

How do RNA folding algorithms work?

Sean R. Eddy

Howard Hughes Medical Institute & Department of Genetics,
Washington University School of Medicine
4444 Forest Park Blvd., Box 8510
Saint Louis, Missouri 63108 USA
eddy@genetics.wustl.edu

October 5, 2004

Programs such as MFOLD and ViennaRNA are widely used to predict RNA secondary structures. How do these algorithms work? Why can't they predict RNA pseudoknots? How accurate are they, and will they get better?

The base-pairing of an RNA secondary structure is a sort of biological palindrome. A palindrome is a word or phrase that reads the same forwards and backwards – like “aibohphobia” (the irrational fear of palindromes). The base pairs of an RNA stem (say, GGACU paired to AGUCC) nest in a palindromic fashion, with complementary base pairings rather than identical letters. Of course, the pattern of base pairing in RNA secondary structures is not as simple as a true palindrome. Not all RNA residues are paired, and there are usually multiple stems. “Reengineer” isn't a true palindrome, but it's analogous to a four-base-pair RNA stem loop (reen/neer) with a two-residue loop (gi). “Sniffiness” isn't a palindrome either, but its letters can be fully paired into three nested “stems” (s/s, nif/fin, and es/se). Still, there is a fundamental relationship between RNA folding algorithms and algorithms for dealing with palindrome-like, nested pairwise interactions. Though RNA folding algorithms may look daunting, this is mostly just because of the detailed scoring systems that are used. We can strip that complexity away and lay bare the mechanics of the underlying folding algorithm. The problem of simply finding the structure with the maximum number of base pairs provides a clear example of how RNA folding algorithms work.

Base pair maximisation: a simple example.

To identify the structure with the maximum number of base pairs, our scoring system is just a +1 per base pair, 0 for anything else. Imagine looking at one contiguous subsequence from position i to position j in our complete sequence of length N , and calculating the score of the best structure for just that subsequence – that is, the maximum number of nested base pairs that the subsequence can form. The key is to recognize that this optimal score (call it $S(i, j)$) can be defined recursively

in terms of optimal scores of smaller subsequences. As shown in the top of Figure 1, there are only four possible ways that a structure of nested base pairs on $i..j$ can be constructed:

1. i, j are a base pair, added on to a structure for $i + 1..j - 1$.
2. i is unpaired, added on to a structure for $i + 1..j$.
3. j is unpaired, added on to a structure for $i..j - 1$.
4. i, j are paired, but not to each other; the structure for $i..j$ adds together substructures for two subsequences, $i..k$ and $k + 1..j$ (a bifurcation).

Consider the first case. If we add on a i, j base pair onto $i + 1..j - 1$, what is the score $S(i, j)$? Crucially, we know (from the definition of our scoring system) that the score we add for the base pair i, j is *independent* of any details of the optimal structure on $i + 1..j - 1$. Similarly, the optimal structure on $i + 1..j - 1$ and its score $S(i + 1, j - 1)$ are unaffected by whether i, j are base paired or not (or indeed, anything else that happens in the rest of the sequence). Therefore, $S(i, j)$ in case 1 is just $S(i + 1, j - 1)$ plus one, if i, j can base pair.

Similar independence arguments hold for the remaining three cases. In case 2, the optimal score $S(i + 1, j)$ is independent of the addition of an unpaired base i , so $S(i + 1, j) + 0$ is the score of the optimal structure on i, j conditional on i being unpaired. Case 3 is the same thing, but conditional on j being unpaired. In case 4, where we deal with putting two independent substructures together, the optimal score $S(i, k)$ is independent of anything going on in $k + 1..j$, and vice versa, so $S(i, k) + S(k + 1, j)$ is the score of the optimal structure on i, j conditional on i and j being paired but not to each other.

Since these are the only four possible cases, the optimal score $S(i, j)$ is just the maximum of the four possibilities. We've thus defined the optimal score $S(i, j)$ recursively as a function only of optimal scores of smaller subsequences; so we only need to remember these scores, not the combinatorial explosion of possible structures. Mathematically this recursion looks like:

$$S(i, j) = \max \begin{cases} S(i + 1, j - 1) + 1 & \text{[if } i, j \text{ base pair]} \\ S(i + 1, j) \\ S(i, j - 1) \\ \max_{i < k < j} S(i, k) + S(k + 1, j). \end{cases}$$

To run this recursion efficiently, we just need to make sure that whenever we try to compute an $S(i, j)$, we already have calculated the scores for smaller subsequences. This sets up a dynamic programming algorithm. We tabulate the scores $S(i, j)$ in a triangular matrix. We initialize on the diagonal; subsequences of length 0 or 1 have no base pairs, so $S(i, i) = S(i, i - 1) = 0$ (by convention, the $i, i - 1$ cells represent zero length sequences; the recursion must never access an empty matrix cell). Then we work outwards on larger and larger subsequences, until we reach the upper right corner, as shown in the bottom of Figure 1. This corner is $S(1, N)$, the score of the optimal structure for the complete sequence from $i = 1$ to $j = N$. Then, from that point, we

recover the optimal structure by tracing back the optimal path that got us into the upper corner, one step in the structure at a time.

Storing the $S(i, j)$ matrix requires memory proportional to N^2 , similar to what sequence alignment algorithms need. That's not a big deal these days; folding $N = 1000$ nt just needs a couple of megabytes. However, the innermost loop of having to find optimal potential bifurcation points k means that the folding algorithm requires time proportional to N^3 , a factor of N more time-intensive than sequence alignment. RNA folding calculations often require a hefty amount of computer power.

What RNA folding programs really score

Simple base pair maximisation is a poor scoring scheme for RNA structure prediction. It is more plausible that an RNA adopts a globally minimum energy structure, not the structure with the maximum number of base pairs. Therefore, the usual approach is to predict an overall free energy for a secondary structure, approximating this overall free energy as a sum of independent terms for different loops and base pairing interactions. The thermodynamic model has been developed in conjunction with the development of dynamic programming folding algorithms, so the independence assumptions in the thermodynamic model's terms have been made compatible with the independence assumptions needed for recursive dynamic programming algorithms to work. Energy minimisation algorithms become somewhat complex, with more detailed recursions which distinguish different lengths and types of loops, and which score base pairs according to nearest-neighbour stacking interactions with adjacent base pairs. Nonetheless, the mechanics of the algorithm are pretty much the same [1].

Why no pseudoknots?

In addition to nested stem-loop base pairing interactions, RNAs can also make non-nested base pairs between a loop of one stem and residues outside that stem: a so-called RNA pseudoknot. For example (continuing the palindromish analogy) in the phrase "no, a reiteration", no/on and are/era can be matched up with nested interactions, but the remaining letters it/ti can only be matched up if one makes a non-nested, pseudoknotted interaction, in which these connections cross the interactions made by the are/era stem.

The dynamic programming algorithm we discussed here can't deal with pseudoknots, because pseudoknots violate the recursive definition of the optimal score $S(i, j)$. For example, consider adding a pseudoknotted base pair i, k onto the subsequence $i + 1, j$, where the base pairing partner k lies somewhere *inside* the $i + 1..j$ interval. We can't just add a score for an i, k base pair onto $S(i + 1, j)$ to get $S(i, j)$, because we need to know that k is available to base pair with i ; maybe k was already paired with some other residue in the optimal substructure $S(i + 1, j)$. The algorithm hasn't kept track of this. The whole point of how the recursion works is that we only need to remember $S(i + 1, j)$, not any of the details of the combinatorial explosion of possible structures on the interval $i + 1, j - 1$, so the recursion is invalidated.

There are RNA folding algorithms that deal with pseudoknots, but each has at least one serious limitation of its own. There is an efficient algorithm (maximum weighted matching) that can guar-

antee optimal solutions, so long as one uses a simple base-pair dependent scoring system, not the more realistic stacking-dependent thermodynamic model. Very complex dynamic programming algorithms that guarantee optimal pseudoknotted solutions under the thermodynamic model are known, but they are too inefficient for most practical uses. Finally, different efficient heuristic approaches exist for searching for reasonable, though not provably optimal, pseudoknotted structures under the thermodynamic model.

Elegant, but still too often wrong.

In practice, benchmarks of prediction accuracy on single RNA sequences show that current RNA folding programs get about 50-70% of base pairs correct, on average. This is useful for many purposes, but not as good as we'd like.

Dynamic programming algorithms for RNA folding are guaranteed to give the *mathematically optimal* structure. Any lack of prediction accuracy is more the scoring system's problem than the algorithm's problem. The fundamental trouble seems to be that the thermodynamic model is only accurate to within maybe 5-10%, and a surprising number of alternative RNA structures lie within 5-10% of the predicted global energy minimum. It's therefore hard for a single sequence folding algorithm to resolve which of the plausible lowest-energy structures is correct. Much current research focuses on adding more biological information to the scoring model to further constrain RNA structure predictions. For example, several new approaches have attempted to combine thermodynamic scores with comparative sequence information, in order to predict consensus RNA structures for homologous RNA sequences. Nonetheless, for most of these approaches, the mechanics of the underlying dynamic programming algorithm remain essentially the same.

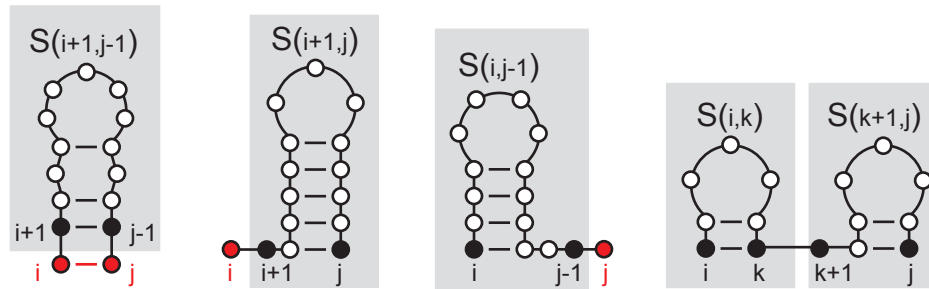
References

- [1] M. Zuker. Calculating nucleic acid secondary structure. *Curr. Opin. Struct. Biol.*, 10:303–310, 2000.

Figure 1 legend.

Top: The four cases examined by the dynamic programming recursion. Red dots mark the bases being added onto previously calculated optimal substructures (i, j pair, unpaired i , or unpaired j). Grey boxes are a reminder that the recursion tabulates the *score* of the smaller optimal substructures, not the structures themselves. Example substructures are shown in the grey boxes solely as examples. **Bottom:** The dynamic programming algorithm in operation, showing the matrix $S(i, j)$ for a sequence GGGAAUCC after initialisation, after the recursive fill, and after an optimal structure with three base pairs has been traced back.

Recursive definition of the best score for a subsequence i,j looks at four possibilities:



1. i,j pair;
2. i unpaired;
3. j unpaired;
4. bifurcation.

Dynamic programming algorithm for all subsequences i,j , from smallest to largest:

		$j \rightarrow$									
		G	G	G	A	A	A	U	C	C	
$i \downarrow$	G	0									
	G	0	0								
	G		0	0							
	A			0	0						
	A				0	0					
	A					0	0				
	U						0	0			
	C							0	0		
	C								0	0	

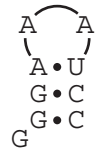
Initialisation;

		$j \rightarrow$									
		G	G	G	A	A	A	U	C	C	
$i \downarrow$	G	0	0	0	0	0	0	1	2	3	
	G	0	0	0	0	0	0	1	2	3	
	G		0	0	0	0	0	1	2	2	
	A			0	0	0	0	1	1	1	
	A				0	0	0	1	1	1	
	A					0	0	1	1	1	
	U						0	0	0	0	
	C							0	0	0	
	C								0	0	

Recursive fill;

		$j \rightarrow$									
		G	G	G	A	A	A	U	C	C	
$i \downarrow$	G	0	0	0	0	0	0	1	2	3	
	G	0	0	0	0	0	0	1	2	3	
	G		0	0	0	0	0	1	2	2	
	A			0	0	0	0	1	1	1	
	A				0	0	0	1	1	1	
	A					0	0	1	1	1	
	U						0	0	0	0	
	C							0	0	0	
	C								0	0	

Traceback;



Result.