RNA Macrostates: Theory and Tools

Michael Flynn

A Thesis
submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Physics

WILLIAMS COLLEGE
Williamstown, MA

Acknowledgments

This is the acknowledgements section.

Contents

List of Figures

Chapter 1

Introduction

1.1 RNA Secondary Structure Prediction

In the nucleus of a cell, RNA is constantly being synthesized by transcribing sections of DNA onto a portable chain of the nucleic acids adenine, gaunine, cytozine, and uracil. These RNAs serve several functions inside of the cell, including acting as:

mRNA: 'messanger RNA', which travel to ribosomes to serve as blueprints for proteins,

tRNA: 'transfer RNA', which bond to and transport amino acids to the ribosome to be formed into proteins,

rRNA: 'ribosomal RNA', which make up ribosomes,

ncRNAs 'non-coding RNAs', which are not involved in the manufacturing of proteins, instead being used as tools of the cell for tasks including the regulation of gene expression.

The last group, ncRNAs, comprises the majority of RNAs synthesized and have mostly unknown functions. It is unlikely that they just float around the cell uselessly, rather they are the tools the cell uses to build and run itself. It is also widely believed that the function of a strand of RNA is directly related to its structure. While DNA is composed of 2 seperate strands woven together in a double helix, RNA is most often found as a single strand that folds back onto itself. There are considered to be 3 main levels of structure of RNA. The first, the primary structure, is the sequence of nucleic acids that make up the strand of RNA i.e. 'GACCUUGGGGCCCC...'. The second, the secondary structure, is how these bases fold back on each other and form base pairs, most often of the Watson and Crick variety ('G-C', 'A-U', although sometimes 'G-U' is possible as well). The third, the tertiary structure, is how the structure bends on a larger scale as the stems and loops formed by the secondary structure interact with each other.

Primary structure can be readily observed by modern sequencing technology, however it is the secondary structure that determines the shape of the molecule, which is the most important when considering interactions with other biological molecules. The full specification of a secondary structure includes a list of every pair that is made by 2 of the nucleic bases in the primary structure making a bond. Finding the secondary structure of an RNA molecule is a different task than finding the primary structure of RNA because there is no single solution for one chemical chain. For an individual RNA molecule there are many valid pairings, in fact for a sequence of length n there are $O(1.8^n)$ secondary structures [4].

To find which of these structures the molecule is likely to assume in nature, we turn to statistical mechanics. We approximate the RNA molecule as an isolated system in contact with a thermal resevoir that is the cell, with each secondary structure as a state of that system. In such a system the probability of any state s is its boltzmann factor divided by the partition function:

$$P(s) = \frac{1}{Z}e^{-\beta E(s)}, \tag{1.1}$$

where $\beta = \frac{1}{RT}$, R is the gas constant, T is the temperature, and

$$Z = \sum_s e^{-\beta E(s)}. \tag{1.2}$$

Initially researchers were satisfied with presenting the MFE (minimum free energy) state as the state the molecule assumes in nature, after all this state is the most probable. However it has become clear that this analysis is unsufficient, as MFE structures still are not very probable. Statistical procedures, sampling the Boltzmann distribution, are a more modern tool for predicting the secondary structure found in nature.

1.2 The Energy Model of RNA

Computing the partition function and probabilities is impossible without an energy model for RNA, $E(s)$. Setting the energy of the single-stranded (no pair) state as $E = 0$, the energy model must accurately estimate an energy for a folded secondary structure. In chemical experiments, this can be measured by using a strand with an known secondary structure and finding it's ΔG by deducing it from the relative concentrations of single-stranded states to double-stranded states in solution (see UV melting section).

The first energy models developed by researchers awarded energy bonuses to pairs formed. One of the first papers by Nussinov and Jacobson [7] awarded the same amount of points to A-U and G-C pairs and found the MFE state, so the algorithm reduced to finding the legal folding with the most base pairs. After, it was discovered exprimentally that G-C pairs are more stable than A-U pairs. Because of this, in further iterations the energy would be determined by counting hydrogen bonds of cannonically paired bases, assigning each -1 kcal/mol of free energy. This would mean that GC pairs are given -3 kcals/mol, AU and GU pairs are both given -2.

The algorithm developed for this model minimized the free energy. Both this algorithm and the previous one had elementary dynamic programming solutions. They were useful models to use as a baseline, however, even the second iteration was not very accurate, on average only 20.5% of known base pairs are correctly predicted. Later energy models would use it as a control for the hypothesis that they increased secondary structure prediction accuracy [5].

Indeed, much improvement was made over the hydrogen bond model by expanding it to include what is now called the Nearest Neighbor model. Experiments made it clear that energy of an RNA folding is not just linearly dependent on the bonds that are made. There are significant interaction effects between nearby bases and bonds, this is called 'sequence dependence', and there are polymer physics based energy terms that scale logarithmically with the length of a loop (this can be thought of as the energy it takes to bend the strand). To handle these interaction effects in a model, we approximate that they are contained within loop regions. We divide our structure into its loops and compute the energy of each loop, with a seperate energy model for each.

In Figure 1-1, these loops are described. In brief, there is a different category and energy model for each type of loop with different numbers of paired and unpaired bases. For example a stack loop has 2 pairs and no unpaired bases. An internal loop has 2 pairs and arbitrary unpaired bases. A hairpin has 1 pair and arbitrary unpaired bases. Finally a multi-loop has at least 3 pairs and arbitrary unpaired bases.

There is a 5th type of loop, called a pseudoknot. A pseudoknot consists of 2 "crossing" pairs. This breaks the standard topology of the loop diagrams and makes computation of the partition function provably NP-hard [TODO: cite]. For this reason, pseudoknots are excluded from consideration. This is not ideal because pseudoknots do appear in nature, notably in ribosomal RNA (Mathews et al 1999). However, some of the results of this reseach might make the computation of a certain restricted set

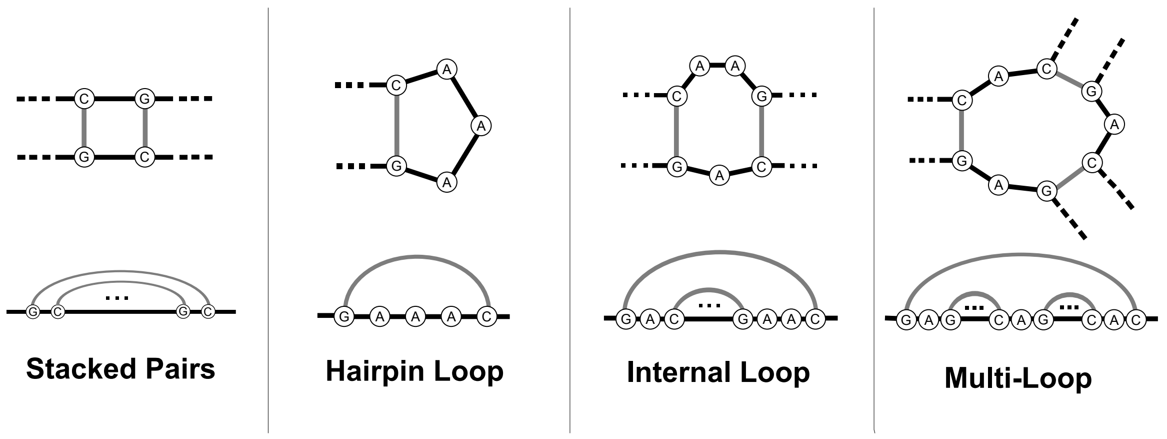


Figure 1-1: The 4 types of loops, black connections are bonds in the RNA backbone, grey connections are hydrogen bonds between bases. The types are: *Stacked Pairs*, adjacent pairs of bonded bases, *Hairpin Loops*, one bonding pair closing off a turn in the RNA backbone, *Internal Loops*, which can range from bulges to long loops, connecting to pairs with 2 chains of unpaired bases, and *Multi-Loops*, which connect 3 or more pairs.

of pseduoknots much more efficient.

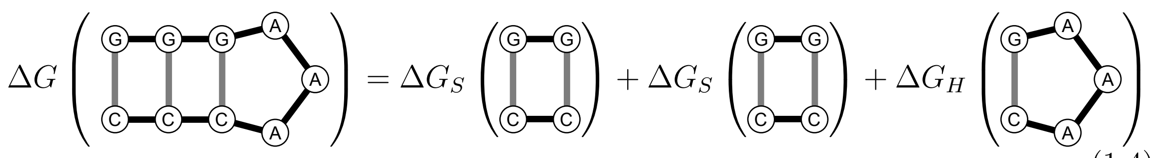
[TODO: pseudoknot figure]

1.2.1 Loop Parameters

One might wonder what the parameters, the terms, of this model would be. The model is simply that the energy of a structure, is the sum of it's loops:

$$E(s) = \sum_{l \in \text{loops}} E(l). \tag{1.3}$$

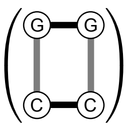
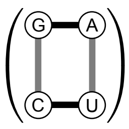
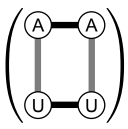
For example for a very predictably folding strand 'GGGAAACCC' (G's really want to pair with C's and A's resist pairing), we can decompose the energy as such:



(1.4)

Where ΔG_S is the function for energy of a stack loop and ΔG_H is the function that gives you the energy of a hairpin loop.

A model of loop energy should capture as much non-linearity as possible. For stack loops, the legal pairings limit the possible loops types. In fact, instead of worrying about how to model the loop's energy using polymer physics, we can just exhaust the possible stack loops, giving each a seperate parameter, i.e.

ΔG_S  $= \beta_{S1},$ ΔG_S  $= \beta_{S2},$ ΔG_S  $= \beta_{S3}, \ \ldots$ (1.5)

Where $\beta_{S1},\ \beta_{S2},\ \ldots,\ \beta_{Sn}$ are parameters to a linear regression model we fit to ΔG of 'GGGAAACCC' and similar sequences (see section: UV Melting Experiments). This is what is done in practice.

For the other type of loops, the sizes can grow unbounded, so simply having a term for each loop will not work. However, at least for internal and hairpin loops, in keeping with the nearest-neighbor philosophy, we can keep a term for every combination of the bases that start the loop, and then have a general term for computing the length, addressing these both directly we have:

Hairpin Loop Loops with 3 and 4 unpaired bases are kept in special tables of triloop and tetraloop parameters, respectively. Each possible tetraloop and triloop has it's own energy determined by experiment. Beyond that, the general model is

$$\Delta G_H(n > 3) = \Delta G_{init}(n) + \Delta G_{HStack}(\text{Initializing Stack}) \tag{1.6}$$

$$+ \Delta G(\text{bonuses}). \tag{1.7}$$

As you can see there is an initialization term and a stack term that are both fitted via linear regression. The bonus term is for various special loops that have been experimentally found to be more stable. It's outside the scope of this thesis to get into them, but they are specified in the paper by Mathews et al [5].

Internal Loop Much like stack loops, internal loops are given individual parameters for 1×1, 1×2, 2×2, internal loops, where $n \times m$ denotes one arm of the loop having n unpaired bases and the other having m. This results from extensive studies on the ΔG's of these internal loops. For the rest of internal loops there's a model of the form:

$$\Delta G_{int}(n \times m) = \Delta G_{init}(n+m) + \Delta G_{asymmetry}(|n-m|) \tag{1.8}$$

$$+ \Delta G(\text{bonuses}). \tag{1.9}$$

Where each term on the right is a regression parameter and the bonus term is similar to the hairpin bonus term for loops composed and ended with certain special kinds of bases determined experimentally to be more stable.

Multi-Loops Multi-Loops are harder to create experimental strands for and because of this there are no individual parameters for multi loops. In fact, for modeling mutli-loops we regress to a more simple linear model of the form:

$$\Delta G_{multi} = a_1 + b_1 n + c_1 h + \Delta G_{dangle} \tag{1.10}$$

Where a_1 is a penalty for starting a mutl-loop, n is the number of unpaired bases in the loop, b_1 is an energy penalty per unpaired base, h is the number of pairs in the structure, c_1 is the energy bonus per pair, and ΔG_{dangle} is a term similar to stack loop terms which include the energy of the unit composed of a pair and its 2 adjacent unpaired bases, if it has them.

[TODO: coaxial stacking?]

1.2.2 UV Melting experiments

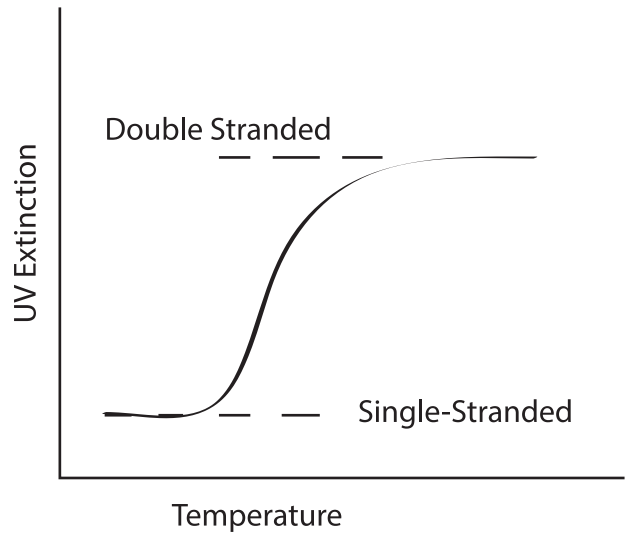
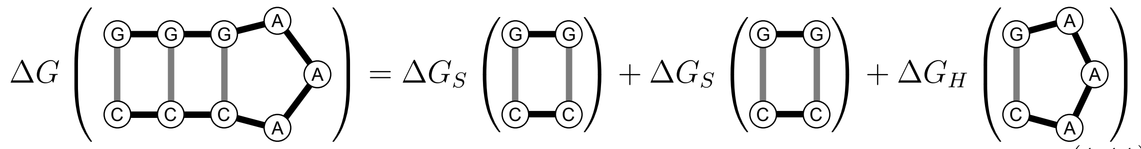


Figure 1-2: An example of the output of a UV melting experiment. We should expect to see 2 levels of extinction in the graph, one corresesopoding to where single-strandedness is the equilibrium condition of the strand and the other where double strandedness is the equilibrium. As the temperature increases, the strand absorbs more energy from the environment allowing it to escape the double-stranded state and so there is a transition interval where as the temperature increases the UV extinction goes from the double-stranded extinction to the single stranded extinction.[TODO: this graph has double stranded and single stranded extinction switched]

Loop regions are given energies as parameters to linear regression models of free energy change in predictably folding strands. For example, the strand ‘GGGAAACCC’ folds predictably into a structure with all the G’s paired to the C’s and a 3-A hairpin turn (because G’s pair very strongly to C’s and A’s tend to resist pairing) as seen in Equation 1.4. Large amounts of identical strands are synthesized and put into solution and heated. A two-state assumption is made, either the RNA is folded in a ‘double-stranded’ state or unfolded in a ‘single-stranded’ state. As the solution heats, there is enough ambient energy to put all the strands in the unfolded, no-bonds state. RNA is an organic, aromatic molecule that absobes light in the UV spectrum in different amounts depending on whether it is in a folded state or unfolded state, so the UV absobtion is fit to a curve that then tells us about the relative concentrations of the single-stranded vs. double-stranded state, which in turn tells us about the free energy change between the two at a given temperature. This free energy change is extracted and treated as a function of the loop variables, which are then fitted to a linear model over experiments on many different such strands [8].

For example, if we wanted to compute the free energy change of a strand and fit it to it's loop parameters:



(1.11)

We synthesize large amounts of the strand 'GGGAAACCC', put them in solution and heat them and record their UV extinction. Observing the curve to be like something in Figure 1-2. The melting temperature T_m is defined to be where the concentrations of single stranded and double stranded molecules are equal, and this is taken to be the inflection point of the graph in Figure 1-2. From there Van't Hoof analysis is performed, where the concentration, C_T, is varied and this follows a model of the form:

$$\frac{1}{T_m} = \frac{R}{\Delta H} \log C_T + \frac{\Delta S}{\Delta H}. \tag{1.12}$$

This equation comes from the relation $\Delta G = -RT \log K_{eq}$ and plotting $1/T_m$ against $\log C_T$ and fitting a linear model gives us the parameters ΔS and ΔH from the slope and intercept and therefore ΔG through the relation

$$\Delta G = \Delta H - T\Delta S. \tag{1.13}$$

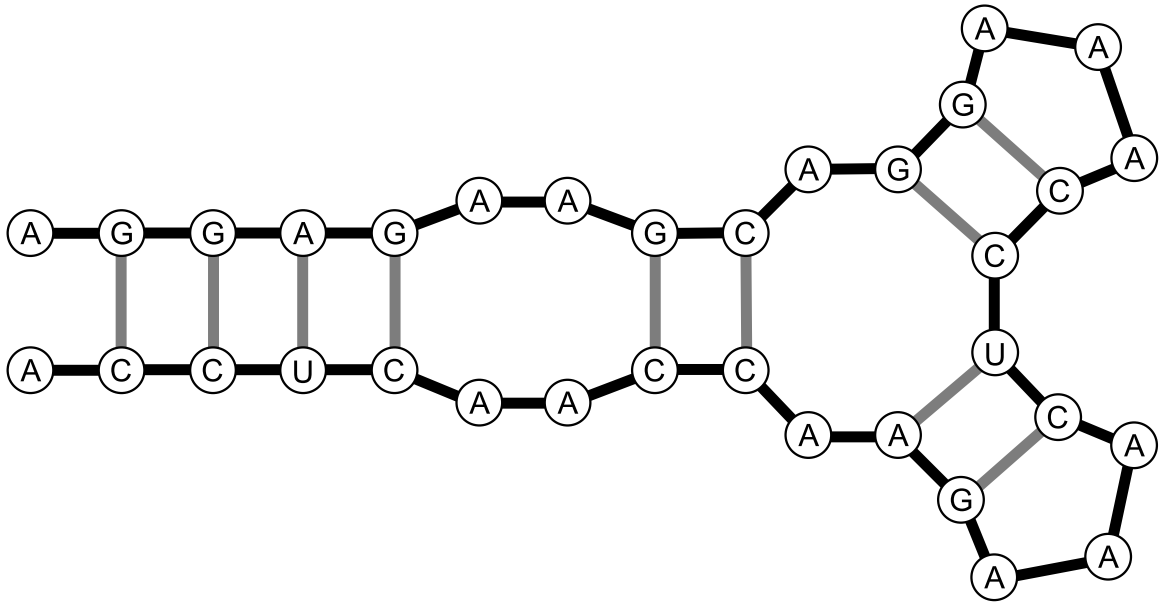


Figure 1-3: This is one possible folding out of the $O(1.8^n)$ secondary structures of the sequence 'AGGAGAAGCAGGAAACCUCAAAGAACCAACUCCA'.

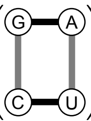
If we repeat this process over several strands, we can then fit the ΔG's to a linear model based on the many parameters described in the Loop Parameters section. From there, when we want to compute the energy of a folding, all we have to do is seperate it into loops and sum the energies of each based off the parameters of this model.

1.2.3 Example Energy Computation

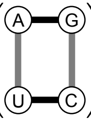
Here's an example of the energy computation of a secondary structure, pictured in Figure 1-3. There are 3 stack loop of the type:

$$\Delta G\left(\begin{array}{ccc} \text{G} & - & \text{G} \\ | & & | \\ \text{C} & - & \text{C} \end{array}\right) = -3.26 \text{ kcal/mol}, \tag{1.14}$$

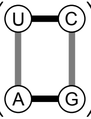
and one of each:



$$\Delta G \left(\quad \right) = -2.35 \text{ kcal/mol}, \tag{1.15}$$

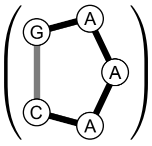


$$\Delta G \left(\quad \right) = -2.35 \text{ kcal/mol}, \tag{1.16}$$



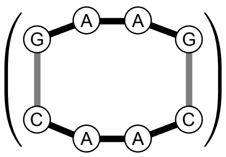
$$\Delta G \left(\quad \right) = -2.08 \text{ kcal/mol}. \tag{1.17}$$

These are all lookups from linear regression parameters. Besides the stacks, there are 2 identical hairpins, an internal loop, an external loop, and a multi-loop. For the hairpins the energy can be looked up in a special parameter table designed specifically for hairpins with 3 unpaired bases called triloops:



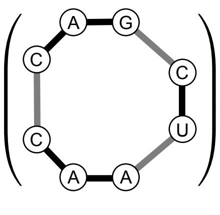
$$\Delta G \left(\quad \right) = 5.8 \tag{1.18}$$

For the internal loop, there is a direct lookup for the 2×2 mismatch in a table:



$$\Delta G \left(\quad \right) = 0.5 \text{ kcals/mol}. \tag{1.19}$$

The multibranch loop has 3 pairs and 2 unpaired bases. The energy is therefore:



$$\Delta G \left(\quad \right) = 3.4 + 0 * 3 + 0.4 * 2 = 4.2 \text{ kcals/mol}. \tag{1.20}$$

If we sum up the contributions, we get that $\Delta G = -6.36$ kcals/mol. The energy of the MFE state can be computed using software, and I find it to be -3.06 with just the first stem and a large hairpin. [TODO: figure out what is wrong, make ct file, find energy].

1.3 Critique of the Energy Model

The energy model of RNA is not perfect, notably so. It is complex, there are linear models, non-linear models, lookup tables, and bonuses. Complexity is a very undesirable quality for a model, it makes it hard to implement and hard for new researchers to get started in the field. Not only that, but many parameters have very high error. For example, the initialization penalty for hairpin loops is supposed to be different for different hairpin lengths, but most of the terms aren't even 1σ away from each other. Because of this, it is not clear what advantage these extra parameters give, perhaps they could be reduced to one, to simplify the model greatly. Another problem is that the data of the original experiments has been lost, so no one can take a closer look at how well the parameters fit, and no one seems to be interested in running the experiments again. The only thing that seems to keep the current parameters in good standing is relatively solid results in prediction, with something like 80% of base pairs correctly predicted. However, to make any progress from this point, it is essential to get the energy model correct.

If there was a list of big problems to tackle in RNA secondary structure prediction, fixing the energy model should be a top priority. The new model should be consistent and physically based and confirmed by experiments. Some groups have made progress on creating energy models for multi-loops that may perform better than the current parameters [TODO: insert aalberts plug]. However, none of these changes have been implemented in the software packages that currently hold the grand majority of the secondary structure prediction market. There are reasons for this, including the fact that the software is massive, complicated, and poorly documented. Perhaps a new energy model could usher in a better software framework for RNA prediction that has:

1. Better naming conventions, with function, variable, and source file names describing precisely what they represent,
2. Consistently documented source files, perhaps with a full manual created with doxygen,
3. Simpler energy models, with less branching and complexity.

This is not to belittle the efforts of those that implemented Unafold and RNAStructure, after all they accomplished the great feat of implementing these complex software suites. Instead, it is to note that the job is not done, and the items listed above are things that need to be accomplished.

1.4 Applications of RNA Structure Prediction

RNA Secondary structure has many clinical applications including in sequence design.

[TODO: finish this section]

Chapter 2

Partition Function Computation and Improvements

2.1 Introduction

The partition function for a thermodynamic system of fixed volume, in contact with a heat reservior with absolute temperature T, is

$$Z = \sum_{s} e^{-E(s)/RT}, \tag{2.1}$$

where s denotes a particular state of the system, $E(s)$ is the energy of that state, and RT is the gas constant multiplied by the temperature, specified above. Each particular term in the sum is called that state's Boltzmann factor. The probability of a state is then said to be its Boltzmann factor divided by the partition function, or

$$P(s) = \frac{1}{Z} e^{-E(s)/RT}. \tag{2.2}$$

In our model an RNA strand is such a thermodynamic system, its secondary structure is its state, and the rest of the cell is the reservoir of heat. We assign each secondary structure an energy according to the free energy model described in Chapter 1. According to this model, the energy of a secondary structure is the sum of the energies of its loops.

To compute the partition function we must sum up every possible secondary structure. There are many possible secondary structures, on the order of 1.8^n where n is the sequence length. However, the computation of the partition function benefits greatly from the linear nature of the energy model: it allows us to recursively define the partition function. For example, say we have computed the full partition function of a strand of n bases, define a function Q such that:

$$Q(i,j) = \text{ sum of boltzmann factors for every structure contained between } i \text{ and } j. \tag{2.3}$$

The full partition computation can therefore be represented as the function Q acting on the bounding two bases 1 and n:

$$Q(1,n) = \sum_{s \text{ on } [1,n]} e^{-E(s)/RT}. \tag{2.4}$$

If we now extend the strand by attaching a small hairpin to the end, as pictured in Figure 2-1, there is no need to go back and recompute the energy of every single secondary structure, rather we can just multiply the already-computed partition function by the boltzmann factor of the hairpin turn (let $E(h)$ be the hairpin energy) to get the new partition function:

$$Q(1,n+5) = \sum_{s \text{ on } [1,n]} e^{-(E(s)+E(h))/RT} = Q(1,n)e^{-E(h)/RT}. \tag{2.5}$$

This replaces an exponential-time computation (the sum), with a constant time computation (multiplying the pre-computed $Q(1,n)$ with the energy term). In general, this technique allows us to compute the partition function efficiently, working up from

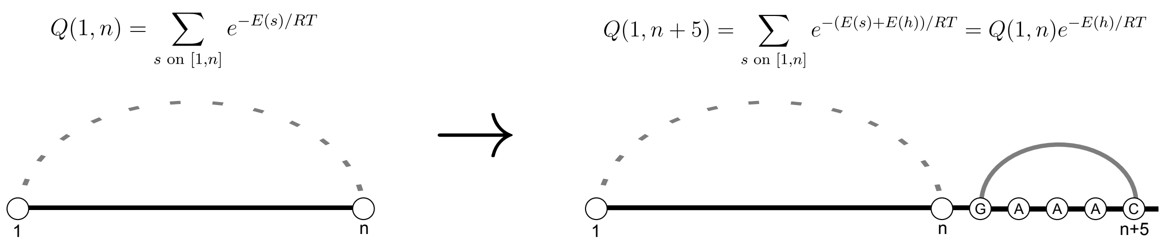

$Q(1,n) = \sum_{s \text{ on } [1,n]} e^{-E(s)/RT}$
$Q(1,n+5) = \sum_{s \text{ on } [1,n]} e^{-(E(s)+E(h))/RT} = Q(1,n)e^{-E(h)/RT}$
1
n
1
n
G
A
A
A
C
n+5

Figure 2-1: Extending the strand by adding an attached hairpin adds the energy of the hairpin to each structure. To compute the new partition function $Q(1, n+5)$, we do not need to sum the energies of all possible structures again, we can simply use the old value, $Q(1, n)$, and multiply it by the Boltzmann factor of the hairpin. This is the concept behind the dynamic programming algorithm.

$Q(1, 2)$ to $Q(1, n)$. In computer science this technique is called dynamic programming, a fancy name for the technique of storing the results of a computation in a table for later use.

The dynamic programming algorithm for computing the partition function of an RNA strands has several versions, depending on how in depth you go with the energy model. The simplest algorithms allow for only structures with nested base pairs. Nested means that the structure can be represented with the parenthesis, for example (((..((((....))))(((..)))..))), or without crossing pairs in expanded loop diagrams such as the ones in Figure 2-1. Note that this kind of structure excludes pseudoknots. If you ignore psuedoknots, which is standard in the field, and if you make an approximation that internal loops will never exceed a certain length, the fastest algorithm runs in $O(n^3)$. We believe that we can streamline this computation even more, taking advantage of the fact that empirically, the number of probable base pairs of a strand of length n seems to grow like n, not n^2. This is the same result we used to speed up the stochastic traceback algorithm and potentially a partition function algorithm that includes pseudoknots. The algorithm will be presented in its simplest form, with more complicated versions available in the appendices.

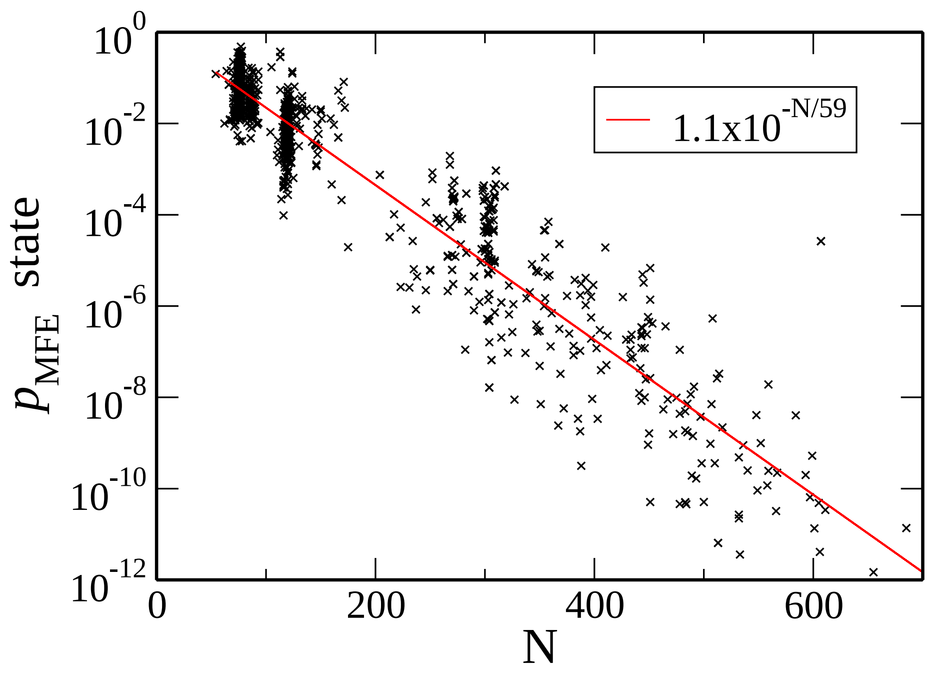


Figure 2-2: A plot of probability of the MFE state as computed $P(MFE) = e^{-E_{min}/RT}/Z$ for sequences of many different lengths. For sequences of reasonable length, even though it is the most probable state, the MFE probability approaches floating point error. This forces us to approach probabilities differently, using sums of probabilities of many similar states instead of just one state. [TODO: where does this data come from?]

2.2 Motivation

There exists an efficient algorithm to find the minimum free energy structure of a sequence. The fundamental algorithm for the MFE was developed by Zuker and Steigler in 1981 [10]. However, perhaps counterintuitively, the MFE structure is not always the structure that is found in nature, even though natural systems tend to their lowest energy state. It is less counterintuitive after seeing Figure 2-2. The probability of any individual state is bounded from above by the MFE, which approaches zero very quickly as the length of the sequence increases. The encourages us to treat the probabilities of secondary structures given to us by Boltzmann statistics as probability densities instead of actual probabilities. The structure that is found in nature is most likely going to be from the most probable macrostate, where we define macrostate as following:

Macrostate: A set of RNA secondary structures sufficiently close to a local energy minimum structure s_{min}.

The "sufficiently close" in the definition is not widely agreed upon, and has spawned several different techniques of defining and computing RNA macrostates (see Clustering chapter).

This logic was presented by Ye Ding, Chi Yu Chan, and Charles E. Lawrence in their 2005 paper "RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble" [1]. What they found that, as suspected, the MFE state was not always a member of the most probable macrostate. Using a method based on centroids of statistically sampled states, they found that their method made 30% fewer prediction errors, improving positive predictive value by 46.5% and sensitivity by 21.7%. Figure 2-3 is an illustration of this concept.

In all the clustering algoritms, since we must know boltzmann statistics of different states, we must compute the partition function. In certain situations, such as partition function clustering, the partition function is recomputed several times. If the partition function takes on the order of hours or days to compute, this can make partition function clustering a major bottleneck in whatever process RNA secondary structure prediction is being used in. However in these situations it is also true that the partition function is recomputed with almost the same conditions, just certain pairs restricted.

We've been able to show via experiment that the partition function only admits roughly $O(n)$ pairs with probabilities above thresholds around the machine precision limit. If we have the partition function already computed, we can recompute it by only adding in pairs that have sufficient probability. We can also extend this method: if a good heuristic appears in the future, one that can eliminate a large number of pairs, while being computationally cheap, we should be able to use the results to speed up the partition function computation. This motivates a method

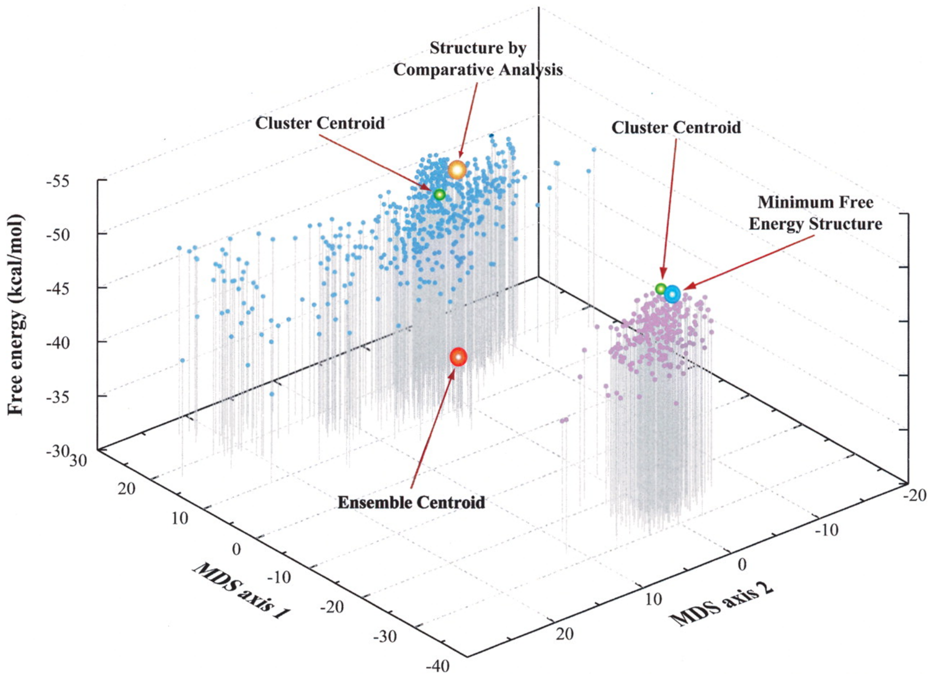


Figure 2-3: An RNA strand with mutliple macrostates, found by clustering analysis and plotted along 2 principle components. The MFE state is not in the most probably macrostate, neither is the ensemble centroid, but the macrostate centroid gives good predictions of secondary structure. Figure from Ding, Chu, & Lawrence (2005).

of computing the partition function using a known-pairs heueristic to prune away unneccesary computation, and that is what we have implemented.

2.3 The Computation

The standard way of computing the partition function involves filling out a table where the (i, j)th member represents the partition function for the substrand from base i to base j, $Q(i, j)$. Because the energy model for RNA is linear, $Q(i, j)$ can be expressed as a function of nearby members of this table. This function is called the recurrence relation for the partition function of RNA. Because the free energy model is so complicated and has gone through many iterations, differenct RNA folding software packages implement different versions of the recurrence relation, and they vary widely in complexity. This version is as simple as possible, there is an energy

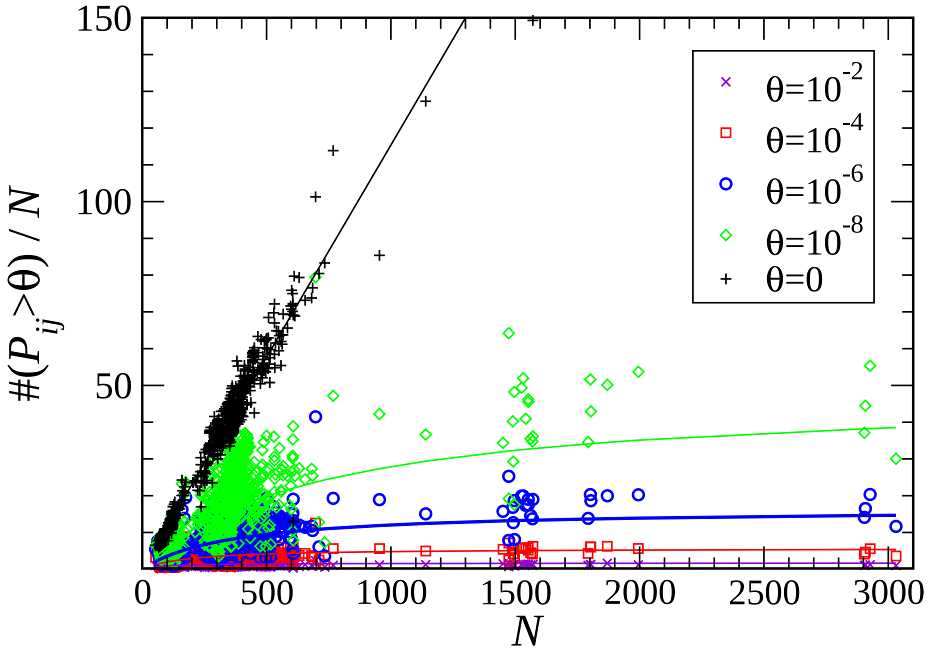


Figure 2-4: A plot of 'number of pairs with probability greater than colored threshhold' vs length of sequence. If there is no threshold, the number of pairs possible goes up like n^2, but even with small thresholds such as requiring a pair to have probability greater than 10^{-8}, the number of possible pairs goes flat as n increases. Therefore, a great amount of efficiency can be gained by restricting the pairs under consideration, while not sacrificing much accuracy since the thresholds are so low.

---

penalty for starting a multiloop, a, and an energy penalty for keeping a base unpaired, b, otherwise we hold to the standard energy functions.

The definitive representation of the recurrence relation for RNA was formulated in 1990 by J.S. McCaskill in his paper *The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure* [6]. Here I will follow the clear presentation by Dirks and Pierce [3].

Starting at the outermost layer, we compute $Q(i, j)$ by adding 2 cases together, either the strand is empty on that interval or it has a rightmost pair (d, e) (see Figure 2-5). If it has a rightmost pair, the partition function must be summed over every combination of structures on the substrand $(i, d-1)$ as well as every structure that

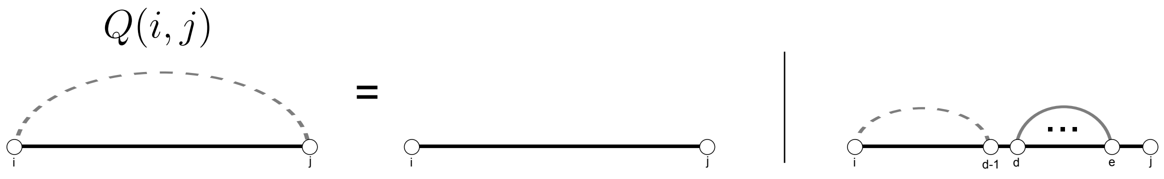


Figure 2-5: The recurrence relation for $Q(i,j)$. When we compute $Q(i,j)$ we only need to consider two cases, and add them together: either the strand will be empty of pairs, or there must be a rightmost pair followed plus whatever else is left, which can be represented as $Q(i,d-1)$. The partition function "underneath" the rightmost pair is $Q^b(d,e)$, which is the partition function with d and e constrained to be paired. The mathematical formula corresponding to this diagram is in equation 2.7.

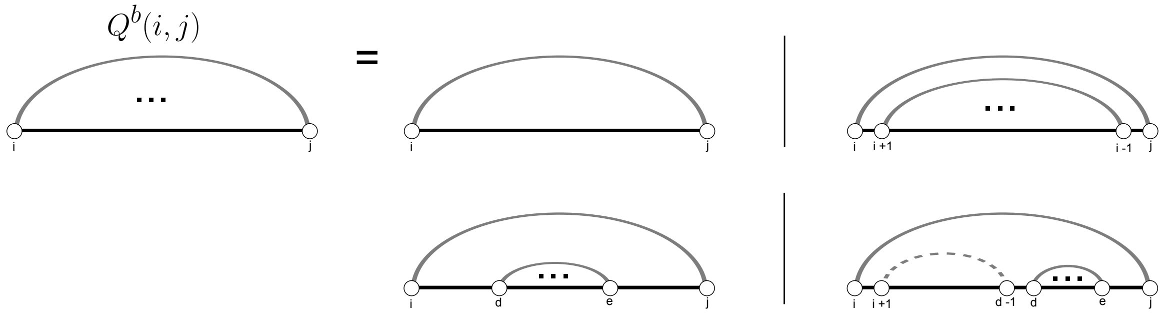


Figure 2-6: The recurrence relation for $Q^b(i,j)$. We consider 4 cases, corresponding to each of the type of loop that could form under the (i,j) pair, and add them together. The loop could form a hairpin, a stack loop, an internal loop, or a multi loop. The mathematical formula corresponding to this diagram is in equation 2.8.

---

could form underneath the pair (d,e). Define:

$$Q^b(i,j) = \text{sum of boltzmann factors for every structure contained underneath pair } (i,j) \tag{2.6}$$

Recalling Figure 2-1, we can get every combination of such states by multiplying $Q(i,d-1)$ by $Q^b(d,e)$ and an energy penalty for leaving $(e+1,j)$ empty. Of course, it must be summed over all possible indices d and e:

$$Q(i,j) = \overbrace{e^{-b(j-i+1)/RT}}^{\text{empty}} + \overbrace{\sum_{\substack{d,e \\ i\le d<e\le j}} Q(i,d-1)Q^b(d,e)e^{-b(j-e)/RT}}^{\text{rightmost pair}}. \tag{2.7}$$

To compute $Q^b(i,j)$ we sum over the 4 cases of loops that could form underneath the assumed pair (i,j). This could mean a hairpin loop, with energy function E_h, a stack loop, with E_s, an interior loop, with E_i, or a multiloop with a, the multiloop penalty. For each new pair created, we must multiply in Q^b for that section as well, and for every open region created, Q (see Figure 2-6).

$$\begin{aligned} Q^b(i,j) = & \overbrace{e^{-E_h(i,j)/RT}}^{\text{hairpin}} + \overbrace{e^{-E_s(i,j)/RT} Q^b(i+1,j-1)}^{\text{stack}} \\ & + \overbrace{\sum_{\substack{d,e \\ i<d<e<j}} E_i(i,d,e,j) Q^b(d,e)}^{\text{internal}} \\ & + \overbrace{e^{-a/RT} \sum_{\substack{d,e \\ i<d<e<j}} Q(i+1,d-1) Q^b(d,e) e^{-b(j-e-1)/RT}}^{\text{multiloop}}. \end{aligned} \tag{2.8}$$

Assuming that we've already computed and stored $Q(d,e)$ and $Q^b(d,e)$ for all (d,e) such that $i \le d < e \le j$, each of those functions is just a table lookup. Therefore the only compute-bounding term is the sum over d and e, which makes computing $Q(i,j)$ an $O(n^2)$ computation, a big speed up over $O(1.8^n)$. The same goes for $Q^b(i,j)$. One catch is that in order to compute $Q(i,j)$ or $Q^b(i,j)$, you must compute all $O(n^2)$ $Q(d,e)$'s and $Q^b(d,e)$'s on the inside, so the entire algorithm is actually $O(n^4)$ but there are known efficiency improvements that bring it down to $O(n^3)$. This thesis presents improvements that bring the algorithm down to $O(n^2)$, provided a good hueristic, in the next section.

2.4 Efficient Partition Function

Let us assume that we have a huerstic of the form of a function

$$K : \text{Base Index} \to [\text{Base Index}], \tag{2.9}$$

that takes the index of a base as input and outputs a list of other bases it can pair to, where the length of this list is always very short and can be considered constant length in relation to the sequence length n. This is not an unreasonable assumption in certain cases. For example if we have already computed the partition function of the strand but are now restricting certain bases from pairing, as in Nestor PF clustering, we already know a lot about what pairs can form. Another possibility could be restricting the base pairs based on sequence alignment. We could also just restrict non-Watson-Crick pairs to start. Importantly, we have seen empirically that the list of pairs above a probability theshold for each base will be very short.

The problem terms of each computation are the sums over d and e. In order to get the algorithm down to $O(n^2)$ each $Q(i,j)$ and $Q^b(i,j)$ must take constant time. Efficiency gains have already been seen by McCaskill by limiting the internal loop length, introducing the constraint that $(d-i)+(j-e) < L$ for some constant L. This way, as n computation is limited to $O(L^2)$ which is a constant. That leaves only the multi-loop term to make a constant time computation. Using the hueristic, this is possible. Let us define a new function (and table) $Q^m(i,j)$ which is defined as (this time explicitly summing d and e):

$$Q^m(i,j) = \sum_{d=i}^{j} \sum_{e=d+1}^{j} Q(i,d-1)Q^b(d,e)e^{-b(j-e)/RT}. \tag{2.10}$$

We can take the out all the terms which contain $e = j$ to split the formula into two parts (note that I limited the first term's d sum to only go to $j-1$ because d must always be less than e):

$$\begin{aligned} Q^m(i,j) = &\sum_{d=i}^{j-1} \sum_{e=d+1}^{j-1} Q(i,d-1)Q^b(d,e)e^{-b(j-e)/RT} \\ &+ \sum_{d=i}^{j} Q(i,d-1)Q^b(d,j). \end{aligned} \tag{2.11}$$

The first term in this new equation is the same thing $Q^m(i,j-1)e^{-b/RT}$. In the second term, it is always implied that d pairs with j, so we can substitute the summation index for our hueristic for j, $K(j)$, without losing anything. Therefore, our formula becomes:

$$Q^m(i,j) = Q^m(i,j-1)e^{-b/RT} + \sum_{k \in K(j)} Q(i,k-1)Q^b(k,j). \tag{2.12}$$

Now, we can examine the efficiency of computing $Q^m(i,j)$. The first term is a constant factor and the second is a sum over only a constant number of indices by our empirical assumptions about the hueristic. Therefore the computation of Q^m can be treated as a constant time computation.

Now the computation for $Q(i,j)$ becomes:

$$Q(i,j) = e^{-b(j-i+1)/RT} + Q^m(i,j), \tag{2.13}$$

and for $Q^b(i,j)$:

$$\begin{aligned} Q^b(i,j) =& e^{-E_h(i,j)/RT} + e^{-E_s(i,j)/RT}Q^b(i+1,j-1) \\ &+ \sum_{\substack{d,e \\ i<d<e<j \\ (d-i)+(j-e)<L}} E_i(i,d,e,j)Q^b(d,e) \\ &+ e^{-a/RT}Q^m(i+1,d-1). \end{aligned} \tag{2.14}$$

So assisted by this hueristic we are able to reduce all table updates to constant time computations. This means the algorithm is only limited by the number of (i,j)'s which grows with $O(n^2)$.

2.5 Results

Calculating the partition function is still difficult, but it can be improved in the asymptotic case. The results show that the new computation has a large constant factor in the beginning of the algorithm. This could be related to the way the internal loop sum is calculated, if $L = 30$, the constant attached the the loop sum is $L^2 = 900$. This constant term will only stop dominating as the number of bases gets much larger than 900, as we can see by the plot. If we reduce L, we should also see a reduction in probability barrier, we get further reductions in runtime.

[TODO: include plot with different times for different probability thresholds]

A large part of the verification of an improved algorithm is showing that it produces acceptable results compared to the old algorithm, as you can see in the plot, our algorithm gives the same results up to 1 part in 1 million [TODO: check this]. Increasing the probability threshold changes the results by [TODO: find out].

[TODO: include plot of verification]

2.6 Conclusion

The partition function computation is hard and is a limiting factor in RNA structure prediction. However, given a heuristic to predict the base pairs, we can make the algorithm run much faster. These improvements can be used in applications such as clustering based on our nestedness measure, using the partition function. These improvements also open up several possible branches for further investigation. Can a heuristic be generated before the partition function is computed, so that the pairs can be restricted and the computation done in much faster time than the $O(n^3)$ algorithm? Could this concept be applied to the pseudoknot algorithm to possibly speedup the computation for that as well? Applications of this sort could have tremendous impact on RNA secondary structure prediction, opening up incredible avenues of potential.

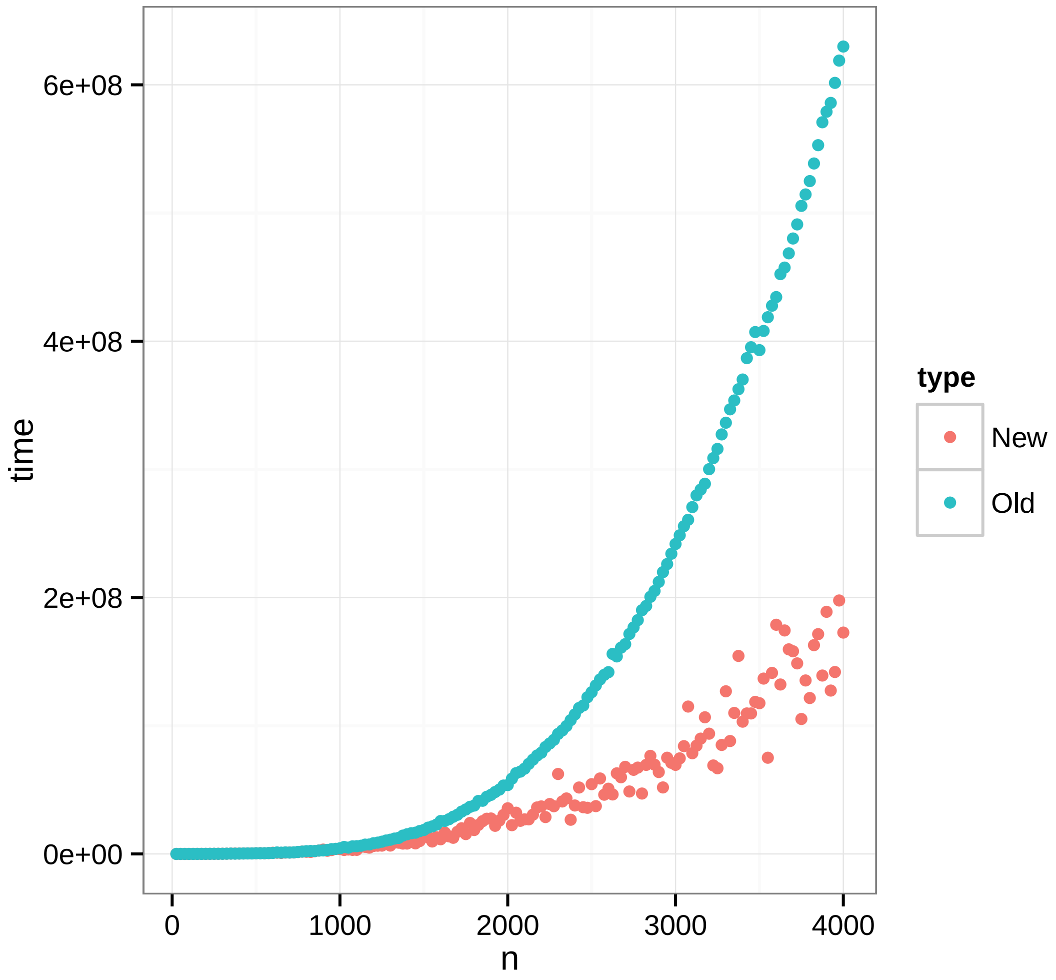


Figure 2-7: Computation time vs length for random sequences, the old $O(n^3)$ partition function run against the new $O(n^2)$ partition function. Not quite the results we were after, however, we can always set the max internal loop length L to a lower value and increase the probability threshold θ.

Chapter 3

Stochastic Traceback Algorithm and Improvements

3.1 Introduction

The stochastic traceback algorithm was introduced by Ye Ding and Charles Lawrence [2] as a means to explore the energy landscape of RNA by sampling structures according to their Boltzmann probabilities. This was important because the minimum free energy structure was very sensitive to errors in the parameters of the free energy model, and although algorithms existed for generating suboptimal structures, they either sampled a very limited set of states [9], or had exponential runtime and the output did not have the same distribution as the physical ensemble of states [**?**]. Structures sampled according to this algorithm can be clustered into macrostates, whos properties can be examined.

The method first computes the partition function, then it "traces back" over the contents of the tables calculated during that algorithm. Specifically, the tables $Q(i, j)$, $Q^b(i, j)$, etc. now contain information about the conditional probabilities of bases pairing. Starting from a bare, undetermined strand at the top of the recurrence relations, the stochastic traceback samples pairs between bases until a structure is formed such that its probability of it being created by the algorithm is equal to its boltzmann weight $e^{-E(s)/RT}/Z$.

The general principle of the backwards trace is that, presented with several possibilities for the structure along a sequence from i to j, the sampling probability for a case is the contribution to the partition function by that case's partition function. For example, since the partition function $Q(i,j)$ is defined:

$$Q(i,j) = \overbrace{e^{-b(j-i+1)/RT}}^{\text{empty}} + \overbrace{\sum_{\substack{d,e \\ i\le d<e\le j}} Q(i,d-1)Q^b(d,e)e^{-b(j-e)/RT}}^{\text{rightmost pair}}, \tag{3.1}$$

during a stochastic traceback the probability of sampling a pairless strand, and the probability of drawing a rightmost pair (d,e) and recursing from there are:

$$P(\text{empty}|i,j) = \frac{e^{-b(j-i+1)/RT}}{Q(i,j)} \tag{3.2}$$

$$P(\text{pair }(d,e)|i,j) = \frac{Q(i,d-1)Q^b(d,e)e^{-b(j-e)/RT}}{Q(i,j)}. \tag{3.3}$$

Notice that when summed together for every possible case of (d,e), the numerator becomes the definition of $Q(i,j)$, so these probabilities are naturally normalized. Likewise, since $Q^b(i,j)$ is defined as

$$\begin{aligned} Q^b(i,j) = {} & \overbrace{e^{-E_h(i,j)/RT}}^{\text{hairpin}} + \overbrace{e^{-E_s(i,j)/RT}Q^b(i+1,j-1)}^{\text{stack}} \\ & + \overbrace{\sum_{\substack{d,e \\ i<d<e<j}} e^{-E_i(i,d,e,j)/RT}Q^b(d,e)}^{\text{internal}} \\ & + \overbrace{e^{-a/RT}\sum_{\substack{d,e \\ i<d<e<j}} Q(i+1,d-1)Q^b(d,e)e^{-b(j-e-1)/RT}}^{\text{multiloop}}. \end{aligned} \tag{3.4}$$

The probability of sampling a hairpin, stack loop, internal loop, and multiloop are:

$$P(\text{hairpin}|i,j) = \frac{e^{-E_h(i,j)/RT}}{Q^b(i,j)} \tag{3.5}$$

$$P(\text{stack}|i,j) = \frac{e^{-E_s(i,j)/RT}Q^b(i+1,j-1)}{Q^b(i,j)} \tag{3.6}$$

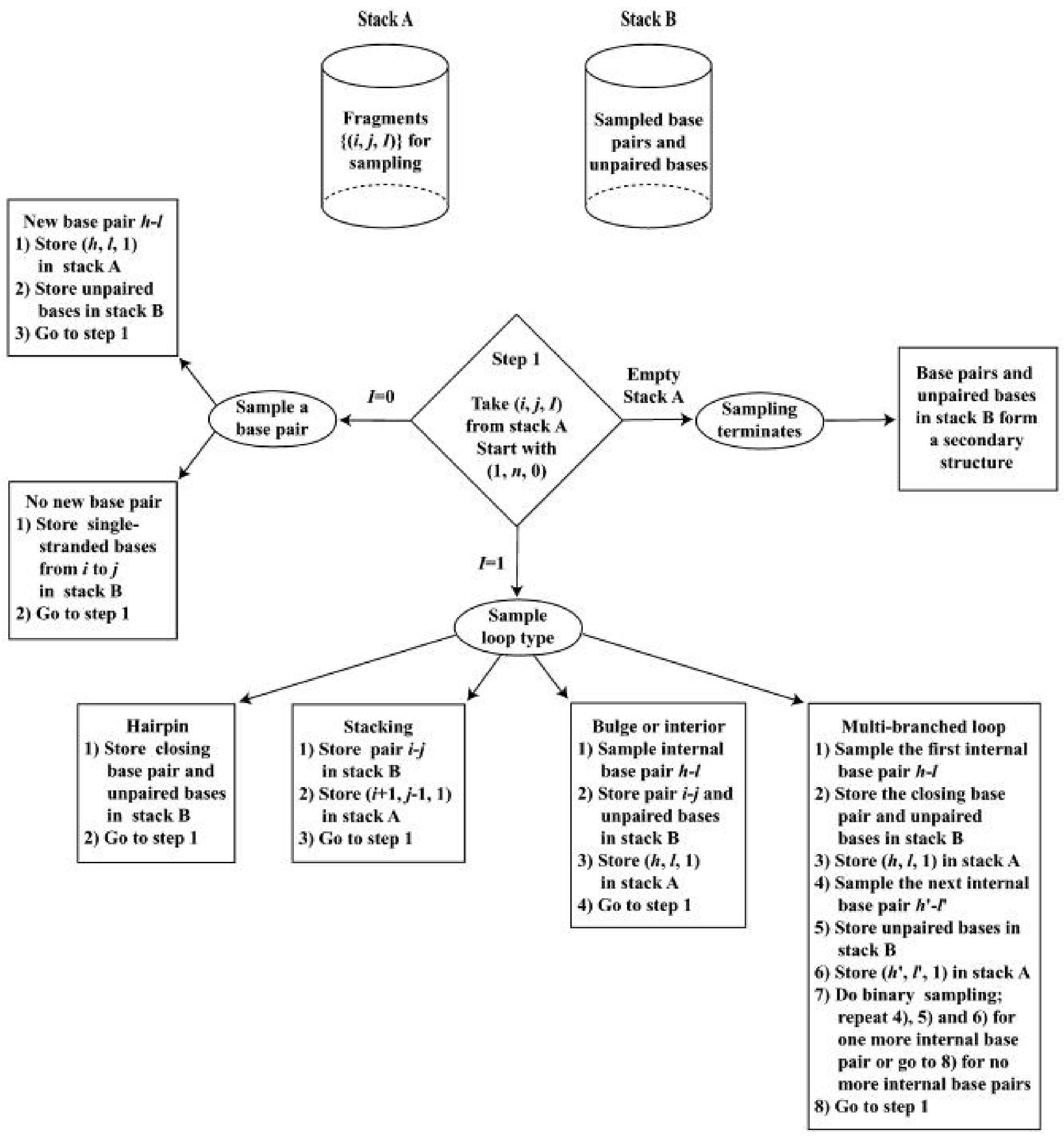
$$P(\text{internal }(d,e)|i,j) = \frac{e^{-E_i(i,d,e,j)/RT}Q^b(d,e)}{Q^b(i,j)} \tag{3.7}$$

$$P(\text{multi }(d,e)|i,j) = \frac{e^{-a/RT}Q(i+1,d-1)Q^b(d,e)e^{-b(j-e-1)/RT}}{Q^b(i,j)} \tag{3.8}$$

Notice for the same reason as before, all the probabilities for the Q^b recursion sum to one. In general, the stochastic traceback can be readily understood by examining the recurrence relation figures from chapter 2: Figure 2-5 and Figure 2-6. We start at the Q recursion for the strand, we may pick a pair and add it to the structure, but in general, for every pair added we recurse down to Q^b on that region to determine the structure below that pair. For every structure left undetermined, we use the probabilities from Q to determine that structure and so on.

The algorithm's implementation uses two stack data structures. A stack can be thought of as a literal "stack", like papers stacked on a desk, except instead of paper their are items of data. There are two basic operations, one to put an item on the top of the stack, and another to retrieve an item off the top. These are called "push" and "pop" operations, respectively, in Computer Science. The data items we will be pushing on to the first stack, A, are of the form $\{(i,j),b\}$ where i and j are indexes along the strand and b is either *true* if we have determined that i and j are paired, or $false$ otherwise. The second stack, B, is where we'll collect pairs and unpaired bases for one sample.

The algorithm displayed in Figure 3-2 is what gets the job done. The probability of the structure is equal to the product of the probabilities that determined its pairs and empty regions, but if you follow the recursion all the way down you see that this



Stack A
Fragments {(i, j, I)} for sampling
Stack B
Sampled base pairs and unpaired bases
New base pair h-l
1) Store (h, l, 1) in stack A
2) Store unpaired bases in stack B
3) Go to step 1
Sample a base pair
I=0
Step 1
Take (i, j, I) from stack A
Start with (1, n, 0)
Empty Stack A
Sampling terminates
Base pairs and unpaired bases in stack B form a secondary structure
No new base pair
1) Store single-stranded bases from i to j in stack B
2) Go to step 1
I=1
Sample loop type
Hairpin
1) Store closing base pair and unpaired bases in stack B
2) Go to step 1
Stacking
1) Store pair i-j in stack B
2) Store (i+1, j-1, 1) in stack A
3) Go to step 1
Bulge or interior
1) Sample internal base pair h-l
2) Store pair i-j and unpaired bases in stack B
3) Store (h, l, 1) in stack A
4) Go to step 1
Multi-branched loop
1) Sample the first internal base pair h-l
2) Store the closing base pair and unpaired bases in stack B
3) Store (h, l, 1) in stack A
4) Sample the next internal base pair h'-l'
5) Store unpaired bases in stack B
6) Store (h', l', 1) in stack A
7) Do binary sampling; repeat 4), 5) and 6) for one more internal base pair or go to 8) for no more internal base pairs
8) Go to step 1

Figure 3-1: Flowchart of the algorithm from Ding and Lawrence's paper [2]. Basically: if we are on a $Q(i, j)$ recursion, sample either an empty section or a rightmost pair. For Q^b choose a loop type and corresponding pairs if applicable. After both cases recurse down appropriately so that Q^b is drawn for any pair created and Q for any open region.

The initialization of the algorithm is to push $\{(1,n), false\}$ onto the stack. From there the algorithm repeats the following steps:

1. **Pop an element, $\{i, j, b\}$ off stack A.**
2. **Case b is $false$:**
   (a) **($Q(i,j)$ recursion) Pick empty or pair (k,l) with probabilities listed above. If empty, push all pairs on $[i,j]$ inclusive onto B as unpaired bases, if not push $\{(d,e), true\}$ and $\{(i, d-1), false\}$ onto A.**
3. **Case b is $true$:**
   (a) **($Q^b(i,j)$ recursion) Choose what type of loop (i,j) is from the probabilities listed above.**
   (b) **Push the appropriate elements onto the stack for that loop type, see Figure 3-1.**
4. **If stack A is empty, the pairs and unpaired bases in stack B become a sampled structure. Reinitialize for additional samples.**

Figure 3-2: The stochastic traceback algorithm is implemented by pushing the recursive elements back onto the stack to sample a full structure.

---

results in just the boltzmann factor for the structure:

$$P(s) = \prod_{\text{cases}} \frac{P(\text{case})}{Q(\text{case})} = \frac{1}{Z} e^{-E(s)/RT}. \tag{3.9}$$

3.2 Motivation

In the past 10 years, the stochastic traceback algorithm has become an increasingly central part of RNA secondary structure prediction algorithms (Ding et al 2005, [TODO: cite more]). This is because they present many advantages over the minimum free energy prediction. It can be shown that the minimum free energy state, even though it is the most probable state, can still have astronomically unlikely probabilities on average for typical strands of reasonable length ([TODO: cite, figure]). The more important concept in understanding the physical behavior of an RNA strand is therefore the overall shape of the energy landscape. Althogh the probability of any individual structure might be infinitesimally small, there can be shown to be relatively few large basins containing clusters of similar foldings. The consensus structures and the difference between the consensus structures of these basins define the function of the RNA molecule.

The way the stochastic algorithms probe that is by providing structures to group into these basins, and since the stochastic traceback algorithm samples states with the exact probability defined by the partition function, we know that the macrobehavior of these samples match what we would probably see in reality. There is one catch and that is statistical error. However, the error can be reduced and the landscape can be further explored the more stochastic samples we make.

The need to sample large numbers of secondary structures makes a speedup very convenient, and that is what motivates our current expedition.

3.3 Efficiency Improvements

Taking advantage of the empirical fact that the number of probable base pairs for an RNA strand tend to grow very slowly, we can restrict our traceback to only explore bases that we know can pair with one another. This is as simple as replacing the old recurrence relations with the new ones. For example during a $Q(i,j)$ state of the recursion there would be probabilities of:

$$P(\text{empty}|i,j) = \frac{e^{-b(j-i+1)/RT}}{Q(i,j)} \tag{3.10}$$

$$P(\text{goto } Q^m|i,j) = \frac{Q^m(i,j)}{Q(i,j)}. \tag{3.11}$$

For a Q^m state:

$$P(\text{continue}|i,j) = \frac{Q^m(i,j-1)e^{-b/RT}}{Q^m(i,j)} \tag{3.12}$$

$$P(\text{pair } (k,j)|i,j) = \frac{Q(i,k-1)Q^b(k,j)}{Q^m(i,j)}. \tag{3.13}$$

And finally for $Q^b(i,j)$:

$$P(\text{hairpin}|i,j) = \frac{e^{-E_h(i,j)/RT}}{Q^b(i,j)} \tag{3.14}$$

$$P(\text{stack}|i,j) = \frac{e^{-E_s(i,j)/RT}Q^b(i+1,j-1)}{Q^b(i,j)} \tag{3.15}$$

$$P(\text{internal } (d,e)|i,j) = \frac{e^{-E_i(i,d,e,j)/RT}Q^b(d,e)}{Q^b(i,j)} \tag{3.16}$$

$$P(\text{goto } Q^m|i,j) = \frac{e^{-a/RT}Q^m(i,j)}{Q^b(i,j)}. \tag{3.17}$$

As one can see from the tables, the speedup is enormous. For randomly sampled sequences up to lengths in the thousands, the old stochastic timing grows quadratically, while the new method flatlines below it.

[TODO: add speedup figure]

A good question to ask would be, how do we know that this new algorithm is outputting structures with the correct probabilities. Verification plots here attempt to answer that question.

[TODO: add verification plot]

What we would expect to see from these plots, is that for a given base, we would expect to see it pair with other bases with probabilities given by the partition function as one can see. Of course there is sampling error, so each bin represents a sampling from a Bernoulli distribution. For n^2 samples, we would expect [Todo: find out what error we expect] error. The number of samples that violate the bounds, do not deviate much from what we would expect from doing n^2 experiments, so I think we can confidently say that the new algorithm is making the correct computation.

[TODO: Add demonstration of full RAM sampling]

3.4 Conclusion

The stochastic traceback algorithm can be sped up to allow large amounts of sampling such that the limiting computation factor is memory, not time. The applications of this improvement are that statistical algorithms like Ding & Lawrence ([TODO: cite]) can minimize their statistical error. Nestor ([TODO: cite]) benefits greatly from these improvements because more structures mean more branches to the tree. In general the algorithm is useful because it removes unnecessary computation that was done before.

Chapter 4

Nestor and Clustering

4.1 Motivation

Given that we know, for any given RNA strand, the probability of an individual state is very low [TODO: reference section], a much more important computation is the overall shape of the strand's free energy landscape. Even if the probability of an individual state is low, if we "integrate" over a basin of free energy, the probability of that set of states could be something tangible.

In the past 10 years, several groups have started to explore this concept. There are two approaches, in general, to define basins and classify structures into them. The first class of methods defines the basins from the top-down: given a number of stochastically sampled structures, we divide them into groups based on some kind of distance metric. These methods tend to be very similar to typical clustering algorithms used in computer science and data analysis.

Another approach is to start at local minima and climb up the energy barriers between minima using the metropolis-hastings algorithm to maintain the correct probabilities according to the partition function. These methods can be used to accurately compute the energies of the transition states between local minima and these can tell you the kinetics of the structure. This technique was developed by [TODO: find Vienna people and cite them].

4.2 Nestor

Enabled by the stochastic traceback algorithm to sample large numbers of states from the Boltzmann distribution, the Nestor algorithm, developed by Aalberts and Jannen, uses a different measure of distance based on the number of conflicting pairs between two structures. This measure is called nestedness and it is defined as $\Psi^{\mu\nu}$ between structures μ and ν, such that

$$\Psi^{\mu\nu} = \sum_{k,l} \sum_{m,n \times (k,l)} P^{\mu}_{kl} P^{\nu}_{mn} \tag{4.1}$$

The algorithm works as follows: given an ensemble of structures S, find the most non-nested pair, p, from S and sort the structures into 2 groups: structures compatible (non-crossing) with p and not compatible (crossing) p. These two sub ensembles become the children of S in a tree that is constructed by recursively splitting ensembles in the same manner.

Chapter 5

Reactivity Experiments

5.1 Methods

[Pretty rudimentary description of Das's process] Das Lab at Stanford perform chemical mapping experiments on RNA molecules. An RNA strand of interest is selected and is from there on called the wildtype strand (abbreviated WT). Then for each nucleotide in the strand a mutant is created switching out that particular base with its Watson and Crick opposite. This is intended to perturb the energy landscape in such a way that dominant loops may become less prominent and other foldings become more stable. SHAPE analysis is then done on each strand to prob which bases are paired and which are not.

Data was obtained in the form of RDat files from Stanford's RNA mapping database. SHAPE reactivity is extracted from these files for the WT and each of its mutants. To normalize the reactivity trace of a strand to a probability on $[0,1]$ first the partition function is calculated for this strand, then probability of each base being paired is computed using the formula

$$basepair formula,$$

and finally these probabilities are rank sorted and fitted to a fermi distribution using a least squares gradient decent fit [figure here]. We believe that the measured reactivity should relate [correlate, correspond?] to the probability that a base is unpaired, so the reactivities are reverse rank sorted and mapped to the fermi distribution found by our fit.

From here, using the assumption that each mutation changes the relative energies of each macrostate without changing their internal structures, we fit this data to a model of k clusters each with n nucleotide probabilities, with then $k * (n + 1)$ cluster probabilities. Therefore we have a model with $k(2*n+1)$ paramters fitting to $n*(n+1)$ data entries. A boxed gradient decent is used to minimize a cost function:

$Costfunction$

This fit results in k fitted clusters with $k * (n + 1)$ cluster probabilities.

These fitted clusters are compared to k nests generated by Nestor. The nests are created using the methods desribed in [Nestor chapter] for each strand. Since these nests are created independant of any other strands, nests for different strands must be matched to each other in order to compare to the fitted clusters.

[paragraph on the matching process, still investigating]

Once these matches are made we can compare the cluster vs nest probabilites for the WT and each mutant and see how they correlate, as well as investigate other clusters that may be found.

Chapter 6

Pseudoknot Algorithm

6.1 Introduction

The partition function algorithm discussed in this thesis and the literature in general has a crucial flaw that makes it incomplete. The algorithm assumes that RNA secondary structure is 'well nested', that there are no overlapping pairs (i, j) and (k, l) such that one of k or l is between i and j and the other is not.

[TODO: include figure of nestedness vs non-nestedness]

It is not true that RNA secondary structures are well-nested in general. When a non-nested pair is present in a secondary structure, that structure is said to contain a 'pseudoknot'. Large groups of classified RNA sequences such as Group I Introns (around 6% pseudeoknotted) and RNase P (around 13% pseudoknotted) have pseudoknots in their chemically determined secondary structure, and it is in these groups that secondary structure prediction performs the worst (Matthews et all 1999). Large RNA molecules tend to have pseudoknots, which is unfortunate because the partition function computation scales much worse when pseudoknots are included.

In general, computation of pseudoknotted structures is hard. In fact, the problem has been shown to be NP-Complete [TODO: cite]. Algorithms to perform the computation for a signficiant subset of pseudoknots have been designed, most notably by Dirks and Pierce, who have specified an $O(n^8)$ algorithm and a $O(n^5)$ simplification of that algorithm, using the same approximations used to reduce the complexity of the internal loop energy computation.

Our improvements bring a new concept to the computation of the pseudoknot partition function: first do the unpseudoknotted partition function computation, then redo the partition function including pseudoknots, filtering the allowed pairs by their probabilities according to the first computation. This approach basically combines a heuristic and dynamic programming approach.

[TODO: compile pseudoknot recursion figures]

6.2 Derivation

The derivation of our pseudoknot recursion follows the path of our original. The partition function is recursive in the same way, except now there is a term for bases i and j containing a pseudoknot, $Q^p(i,j)$, to go along with the standard paired region $Q^b(i,j)$.

[TODO: straighten out internal loop penalities vs external loop penalties]

$$Q(i,j) = e^{-\beta b(j-i)} + \sum_{k,l} Q(i,k-1)\left(Q^b(k,l) + Q^p(k,l)\right) \tag{6.1}$$

For a paired region we can now have a hairpin loop, an interrior loop, a paired region with an undetermined region, one pseudoknot, or one pseudoknot with an undetermined region.

$$
\begin{aligned}
Q^b(i,j) =& Q^{HAIRPIN}(i,j) + QBI(i,j) + QPI(i,j) \\
&+ \sum_{k,l} Q(i,k-1)(Q^b(k,l) + Q^p(k,l))
\end{aligned}
\tag{6.2}
$$

For a pseudoknotted region, we must account for every way we could pseudoknot a region, this includes summing over 6 indices (see figure). These include new terms for the undetermined region inside a pseudoknot $Q^z(i,j)$ and a pseudoknot internal loop $Q^b(i,k,l,j)$.

$$
Q^p(i,j) = \sum_{a,b,c,d,e,d,f} Q^g(i,a,d,e)Q^g(b,c,f,j)Q^z(a+1,b-1)Q^z(c+1,d-1)Q^z(e+1,f-1)
\tag{6.3}
$$

The undertermined loop inside a pseudoknot, $Q^z(i,j)$, is the same as $Q(i,j)$, except with energy penalties set by the pseudoknot model.

$$
Q^z(i,j) = e^{-\beta b_p(j-i)} + \sum_{k,l} Q^z(i,j)Q^b(i,j)
\tag{6.4}
$$

The last term is the internal loop for the pseudoknot (see figure):

$$
Q^g(i,k,l,j) = Q^{INTERNAL}(i,j,k,l) + \sum_{a,b} Q(i+1,a)Q^g(a,k,l,b)Q(l+1,j-1)
\tag{6.5}
$$

Chapter 7

Conclusion

Conclude conclude conclude

Appendix A

UNAfold Implementation of Partition Function Improvements

Note the terms Z_{ND}, $Z_{3'D}$, $Z_{5'D}$, and Z_{DD} are extra free energy terms corresponding to 'dangle energies' which are the results of an experiment later implemented in the model to improve it from the standard energy model. In addition there are AU penalty terms appended to where pairs are made, as AU and GU pairs have penalties associated with forming. These additional energy terms improve the model's predictive ability and bring the model closer to the "truth", however it unfortunately makes the partition function seem very threatening.

A.1 New Q(i,j) derivation

In UNAfold, we have that the old recurrence relations were as follows:

$$Q(i,j) = \sum_{k=i}^{j} \left(Q(i,k-1) + e^{-\frac{b(k-i)}{RT}} \right) Q^1(k,j) \tag{A.1}$$

where

$$\begin{aligned}
Q^1(i,j) = \; & Q^1(i,j-1)e^{-\frac{b}{RT}} \\
& + e^{-\frac{c}{RT}} Z_{ND}(i,j)Q'(i,j) \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(i+1,j)Q'(i+1,j) \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(i,j-1)Q'(i,j-1) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(i+1,j-1)Q'(i+1,j-1)
\end{aligned} \tag{A.2}$$

We can expand the recursive definition of $Q^1(i,j)$:

$$\begin{aligned}
Q^1(i,j) = \sum_{k'=i+1}^{j} e^{-\frac{b(j-k')}{RT}} \Big[\; & e^{-\frac{c}{RT}} Z_{ND}(i,k')Q'(i,k') \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(i+1,k')Q'(i+1,k') \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(i,k'-1)Q'(i,k'-1) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(i+1,k'-1)Q'(i+1,k'-1) \; \Big]
\end{aligned} \tag{A.3}$$

Plugging this into $Q(i,j)$ we get:

$$\begin{aligned}
Q(i,j) = \sum_{k=i}^{j} \sum_{k'=k+1}^{j} \Big(Q(i,k-1) + e^{-\frac{b(k-i)}{RT}} \Big) e^{-\frac{b(j-k')}{RT}} \Big[\; & e^{-\frac{c}{RT}} Z_{ND}(k,k')Q'(k,k') \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(k+1,k')Q'(k+1,k') \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(k,k'-1)Q'(k,k'-1) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(k+1,k'-1)Q'(k+1,k'-1) \; \Big]
\end{aligned} \tag{A.4}$$

Now we'll take the jth element of the second sum and split it out (note that the jth part of the 1st sum has no elements to sum now, so we can decrement that too):

$$
\begin{aligned}
Q(i,j) = \sum_{k=i}^{j-1} \sum_{k'=k+1}^{j-1} \Big(Q(i,k-1) + e^{-\frac{b(k-i)}{RT}}\Big) e^{-\frac{b(j-k')}{RT}} \Big[\; & e^{-\frac{c}{RT}} Z_{ND}(k,k')Q'(k,k') \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(k+1,k')Q'(k+1,k') \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(k,k'-1)Q'(k,k'-1) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(k+1,k'-1)Q'(k+1,k'-1) \Big] \\
+ \sum_{k=i}^{j} \Big(Q(i,k-1) + e^{-\frac{b(k-i)}{RT}}\Big) \Big[\; & e^{-\frac{c}{RT}} Z_{ND}(k,j)Q'(k,j) \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(k+1,j)Q'(k+1,j) \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(k,j-1)Q'(k,j-1) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(k+1,j-1)Q'(k+1,j-1) \Big]
\end{aligned} \tag{A.5}
$$

Notice that the double sum is simply $Q(i,j-1)e^{-b/RT}$ and the terms of the second, single sum are over the pairs with j or $j-1$. Therefore, we can use our heuristic for the pairs of j and $j-1$ to produce the following computation for $Q(i,j)$ which is much more efficient than the previous ones:

$$
\begin{aligned}
Q(i,j) = Q(i,j-1)e^{-b/RT} + \sum_{k(j)} \Big[& \Big(Q(i,k-1) + e^{-\frac{b(k-i)}{RT}}\Big)\; e^{-\frac{c}{RT}} Z_{ND}(k,j)Q'(k,j) \\
& + \Big(Q(i,k-2) + e^{-\frac{b(k-i-1)}{RT}}\Big) e^{-\frac{b+c}{RT}} Z_{5'D}(k,j)Q'(k,j) \Big] \\
+ \sum_{l(j-1)} \Big[& \Big(Q(i,l-1) + e^{-\frac{b(l-i)}{RT}}\Big)\; e^{-\frac{c}{RT}} Z_{ND}(l,j-1)Q'(l,j-1) \\
& + \Big(Q(i,l-2) + e^{-\frac{b(l-i-1)}{RT}}\Big) e^{-\frac{2b+c}{RT}} Z_{DD}(l,j-1)Q'(l,j-1) \Big]
\end{aligned} \tag{A.6}
$$

A.2 Derivation of new Q'(i, j) formula

For $Q'(i,j)$ we start with the recursion:

$$\begin{aligned} Q'(i,j) = Z_H(i,j) + Z_S(i,j)Q'(i+1,j-1) + QBI(i,j) \\ + e^{-\frac{a+c}{RT}} Z_{ND}(j,i) \sum_{k=i+3}^{j-5} Q(i+1,k-1)Q^1(k,j-1) \\ + e^{-\frac{a+b+c}{RT}} Z_{3'D}(j,i) \sum_{k=i+4}^{j-5} Q(i+2,k-1)Q^1(k,j-1) \\ + e^{-\frac{a+b+c}{RT}} Z_{5'D}(j,i) \sum_{k=i+3}^{j-6} Q(i+1,k-1)Q^1(k,j-2) \\ + e^{-\frac{a+2b+c}{RT}} Z_{DD}(j,i) \sum_{k=i+4}^{j-6} Q(i+2,k-1)Q^1(k,j-2) \end{aligned} \tag{A.7}$$

The 4 for loops in this make this an expensive computation as the number of bases gets very high. However, these for loops are very similar to the partition function in structure. Indeed, we could perhaps replace each of them with a function of the form $Q^m(i,j)$ defined as

$$Q^m(i,j) = \sum_{k=i+3}^{j-5} Q(i+1,k-1)Q^1(k,j-1) \tag{A.8}$$

Which would simplify the previous sum to a constant time computation, provided we have memoized Q^m:

$$\begin{aligned} Q'(i,j) = Z_H(i,j) + Z_S(i,j)Q'(i+1,j-1) + QBI(i,j) \\ + e^{-\frac{a+c}{RT}} Z_{ND}(j,i)Q^m(i,j) \\ + e^{-\frac{a+b+c}{RT}} Z_{3'D}(j,i)Q^m(i+1,j) \\ + e^{-\frac{a+b+c}{RT}} Z_{5'D}(j,i)Q^m(i,j-1) \\ + e^{-\frac{a+2b+c}{RT}} Z_{DD}(j,i)Q^m(i+1,j-1) \end{aligned} \tag{A.9}$$

Now there just needs to be a way to efficiently compute Q^m. First we substitute in the expanded version of Q^1:

$$\begin{aligned} Q^m(i,j) = \sum_{k=i+3}^{j-5} \sum_{k'=k+1}^{j-1} Q(i+1,k-1)e^{-\frac{b(j-k')}{RT}} \Bigg[& e^{-\frac{c}{RT}} Z_{ND}(k,k')Q'(k,k') \\ & + e^{-\frac{b+c}{RT}} Z_{5'D}(k+1,k')Q'(k+1,k') \\ & + e^{-\frac{b+c}{RT}} Z_{3'D}(k,k'-1)Q'(k,k'-1) \\ & + e^{-\frac{2b+c}{RT}} Z_{DD}(k+1,k'-1)Q'(k+1,k'-1) \Bigg] \end{aligned} \tag{A.10}$$

Then we do as before and seperate out the jth term of the second sum. Note that there seems to be an additional sum needed to account that I've decreased the first sum's endpoint to $j-6$, but the sum ends up being from $k'=j-4$ to $k'=j-2$ and since Q' for bases less than 4 apart is 0 due to hairpin loop rules, this sum is equal to zero.

$$
\begin{aligned}
Q^m(i,j) = \sum_{k=i+3}^{j-6} \sum_{k'=k+1}^{j-2} Q(i+1,k-1) e^{-\frac{b(j-k'-1)}{RT}} \Bigg[& e^{-\frac{c}{RT}} Z_{ND}(k,k') Q'(k,k') \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(k+1,k') Q'(k+1,k') \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(k,k'-1) Q'(k,k'-1) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(k+1,k'-1) Q'(k+1,k'-1) \Bigg] \\
+ \sum_{k=i+3}^{j-5} Q(i+1,k-1) \Bigg[& e^{-\frac{c}{RT}} Z_{ND}(k,j-1) Q'(k,j-1) \\
& + e^{-\frac{b+c}{RT}} Z_{5'D}(k+1,j-1) Q'(k+1,j-1) \\
& + e^{-\frac{b+c}{RT}} Z_{3'D}(k,j-2) Q'(k,j-2) \\
& + e^{-\frac{2b+c}{RT}} Z_{DD}(k+1,j-2) Q'(k+1,j-2) \Bigg]
\end{aligned}
\tag{A.11}
$$

The double sum is again going to be equal to $Q^m(i,j-1)e^{-b/RT}$, and the second sum can be made much more efficient by our heuristic.

$$
\begin{aligned}
Q^m(i,j) = Q^m(i,j-1)e^{-b/RT} &+ \sum_{k(j-1)} \Bigg[Q(i+1,k-1) e^{-\frac{c}{RT}} Z_{ND}(k,j-1) Q'(k,j-1) \\
&\quad + Q(i+1,k-2) e^{-\frac{b+c}{RT}} Z_{5'D}(k,j-1) Q'(k,j-1) \Bigg] \\
&+ \sum_{k(j-2)} \Bigg[Q(i+1,k-1) e^{-\frac{c}{RT}} Z_{3'D}(k,j-2) Q'(k,j-2) \\
&\quad + Q(i+1,k-2) e^{-\frac{b+c}{RT}} Z_{DD}(k,j-2) Q'(k,j-2) \Bigg]
\end{aligned}
\tag{A.12}
$$

Since the ks for any individual j are found to be quite limited, the final form should be much more efficient at computing the $Q'(i,j)$.

Note that for Q and in many places for Q', instead of a sum over the known k that could possibly begin a leftmost pair, we see a double sum. One of them over k that could end a leftmost pair, and this sum is limited to a certain length below j. This is just making the same assumption that the internal loop computation makes: there are not arbitrarily long strands without base pairs, after a certain number of bases it becomes overwhelmingly more likely to make a base pair that we can virtually ignore the energy of the the cases of length beyond a certain L.

As for the seocnd sum, since the number of probable pairs for a base i has been shown empirically to be roughly constant, regardless of length, the second sum is essentially constant. What this all means is that all $O(n^2)$ computations of $Q(i, j)$'s are roughly constant time. This means that the overall algorithm is $O(n^2)$, an improvement over the previous algorithms asymptotic bound by and order of n.

Bibliography

[1] YE Ding, Chi Yu Chan, and Charles E Lawrence. Rna secondary structure prediction by centroids in a boltzmann weighted ensemble. *Rna*, 11(8):1157–1166, 2005.

[2] Ye Ding and Charles E Lawrence. A statistical sampling algorithm for rna secondary structure prediction. *Nucleic acids research*, 31(24):7280–7301, 2003.

[3] Robert M Dirks and Niles A Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *Journal of computational chemistry*, 24(13):1664–1677, 2003.

[4] Ivo L Hofacker, Peter Schuster, and Peter F Stadler. Combinatorics of rna secondary structures. *Discrete Applied Mathematics*, 88(1):207–237, 1998.

[5] David H Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of rna secondary structure. *Journal of molecular biology*, 288(5):911–940, 1999.

[6] John S McCaskill. The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

[7] Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

[8] Tianbing Xia, John SantaLucia, Mark E Burkard, Ryszard Kierzek, Susan J Schroeder, Xiaoqi Jiao, Christopher Cox, and Douglas H Turner. Thermodynamic parameters for an expanded nearest-neighbor model for formation of rna duplexes with watson-crick base pairs. *Biochemistry*, 37(42):14719–14735, 1998.

[9] Michael Zuker et al. On finding all suboptimal foldings of an rna molecule. *Science*, 244(4900):48–52, 1989.

[10] Michael Zuker and Patrick Stiegler. Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.