Some approaches to RNA secondary/tertiary structure prediction

Jon Malkin and Mukund Narasimhan
CSE 527, Fall Quarter

December 15, 2004

Abstract

As part of our course project, we investigated several approaches to RNA secondary and tertiary structure prediction. In this report, we describe some of the approaches specified by researchers to handle *knotted* tertiary RNA structures that standard Covariance Models do not handle. We also describe an algorithm that has been proposed to improve the (memory) efficiency of existing Covariance Model based approaches.

1 Introduction

The structure of non-coding RNA sequences is recognized as being important for determining the function of the molecule. Unfortunately, experimentally determining structures is a difficult process, typically involving a process such as X-ray crystallography or nuclear magnetic resonance spectroscopy. A much more appealing method is the use of computers to accurately and efficiently predict the form such RNA sequences will take. This paper reviews several published techniques for predicting how RNA sequences will form secondary and tertiary structures.

2 Graph Matching based algorithms

Several techniques covered rely on modeling RNA sequences with graphs. We present a very brief overview of graph notation and terminology before discussing the specific algorithms.

A graph G is an ordered pair (V, E), where V is a set (called the vertex set) and E (called the edge set) is a subset of $V \times V$. G is said to be undirected if $(a, b) \in E$ if and only if $(b, a) \in E$. If $a, b \in V$, we say that the vertices a and b are adjacent if $(a, b) \in E$. A matching is a set of edges $S \subseteq V$ such that every vertex is adjacent to at most one other vertex. More formally, for every $a \in V$, we must have $|\{b \in V | (a, b) \in S\}| \leq 1$.

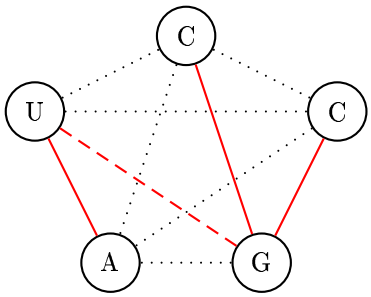

Figure 1: Sample graph for sequence AGCCU. Solid lines represent Watson-Crick pairs, a dashed line represents the "wobble pair," and a dotted line represents bases unlikely to pair.

If the edges of the graph are assigned weights using the weight function $w : E \to \mathbb{R}$, then the maximum weight matching (MWM) problem seeks a matching S such that

$$w(S) = \sum_{e \in S} w(e)$$

is maximized. As a special case, when $w(e) = 1$ for every $e \in E$, we get the maximum cardinality matching problem. The maximum weight matching problem can be solved in $O(|V|^3)$ time using $O(|V|^2)$ space for dense graphs using Gabow's algorithm [7]. This is a very simple algorithm and is commonly used for matching. However, the best known algorithm for matching runs in $O(\sqrt{|V|} \cdot |E|)$, which is a $\sqrt{|V|}$ factor better in the case of dense graphs, and can be considerably better for sparse graphs.

2.1 Graph Matching and RNA structure prediction

Suppose that $b_1 b_2 \ldots b_n$ is the RNA base sequence. We construct a graph $G = (V, E)$ whose vertices $V = \{b_1, b_2, \ldots, b_n\}$, and which contains all possible edges $E = \{(v_i, v_j) | \, 1 \leq i, j \leq n\}$. The idea is to assign a weight $w(v_i, v_j)$ based on biological factors to *score* each possible base pair. This could be a 0-1 score based on Watson-Crick pairs, or based on energy models, based on likelihoods, or other factors or combinations of factors.

Figure 1 shows the graph corresponding to the sequence AGCCU. Differently drawn edges signify differently scored edges (edges with different weights). Notice that there are no additional constraints such as the "nesting" constraints, and so pseudoknots are handled as naturally as nested base pairs. The goal is to find a maximum weight matching and use this matching to predict the structure.

We looked at three papers that used matching based techniques to determine RNA structure including [3, 4, 5]. The principal differences between the papers lie in the algorithms which assign weights to the edges of the graphs. Three techniques used to assign weights to the base pairs include:

1. Setting $w(v_i, v_j) = 1$ for all (v_i, v_j) which are Watson-Crick pairs or the wobble pair (this yields an algorithm to maximize the number of base pairs);

2. Picking $w(v_i, v_j)$ based on base pairs likelihoods (on a given seed set of hand-aligned sequences);

3. Picking $w(v_i, v_j)$ to be the mutual information between positions v_i and v_j (on the probability distribution induced by the hand-aligned sequences) yields an algorithm to maximize the covarying structure;

4. Picking $w(v_i, v_j)$ based on *helix plots* yields an algorithm that maximizes the occurrences of helices.

All the algorithms other than the last one, based on helix plots, yield relatively poor results. It is perhaps somewhat surprising that the algorithm based on mutual information perform poorly. The authors observe that this is especially the case in regions that are highly conserved, so the mutual information between pairs that are obviously good to humans is low.

The helix plot algorithm is the one that performed the best. In this scores are designed to pick out long helices whenever possible. The algorithm for picking scores is as follows.

1. Assign a small positive "good score" for Watson-Crick pairs and the G-U wobble pair, a larger negative "bad score" for other base pairs, and a still larger negative "paired gap" score for penalizing single strand deletions. Construct a matrix with these initial scores.

2. Scan the matrix for potential helices. Scores of base pairs comprising helices of less than some minimum length are changed to the "bad score". Scores of base pairs forming sufficiently long helices are increased by adding a bonus proportional to the length of the helix.

3. Repeat for each sequence in the alignment. Add individual score matrix to get a cumulative matrix.

Some obvious advantages of matching based methods include handling pseudoknots (and even base triples) in a very straightforward method. Further, these methods can even incorporate partial experimental information. The authors of [4] use the results of a nuclease protection assay and mutagenesis experiments to get the following kinds of information:

1. Bases known to be in single stranded regions;

2. Bases known to be paired with known partners;

3. Bases known to be paired but with unknown partners.

The first case can be simply handled by removing all edges adjacent to the vertex corresponding to the base (or assigning zero weight to these edges). The second can be handled by removing all edges not adjacent to both bases. The final case is handled by re-weighting the edges adjacent to these nodes such that any matching in which no edge adjacent to this vertex is present has less weight than any matching that has an edge adjacent to this vertex. Another advantage of matching based techniques is that approximation algorithms can be used to yield fast (suboptimal) results. This could be used, for example, in filtering sequences before applying a complete algorithm.

A general problem with all these methods is that they tend to generate many spurious base pairs. This is handled by assigning small negative weights to "wrong pairs," and also by filtering the output of these algorithms to discard edges that are obviously wrong.

2.2 Possible improvements to graph matching based algorithms

For improving accuracy, since all the "biological information" is encoded in the weights of the edges of the graph, the weights of the edges should not just be an accurate representation of the energies of the bonds that can be formed, but rather reflect the physical process. So, for example, base pairs that are relatively close (less than 4 apart) should have strongly negative weights. Further, like the helix plot algorithm, it is possible that accuracy could be improved by better modeling the helix formation.

The speed of the algorithms could be improved by reducing the number of edges considered by the algorithms. Since the speed of the algorithms is directly proportional to the number of edges, reducing the possible edges could result in a considerable improvement in speed. This could be achieved, for example, by using a simple algorithm to eliminate unlikely edges (such as eliminating all edges which do not occur in the training set). This could reduce the number of edges from $O(|V|^2)$ to $O(|V|)$.

3 Iterative Loop Matching

One problem many standard dynamic programming (DP) algorithms have is that in trying to maximize the number of base pairs, they employ methods that cannot detect pseudoknots. Graph matching can work reasonably well but, as mentioned earlier, produces many spurious bases. Ruan *et al.* proposed a method called Iterative Loop Matching [3] (ILM) as a way to help address these problems.

The basic problem is quite simple. Because dynamic programming relies on nested pairings, algorithms based on it typically cannot come up with RNA structures such as the one shown in Figure 2. ILM proposes a simple solution. First, run a regular DP algorithm, in this case one known as loop matching, to generate scores for each possible pairing. Then, take the loops with the highest

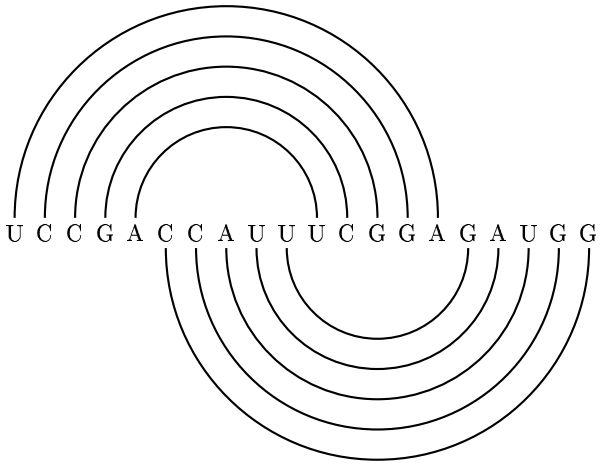

Figure 2: Example of a pseudoknot.

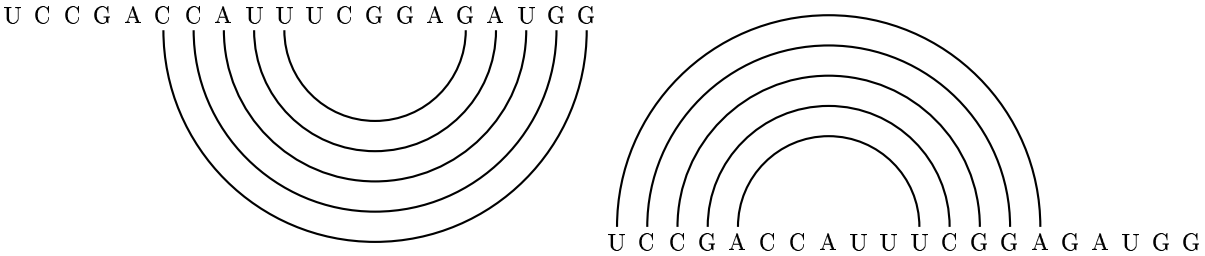

Figure 3: Example from the first iteration of ILM (left) and the second(right) on the same sequence.

confidence, typically taken as their score calculated by any of the methods used for graph matching, and effectively remove those nucleotides from the sequence. When implemented, the bases are simply no longer allowed to pair which helps keep distances between the remaining pairs accurate. The loop matching algorithm can then be iterated until no more pairs are detected or until the score of any helices detected falls below a threshold.

As an example, Figure 3 shows the output from both the first and second iterations of ILM, assuming the lower loop attains a higher score and is selected first.

The scoring method ultimately used for evaluating potential helices ended up being a mix of mutual information and helix plot scores, favoring the helix plots when they deemed there was insufficient data to calculate accurate

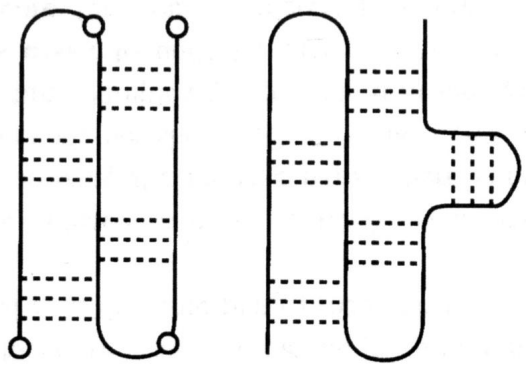

Figure 4: Examples of simple (left) and recursive (right) pseudoknots.

mutual information statistics. The authors evaluated ILM versus maximum weight matching and found that ILM is superior in terms of both selectivity and sensitivity. The authors compared their results to PKNOTS, a dynamic programming method, and found ILM's results were similar. But PKNOTS has the disadvantage of a much longer running time and larger memory usage; the time requirement is $O(n^3)$ and space only $O(n^2)$ make ILM an attractive algorithm.

One note is that the authors introduce the notion of a "virtual distance" between bases once bases have been removed. Pairing is disallowed if the distance is smaller than the virtual distance, similar to how there is typically a minimum hairpin length. It is not explicitly stated, but presumably this helps solve the problem of trying to pair bases on opposite edges of a long helix, where the molecule simply cannot realize such share turns.

4 A dynamic programming based algorithm

Despite the usual challenges of using dynamic programming for RNA structure prediction, it has been shown possible to use such algorithms to predict structures including pseudoknots. Akutsu [6] proposed a fairly simple method of doing so that improved on an earlier publication. The earlier method used a tree adjoining grammar and applied a DP-based parser — the author found that the parser was the essential part of the algorithm and that the tree adjoining grammar could be eliminated.

Using this purely DP-based method, the paper shows that simple pseudoknots can be evaluated in $O(n^4)$ time. While this is the same as in the tree adjoining grammar paper, the author asserts that this version is much easier to understand. A more complex structure such as a recursive pseudoknot can be evaluated in $O(n^5)$. Examples of both types of pseudoknots can be seen in

Figure 4. In both cases, space requirements are $O(n^3)$.

As most formulations of DP are similar in nature, a conceptual-level description should be sufficient. The key feature of this version is that instead of looking at pairs of bases for scores, triplets are scored instead. This allows for more combinations between nodes (in this case, assuming the bases are in order, first and second, second and third, and no connections; the case of first and third would be covered by other triples) and, as proven in the paper, is sufficient for the two cases of pseudoknots mentioned in this section.

As in the above cases, this algorithm can be used with a number of scoring modifications. Also, some extensions can be made; for instance, the energy function used when determining a folding score can depend on nearby bases, without increasing the complexity in terms of factors of n.

Because the time complexity for even the simple pseudoknot case is still rather high, the paper also presents a way to approximate the results in $O(n^{4-\delta})$ time with a score of at least $1-\epsilon$ of optimal, where δ and ϵ are parameters between 0 and 1.

Finally, the paper proves that RNA structure prediction with an arbitrary scoring function is NP-hard if more generalized pseudoknots than recursive pseudoknots are allowed. That is, allowing arbitrary planar graphs, RNA structure prediction will be NP-hard.

Unfortunately, the authors do not present any actual results for this method. They should be similar to the results in the tree adjoining grammar paper, but the implementation will be easier in this case.

5 Improving the memory efficiency of CM based algorithms

The last paper deal with Covariance Models (CM). A covariance model is a model for determining the secondary structure of an RNA base sequence by "matching" it to a Stochastic Context Free Grammar (SCFG). These models handle pairwise dependences in non-knotted structures.

The standard algorithm used for matching a base sequence to a covariance model uses the CKY, or Inside, algorithm. This algorithm guarantees the optimal structure (most probable parse in the SCFG) can be found in polynomial time. In fact, a slight extension of the algorithm can allow one to find the top N solutions (the top N most probable parses) in polynomial time. The following algorithm is the pseudocode for the CKY algorithm:

```
1: ∀ (state, start, len)      α_state[start, len] ← −∞
2: for all len ∈ 0 to n − 1 do
3:   for all start ∈ 0 to n − len do
4:     for all state ∈ 0 to M do
5:       Process Insert/Delete/Match States
6:       if state is a bifurcation (rule : state → child_0 child_1) then
7:         ∀split ∈ 1 to len − 1
```

8: $$\alpha_{\mathsf{state}}[\mathsf{start}, \mathsf{len}] \leftarrow \max \begin{cases} \alpha_{\mathsf{state}}[\mathsf{start}, \mathsf{len}] \\ \alpha_{\mathsf{child}_0}[\mathsf{start}, \mathsf{split}] + \alpha_{\mathsf{child}_1}[\mathsf{split}, \mathsf{len}] + \log p(rule) \end{cases}$$

9: **end if**
10: **end for**
11: **end for**
12: **end for**

The CKY algorithm is a dynamic programming algorithm that combines optimal parses for smaller sequences to find optimal parses for larger sequences, where the len variable in the algorithm represents the length of the sequence. In general, the number of bifurcation states is a small fixed constant, hence the total time complexity of the algorithm is $O(n^2 \cdot M + B \cdot n^3)$ where n is the length of the base sequence, M is the number of states in the CM and B is the number of bifurcation states in the CM. The space complexity is $O(n^2 M)$ (needed to store $\alpha_{\mathsf{state}}[\mathsf{start}, \mathsf{len}]$. Typically the number of states M is $\Theta(n)$, and hence the total time complexity is $O(n^3)$, and a total space complexity of $O(n^3)$.

Often, memory constraints are more of an issue than the time constraints. This is because once the memory requirement exceeds the memory available on the computer, little can be done. However, a (small) increase in the running time of the algorithm is typically not a problem. Another reason for optimizing memory performance is that a large memory footprint can cause the cache performance to deteriorate. For a sufficiently large memory footprint, the program could start swapping memory in and out of the disk, causing the program speed to be limited to the hard disk speed rather than the processor speed, dropping the performance from the order of 10^9 operations per second to 10^3 operations per second.

In [2], Eddy describes an idea to trade time for space. The algorithm given reduces the memory requirement from $O(n^3)$ to $O(n^2 \log n)$. The essential idea behind the algorithm is to recursively break the sequence to be parsed down into smaller subsequences, and simultaneously reduce the SCFG into smaller grammars and parse the smaller subsequence with the smaller grammar. At each step, rather than determining the complete parse of the sequence, only the "splitting point" is determined. As a result, the entire state need not be saved at each step.

6 A summary of some of the code we wrote

One of our project goals was to attempt to modify and enhance existing code for RNA structure prediction to either make it more efficient, or to add new functionality. We give below a summary of our efforts.

6.1 Modifications to Inferno

While traditionally the process of matching an RNA sequence against a CM is described as a dynamic programming algorithm, namely the Inside or CKY algorithm, it can also be viewed as an (exhaustive) search problem. When viewed

this way, we can then ask if traditional search algorithms can be applied to improve the efficiency of parsing. Inferno is a software package distributed by Sean Eddy which implements the CKY algorithm. The inferno code was modified to allow the dynamic programming step to call a upper bounding algorithm to determine if parts of the search space could be pruned away without detailed search, similar to the A^* algorithm. While the bounding algorithm was not completely implemented, the basic framework now exists for such experiments. It is expected that at some future time, this could be used with upper bounds based on HMM based techniques, or graph matching based techniques to see if the performance can be improved.

6.2 Modifications to RNAalifold

We attempted to modify RNAalifold, one of the programs mentioned in [1], as there were some specific recommendations for ways to improve the program's performance. While the code for folding the consensus of aligned sequences was better structured than the single sequence folding code, neither program seems to have been designed to allow for simple modification by anyone other than the author. Because of this, an attempt was made to simply use the program to compare the output on both seed and full sets of families from Rfam [8], but properly aligning the results from the different sets proved to require more time than was available. Results of this are still pending.

References

[1] P. P. Gardner and R. Giegerich, *A comprehensive comparison of comparative RNA structure prediction approaches*, BMC Bioinformatics 2004 5:140

[2] S. R. Eddy, *A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure*, BMC Bioinformatics 2002, 3:18

[3] J. Ruan, G. D. Stormo and W. Zhang, *An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots*, Bioinformatics 2004, 20(1)

[4] J. E. Tabaska, R. B. Cary, H. N. Gabow and G. D. Stormo, *An RNA folding method capable of identifying pseudoknots and base triples*, Bioinformatics 1998 14(8)

[5] Y. Ji, X. Xu and G. D. Stormo *A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences*, Bioinformatics 2004 20(10)

[6] T. Akutsu,*Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots*, Discrete Applied Mathematics 104 (2000) 45:62

[7] H. Gabow, *An efficient implementation of Edmonds's algorithm for maximum matchings on graphs*, J. ACM, 23:221-234, 1976

[8] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, S. Eddy, *Rfam: an RNA family database*, Nucleic Acids Research, 2003, 31, 1, 439-441