinov.pdf

[5] M. Andronescu, V. Bereg, H. H. Hoos, and A. Condon, "RNA STRAND: The RNA secondary structure and statistical analysis database," *BMC Bioinformatics*, vol. 9, no. 1, p. 340, 2008. [Online]. Available: http://www.rnasoft.ca/strand/search.php

[6] K. Darty, A. Denise, and Y. Ponty, "VARNA: Interactive drawing and editing of the RNA secondary structure." *Bioinformatics*, vol. 25, no. 15, pp. 1974–5, Aug. 2009. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00432548

[7] M. Zuker and P. Stiegler, "Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information," *Nucleic Acids Research*, vol. 9, no. 1, p. 133, 1981.