RNA Secondary Structure Prediction on Sets of Strands

Danielle Dees
Georgia Institute of Technology
College of Computing
danielle@cc.gatech.edu

Laura Slaybaugh
Rose-Hulman Institute of Technology
Computer Science
slaybalj@rose-hulman.edu

Anne Condon
University of British Columbia
Computer Science
condon@cs.ubc.ca

August 2, 2001

Abstract

There has been much study of RNA word designs for storage and retrieval of information, where a RNA word is defined as a short strand over the RNA alphabet A, U, C, and G. These RNA words are designed with specific properties in mind, in order to avoid complications that might otherwise arise due to the natural tendencies of this biological matter. One property that word designers must fight is the tendency of RNA to fold onto itself and form a secondary structure. In this paper, we study the secondary structure which occurs when RNA words are concatenated into strands with the ability to store larger quantities of information. Specifically, we consider the following problem: given a list of n pre-designed pairs of words, determine whether any of the 2^n combinations (i.e. strands formed from concatenating one word per pair) will fold into a secondary structure. We will present an efficient algorithm for this problem, which allows us to confirm or deny hypotheses about word design principles through careful testing. We also explore the following related problem: given a set S of words, does any word in S^* fold into a secondary structure? Using results from the theory of context free grammars, we show how we can obtain bounds on the length of concatenated words that need to be tested.

1 Introduction

RNA is a single strand of nucleotides; therefore, secondary structure observation is well-defined and observable in nature. The RNA nucleotides bond to other nucleotides based on their Watson-Crick pairs (AU, UA, CG, GC) or wobble pairs (UG, GU). It is assumed that no pseudo-knots exist in any of the secondary structures and that each base can only bond to one other base. Each pseudo-knot free structure can be viewed as a collection of stacked pairs and loops with exterior base pairs closing each loop. The types of loops are hairpin, internal, and multibranched, with a bulge loop being a particular instance of an internal loop. These structures are illustrated in Figure 1 below.

RNA secondary structure prediction can be done by determining the most stable configuration of the given nucleotides. The optimal structure will have the lowest free energy. Each of the various loops and stacked pairs will contribute a certain amount of energy to the secondary structure configuration. Our program takes in $2n$ pre-defined words, where a word is a sequence of RNA nucleotides of length l, and returns a "yes" or a "no" as to whether any of the possible strands will form a secondary structure. There are 2^n valid strand combinations of these words where we define a "valid strand" to be one in which we choose one of two words for each of the n positions. One application of this algorithm would be the transition of a binary string of length n, where each $1 < i \leq n$ $i \in \{0,1\}$, to an RNA strand. For each of the n binary bits, a word would be chosen for each of the n positions where $1 < i \leq n$ and $i \in \{w(i)$ or $\overline{w(i)}\}$.
Zuker [2] developed an dynamic programming algorithm which runs in time $O((l*n)^4)$ which predicts the secondary structure of a single RNA strand based on this energy model. This algorithm is quite useful to learn about a single sequence of RNA nucleotides; however, for our problem of finding the minimum energy over 2^n strands, this algorithm would be unreasonably slow. Our algorithm will determine the free energy over all 2^n structures in polynomial time by taking advantage of dynamic programming and modifying Zuker's recurrence relations for single strand folding.

2 Basic Algorithm

This algorithm will take in words will be labeled as $w(1), \overline{w(1)}, w(2), \overline{w(2)}, \ldots w(n), \overline{w(n)}$. Let $S = \{z_1 z_2 \ldots z_n | z_i \in \{w(i) \text{ or } \overline{w(i)}\}\}$. This algorithm will return a "yes" or a "no" as to whether any $s \in S$ form a secondary structure. A "yes" is defined as the program returning a negative value. This negative value represents the energy released from the strand as it fold into a thermodynamically favorable structure. Zuker's algorithm takes the lowest energy over all possible configurations that the j bases could create. To achieve a polynomial time algorithm from Zuker's original algorithm, we thought of the words as strands, and minimize over all possible configurations of the word concatenations. This required us to create a word#(x) function that would take in the position of a base, $1 \leq x \leq l * n$ and return a number between 1 and n inclusive. The word#(x) function determines the word number based on the word length l and parameter x. Our modifications to Zuker's algorithm include added parameters into the equations which specify whether S_i, the base at position i , is in $w(word\#(i))$ or $\overline{w(word\#(i))}$ based on the parameter $b_i \in \{T, B, E\}$. The value T represents $w(word\#(i))$, the value B represents $\overline{w(word\#(i))}$, and the value E represents the case where it does not matter to which word S_i belongs. The word specification is only T or B if the recurrence calls functions with a parameter that would be in the same word as a second parameter. Specifying the word verifies that the algorithm's minimal energy is calculated over the concatenation of whole words, and not just over parts of words.

2.1 W'_S

The energy of the optimal structure from S_1 to S_j where $j \leq n * l$ is:

$$W'_S(j) = W'_S(E, j) \tag{1}$$

$$W'_S(b_j, j) = \min \begin{cases} W'_S(x, j-1) & \begin{matrix} x = b_j \text{ if word\#}(j) == \text{word\#}(j-1) \\ x = E \text{ if word\#}(j) \neq \text{word\#}(j-1) \end{matrix} \\ \min\limits_{\substack{1 \leq i \leq j-1 \\ \text{word\#}(i) == \text{word\#}(i-1) \\ b_i \in \{B, T\}}} & V'_S(b_i, b_j, i, j) + W'_S(b_i, i-1) \\ \min\limits_{\substack{1 \leq i \leq j-1 \\ \text{word\#}(i) \neq \text{word\#}(i-1)}} & V'_S(E, b_j, i, j) + W'_S(E, i-l) \end{cases} \tag{2}$$

2.2 V'_S

The energy of a loop which is closed with the base pair S_i, S_j is:

$$V'_S(b_i, b_j, i, j) = \min \begin{cases} eH'_S(b_i, b_j, i, j) \\ eS'_S(b_i, b_j, i, j) + V'_S(x, y, i+1, j-1) \\ VBI'_S(b_i, b_j, i, j) \\ VM'_S(b_i, b_j, i, j \end{cases} \tag{3}$$

where x which corresponds to b_{i+1} and y which corresponds to b_{j-1} are determined by the following pseudo code:

```
x = E;
y = E;
if(word#(i) == word#(i+1)) then
        x = bi;
if(word#(j) == word#(j-1)) then
        y = bj;
if(word#(i+1) == word#(j-1)) then
```

```
        if(x == E and y == E) then
                x = b(1);
                y = b(1);
        else if(x == E) then
                x = y;
        else
                y = x;
```

2.3 VBI'_S

The energy of an internal loop which is closed by the pair S_i, S_j is:

$$VBI'_S(b_i, b_j, i, j) = \min \begin{cases} +\infty \text{ for} j < i + 4 \\ \min\limits_{i<i'<j'<j} eL(b_i, b_j, x, y, i, j, i', j') + V'_S(x, y, i', j') \end{cases} \tag{4}$$

where x which corresponds to $b_{i'}$ and y which corresponds to $b_{j'}$ are determined by the following pseudo code. The function $b()$ which takes in a number returns either $w(word\#(i))$ or $\overline{w(word\#(i))}$ when the important point is that two or more bases are in the same word and not which specific word.

```
x = E;
y = E;
if(word#(i) == word#(i')) then
        x = bi;
if(word#(j) == word#(j')) then
        y = bj;
if(word#(i') == word#(j')) then
        if(x == E and y == E) then
                x = b(1);
                y = b(1);
        else if(x == E) then
                x = y;
        else
                y = x;
```

2.4 VM'_S

The energy of a multibranched loop closed by the base pair S_i, S_j is:

$$VM'_S(b_i, b_j, i, j) = \min_{i+1<h\leq j-1} WM(w, y, i+1, h-1) + WM(x, z, h, j-1) + a \tag{5}$$

$$WM'_S(b_i, b_j, i, j) = \min \begin{cases} V'_S(b_i, b_j, i, j) + b \\ \min\limits_{i<h\leq j} WM'_S(b_i, y, i, h-1) + WM'_S(x, b_j, h, j) \end{cases} \tag{6}$$

where w which corresponds to b_{i+1},x which corresponds to b_h, y which corresponds to b_{h-1}, and z which corresponds to $b_{j-1}$are determined by the following pseudo code:

```
w = E;
x = E;
y = E;
z = E;
if(word#(i+1) == word#(i)) then
        w = bi;
else if(word#(i+1) == word#(h-1)) then
        w = b_1;
```

```
if(word#(h-1) == word#(i+1)) then
        x = w;
else if(word#(h-1) == word#(h)) then
       x = b(2);
if(word#(h) == word#(h-1)) then
        y = x;
else if(word#(h) == word#(j-1)) then
        y = b(3);
if(word#(j-1) == word#(h)) then
        z = y;
if(word#(j-1) == word#(j)) then
        if(z == b(2)) then
                b(2) = bj;
        else if(z == b(3)) then
                b(3) = bj;
        else
                z = bj;
```

3 Details of Base Cases

This algorithm is dependent on eH'_S, eS'_S, and eL'_S which are the free energy equations for a hairpin loop, a stacked pair, and an internal loop respectively. Additionally, we have a table *stack* that returns the energy of stacking the pairs $i, i+1$ on $j, j-1$ and functions that return the penalty for various structures.

3.1 eS'_S

eS'_S originally takes 2 values, i and j. $eS'_S(i,j)$ in turn calls $stack(i, i+1, j-1, j)$. When we call eS'_S we actually want $\min_{s \in S} eS(s,i,j)$. However, we do not need to test every possible $s \in S$ because the nucleotides relevant to eS'_S are only $i, i+1, j-1$, and j. Thus, what we actually do is create a set of strings R as defined below.

$R : \{r_1, \ldots, r_n | r_q[0] = s_i, r_q[1] = s_{i+1}, r_q[2] = s_{j-1}, r_q[3] = s_j | s \in S \text{ and } 1 \leq q \leq n\}$

Thus, we actually call $\min_{r \in R} stack(r[0], r[1], r[2], r[3])$.

3.2 eH'_S

eH'_S originally takes i and j as well. This returns the value $stack(i, i+1, j-1, j) + hPenalty(j-i-1)$. When we call eH'_S we actually want $\min_{s \in S} eH(s,i,j)$ Which simplifies to $\min_{r \in R}\{stack(r[0], r[1], r[2], r[3])\} + hPenalty(j-i-1)$.

3.3 eL'_S

eL'_S originally takes i, i', j', and j, where i and j close exterior end of the bulge or loop and i' and j' close the interior end of the bulge or loop. eL'_S returns the value $stack(i, i+1, j-1, j) + stack(i', i'+1, j'-1, j')$. For our case then,we first define a new set

$RL : \{rl_1, \ldots, rl_n | rl_q[0] = s_i, rl_q[1] = s_{i+1}, rl_q[2] = s_{i'}, \ldots, rl_q[7] = s_j | s \in S \text{ and } 1 \leq q \leq n\}$

eL'_S will return $\min_{rl \in RL}\{stack(rl[0], rl[1], rl[6], rl[7]) + stack(rl[2], rl[3], rl[4], rl[5])\}$.

4 Protein Encodings

While researching this question, contact was made with a pair of researchers at SUNY-Stony Brook, Barry Cohen and Steven Skiena. They had come across the same problem, but had answered it for a different reason. Their research was based on the coding of proteins with RNA. Each RNA strand uses codes of triplets, called "codons", that translate to one of twenty amino acids or a terminate signal. As there are 4^3 possible triplets, there is some redundancy in the codons for amino acids. Some may have just one codon, others have as many as six. When a protein is built from these amino acids, this results in an exponential number of possible RNA strands that would code for the protein. [3]

The program recieved from them reads two input files, one that indicates all the triplet codons and the amino acids they represent, and one that reads a string of amino acids used to build the protein. By rewriting these files to use a set of "codons" and "amino acids" of our own choosing, it is possible to test various word designs. The program is robust enough to handle sets of words with unequal cardinality and "codons" not of length three, however the length of all words across all sets must be equal.

5 Testing Considerations

This program was used to check words designs described in papers by Brenner, Frutos, Braich, and others [4, 5, 6]. Each of the word design strategies keep certain characteristics in mind, such as hamming distance, reverse hamming distance, GC content, three or four letter alphabet, and unique subwords. In each of the tests, the code indicates that no secondary structures would form. This prompted the following question: When designing DNA words, what considerations are most important and is there a ballpark "range" on acceptable values for them?

5.1 Testing Bounds

Before answering the above question, a testing procedure must be developed and acceptable limits set for testing. Therefore the following question must be answered: Given a set of words S, is there an acceptable limit n, for which one can state that no word in S^* will have a secondary structure if no word in S^n has secondary structure? If such a limit exists, one need only test up to n concatenations.

5.2 The Language of Secondary Structures

One characteristic noticable from examining the reccurance relation is the nested quality of secondary structures. This relation of $i < i' < j' < j$ seem to indicate a context freeness to the language of secondary structures. Indeed, several possible grammars have been proposed for limited secondary structures, however, it has also been adequately proven that the general language of secondary structures is beyond context free. [7]

What we would like to introduce now is a context free language for a limited set of secondary structures. This limited set, which we will define as having "minimal" secondary structure, consists of a strand of RNA in which a set of stacked pairs closes a substructure with positive energy. This substructure may consist of any number of internal loops, stacked pairs, multiloops, and hairpins, but all substructres in the strand must have positive energy and that an upper limit exists for what this energy can be. It is important to note that we are defining "substructure" to mean the configuration of the substring in the context of the entire string, not the configuration that the substring would take independent of the rest of the string. Additionally any number of unpaired nucleotides may be dangling on either side of the structure. Another important point to note is that the grammar is not meant to generate the minimal energy configuration for a strand, but rather a structure that has just enough negative energy to allow the structure to form. In reality it is entirely possible that the dangling nucleotides would bond with each other to extend the structure or that the entire strand would refold to create a more stable structre, however, this does not concern us as such a case would only make an already negative energy structure more negative. Below is an example grammar L that would generate a minimal secondary structure.

$$
\begin{array}{rcl}
W & \rightarrow & tV_0\bar{t}|tW|Wt \\
V_0 & \rightarrow & tV_1\bar{t}|tI_0\bar{t}|tM_0\bar{t} \\
V_1 & \rightarrow & tV_2\bar{t}|H_1|tI_1\bar{t}|tM_1\bar{t} \\
V_2 & \rightarrow & tV_3\bar{t}|H_2|tI_2\bar{t}|tM_2\bar{t} \\
V_3 & \rightarrow & H_3|tI_3\bar{t}|tM_3\bar{t} \\
H_3 & \rightarrow & tt_1t_2t_3\bar{t} \\
H_2 & \rightarrow & tt_1t_2\bar{t} \\
H_1 & \rightarrow & tt_1\bar{t} \\
I_1 & \rightarrow & tV_0|V_0t \\
I_2 & \rightarrow & tV_1|V_1t|tI_1|I_1t \\
I_3 & \rightarrow & tV_2|V_2t|tI_2|I_2t \\
M_0 & \rightarrow & WM_0WM_0 \\
M_1 & \rightarrow & WM_0WM_1|WM_1WM_0 \\
M_2 & \rightarrow & WM_1WM_1|WM_2WM_0|WM_0WM_2 \\
M_3 & \rightarrow & WM_0WM_3|WM_1WM_2|WM_2WM_1|WM_3WM_0 \\
WM_0 & \rightarrow & V_0WM_0|V_0 \\
WM_1 & \rightarrow & tWM_0|V_0WM_1|V_1WM_0|V_1|t \\
WM_2 & \rightarrow & tWM_1|V_0WM_2|V_1WM_1|V_2WM_0|V_2 \\
WM_3 & \rightarrow & tWM_2|V_0WM_3|V_1WM_2|V_2WM_1|V_3WM_0|V_3
\end{array}
$$

In the above grammar, $t \in \{A, C, G, U\}$ and $\bar{t}$ is the complement of t. The subscripts on each production keep track of the energy levels of the already built part of the strand. This grammar uses an extremely simplified energy model which keeps track of how many nucleotides are being closed and adds stacked pairs as necessary. It is merely meant to demonstrate that such a grammar can be created. Additionally, the number of productions necessary in the grammar will be dependant on the length of the words in S.

5.3 Pruning

Without loss of generality, consider a strand $s \in S^*$ that has secondary structure. It is known that $energy(s) < 0$, however, one can "prune" words from the beginning and ending of s until what remains is a single stable negative energy structure. It is then possible to "break" any stacked pairs at the end of the structure that are not needed to give the structure negative energy. This new strand s' is composed of a helix closing an undetermined structure of positive energy, s_{ss}. If $energy(s_{ss})$ is greater than the energy of the smallest hairpin creatable by removing whole words from s_{ss}, one can delete words from s_{ss}. This will create a bounds for maximum energy of a substructure. These pruning techniques are described in figures 1 and 2 below.

After using these pruning techniques, the result is a minimal secondary structure as described by the grammar L.

Thus to determine if any strings with secondary structures exist in S^*, one need only test whether $L \cap S^*$ is empty. Additionally, the pumping length, p, of the context free grammar for $L \cap S^*$ can be used to get the maximum bounds for testing.

Figure 1: Pruning words from the end of a strand

Figure 2: Pruning words from the center of a strand

6 Future Work

The most practical direction for future work would be to perform further tests on RNA word sets to determine which elements of word design are most important. These tests would determine which of the more common constraints of word design, GC percent constraint, Hamming constraint, reverse-complement constraint, the reverse constraint, and the free energy constraint [1], are most relevant to word design. This task would include determining ways to hold all but one constraint constant, while varying the other over a range of values. This is a very difficult and time consuming task, as all the variables are dependent on one another. The program that Barry Cohen and Steven Skiena developed could also be extended in many ways to test word design problems involving words of different lengths. In many current word designs, spacers of a smaller length than the words are placed between the words to reduce the secondary structure occurrences. A program which could test these spaces would be useful in many instances.

In addition to further testing, a fully developed grammar for RNA secondary structures could be developed. The grammar we present here is based on a simplified energy model and only generates a subset of all the strands which have a secondary structure. Extensions to this grammar would allow one to prove additional aspects of RNA secondary structure through the use of formal language theory.

References

[1] Marathe, A., A. Condon, and R. Corn. *On Combinatorial DNA word design.* DNA based Computers V, DIMACS Series, E. Winfree, D. Gifford Eds., AMS Press, 2000, 75-89.

[2] Lyngso, R. B., M. Zuker, and C. N. S. Pedersen. *An improved algorithm for RNA secondary structure prediction.* Tech. report BRICS-RS-99-15, Aarhus Univ., Datalogisk afdeling, May1999.

[3] Cohen, B. and S. Skiena. *Optimizing RNA Secondary Structure Over All Possible Encoding of a Given Protein.* Currents in Computational Molecular Biology, 2000, Universal Academy Press, Tokyo.

[4] Brenner, S., et. *In vitro cloning of complex mixtures of DNA on microbeads: Physical separation of deferentially expressed cDNAs.* Proceedings of the National Academy of Sciences of the United States of America 2000 February 15; 97(4): 1665-1670.

[5] Frutos, A. G., Q. Liu, A. J. Thiel, A. M. W. Sanner, Anne E. Condon, Lloyd M. Smith, and Robert M. Corn. *Demonstration of a Word Design Strategy for DNA Computing on Surfaces* Nucleic Acids Research, 25 4748 (1997).

[6] Braich, R. S., C. Johnson, P. W.K. Rothemund, D. Hwang, N. Chelyapov and L. M. Adleman. *Solution of a satisfiability problem on a gel-based DNA computer.* Proceedings of the 6th International Conference on DNA Computation in the Springer-Verlag Lecture Notes in Computer Science series.

[7] Searls, D. *Formal Language Theory and Biological Macromolecules.* manuscript submitted.