Lecture Notes

Rune Lyngsø

17th November 2005

1 RNA Secondary Structure Predction

Ribonucleic acid, or RNA, is as the name suggests a molecule very similar to 2′-deoxyribonucleic acid, or DNA. The two differences are the extra hydroxyl group on the ring-formed ribose sugar on the backbone, and the substitution of thymine with uracil as one of the four possible side chain bases. According to the central dogma, DNA $\rightarrow$ RNA $\rightarrow$ protein, RNA is restricted to a role as an intermediate messenger between the hereditary genetic medium of DNA and the biochemically active molecules of proteins. However, numerous known examples exist of RNAs with structural and catalytic importance, cf. e.g. [1, Section 10.1]. Recent discoveries even seem to indicate that RNA may play a pivotal role in gene regulation [3]. It is therefore of interest to be able to infer the structure, or structural features, for an RNA sequence. An understanding of the types of constraints determining RNA structure can also assist in better modeling of RNA sequence evolution.

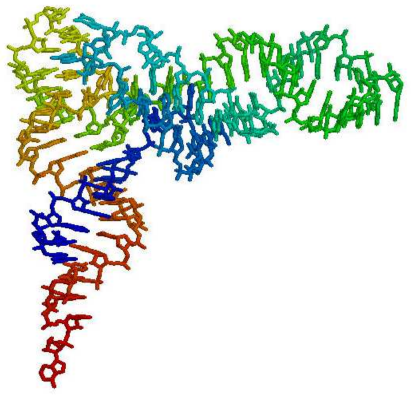
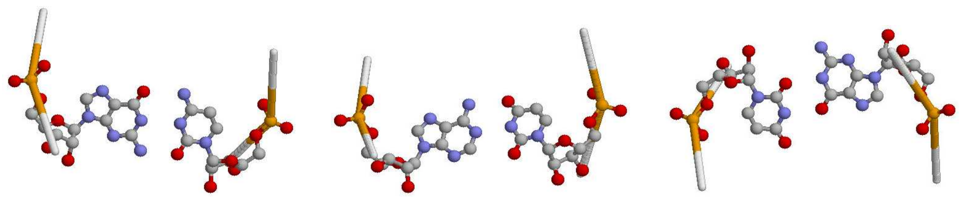


Figure 1: The tertiary structure of yeast phenylalanine tRNA

A prominent feature of RNA structures are the strong interactions formed by base pairings of complementary bases. This base pairing is similar to the Watson-Crick base pairing observed in the DNA double helix, and can clearly be seen in Fig. reffig:trna as steps in a ladder. However, where DNA base pairing is usually formed between two complementary strands, RNA base pairing is usually formed between bases in the same strand causing the RNA molecule to fold back against itself. Moreover, non-Watson-Crick base pairs are observed in RNA, with the Uracil–Guanine wobble base pair being almost as common as the Watson-Crick base pairs, cf. Fig. 2.



(a) Guanine–Cytosine (b) Adenine–Uracil (c) Uracil–Guanine

Figure 2: Base pairs commonly observed in RNA structures

The set of base pairings in the tertiary (i.e. three dimensional) structure of an RNA molecule is called the secondary structure of the RNA. An example secondary structure is illustrated in Fig. 3. Knowing the secondary structure of an RNA molecule reveals a lot of information about the overall structural conformation of the molecule. Moreover, it also reveals information about constraints on the evolution of the RNA sequence: mutation of one of the bases in a pair of base pairing positions will have to be compensated by a mutation in the other position for the base pairing to remain intact. If we already know the tertiary structure, inferring the secondary structure is easy. In the absence of a known tertiary structure, we can still obtain a very good estimate of the consensus secondary structure by the signal left by compensatory mutations if we have a well curated alignment of a large number of homologous RNA sequences, as described in [1, pp. 265–67]. Both these methods are time and graduate student demanding, and are not even always an available option (e.g. if the RNA molecule cannot be crystallised for tertiary structure determination or if no or only a few homologues are known. In this section we will focus on how to predict the secondary structure of an RNA sequence. Alternative introductions can be found e.g. in [1, Chapter 10.2] and [2].

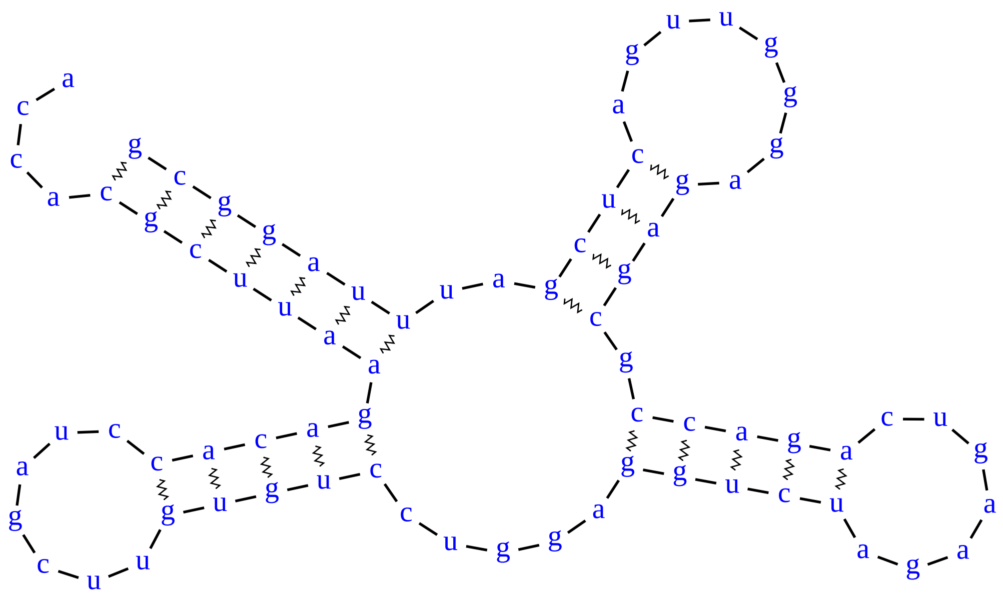


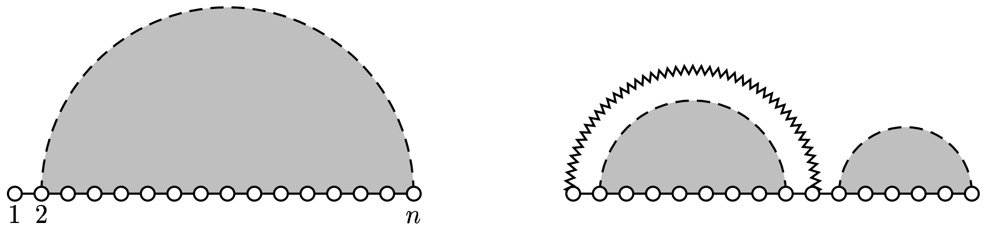
Figure 3: Secondary structure of yeast phenylalanine tRNA

1.1 Maximum Base Pairing

The reason that base pairs are such a prominent feature of RNA structures is that they form energetically quite favourable interactions known as hydrogen bonds. So the more base pairs a structure contains, the more hydrogen bonds are formed. A first attempt to predict the structure of an RNA sequence could thus be to find a structure having a maximum number of base pairs. This is also known as the Nussinov algorithm [4].

Given an RNA sequence s of length n, we do not know what the optimum structure looks like. But as illustrated in Fig. 4 we know that the leftmost base will be in one of two configurations: it will either be unpaired, or it will be paired to some other base. So the optimum structure for s will either be the optimum structure for $s[2..n]$ with s[1] left unpaired, or it will have $s[1]$ base paired with $s[k]$ for some $k \in \{2, \dots, n\}$[1]combined with the optimum structure for $s[2..k-1]$ and the optimum structure for $s[k+1..n]$. In other words, if we let N(i, j) denote

[1]Due to physical constraints, two bases separated by less than three other bases cannot get into a confi guration where they can form a base pair. Henceforth we will include this constraint in the recursions and algorithms we develop. In the current situation, the constraint implies that we need only consider $k \in \{5, \dots, n\}$.



(a) First base unpaired

(b) First base forms base pair with some other base

Figure 4: Possible configurations of the leftmost base in a secondary structure. In the graphical notation we will use, a zigzagged line indicates a base pair, a dashed line simply encloses a region without making any statements about the presence or absence of a base pairing, and a grey shaded region indicates a region with unknown secondary structure.

the number of base pairs in an optimum structure for $s[i..j]$, then

$$N(i,j) = \begin{cases} 0 & \text{if } j \leq i+3 \\ \max \begin{cases} N(i+1,j) \\ \max\limits_{\substack{5 \leq k \leq n \\ s[1] \equiv s[k]}} \{1 + N(i+1,k-1) + N(k+1,j)\} \end{cases} & \text{otherwise} \end{cases} \tag{1}$$

where the notation $s[1] \equiv s[k]$ indicates that $s[1]$ and $s[k]$ can form a legal base pair (i.e. one of the three types of base pairs in Fig. 2). It should hardly come as a surprise that we can transform this recursion into a dynamic programming algorithm for finding the number of base pairs of an optimum structure for a given RNA sequence s. One example is Algorihtm 1 where we compute the maximum number of base pairs for substrings of s in order of increasing length.

Determining the complexity of Algorithm 1 is relatively straightforward. We have three nested loops, each of which is executed $\mathrm{O}(n)$ times. So the total number of operations performed by the algorithm is $\mathrm{O}\left(n^3\right)$. The same complexity can also be observed from Eq. (1) – we have $\mathrm{O}\left(n^2\right)$ recursive elements that each is the maximum over $\mathrm{O}(n)$ values.

So far this only allows us to compute the score of an optimum structure, but not to actually predict a secondary structure for s by finding an optimum structure. As you have seen several times in this course already (e.g. the Sankoff algorithm and pairwise alignment), once we have described how to determine the score of an

Algorithm 1 Maximum number of base pairs

```
for $l = -1$ to $3$ do
  for $i = 1$ to $n - l$ do
    $N(i, i + l) = 0$
for $l = 4$ to $n - 1$ do
  for $i = 1$ to $n - l$ do
    $N(i, i + l) = N(i + 1, i + l)$
    for $k = i + 4$ to $i + l$ do
      if $s_i$ and $s_k$ can form a base pair then
        $N(i, i+l) = \max\{N(i, i+l), 1 + N(i+1, k-1) + N(k+1, i+l)\}$
```

optimum configuration it is easy to find an optimum configuration by backtracking the score. All we need to do is repeatedly ask how an optimum score was obtained at a particular point. This principle is formalised in Algorithm 2. An example of the dynamic programming table and backtrack route computed by Algorithms 1 and 2 is illustrated in Fig. 5.

Algorithm 2 `backtrack`(i, j): Backtrack maximum number of b.p. for $s[i..j]$

```
if $N(i, j) = 0$ then
  No base pairs in optimum structure for $s[i..j]$
else
  if $N(i, j) = N(i + 1, j)$ then
    backtrack$(i + 1, j)$
  else
    for $k = i + 4$ to $j$ do
      if $s[i]$ and $s[k]$ can form a base pair and $N(i, j) = 1 + N(i+1, k-1) +$
      $N(k + 1, j)$ then
        Report base pair $i \cdot k$
        backtrack$(i + 1, k - 1)$
        backtrack$(k + 1, j)$
```

Structure prediction based on just maximising the number of base pairs is too simplistic to yield acceptable results. Just one important consideration left out with this simple score function is base pair stacking. A closer inspection of RNA secondary structures will reveal that base pairs almost always occur stacked onto other base pairs. In fact, stacked base pairs is almost exclusively the only structural element providing a stabilising effect to the structure in the free energy approach that we will discuss in the next section. We will now briefly present an algorithm for finding the structure with the maximum number of base pair stacks, i.e. pairs

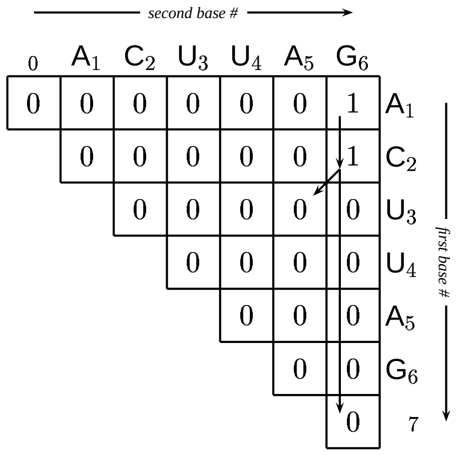


Figure 5: The dynamic programming table and backtrack tree computed for the sequence ACUUAG. The structure computed is $C_2 \cdot G_6$.

of neighbouring base pairs $i \cdot j$ and $(i+1) \cdot (j-1)$. This algorithm will have a structure almost identical to the free energy based prediction we will discuss in the next section, but without all the clutter of detailed weighting of structural elements.

When counting base pair stackings, we cannot quite use the simple recursion of Eq. (1) illustrated in Fig. 4. When postulating a base pair between $s[1]$ and $s[k]$, its score depends on whether we also have a base pair between $s[2]$ and $s[k-1]$. Hence, in our recursion we need to keep track of whether a base pair is present between the two flanking bases of the substrings. So if we let $V(i,j)$ denote the optimum number of base pair stackings for $s[i..j]$ under the constraint that $s[i]$ and $s[j]$ forms a base pair, and $W(i,j)$ denote the optimum number of base pair stackings for $s[i..j]$ for unconstrained structures, we obtain the recursion

$$V(i,j) = \begin{cases} -\infty & \text{if } s[i] \text{ and } s[j] \text{ cannot form a base pair} \\ \max \begin{cases} W(i+1,j-1) \\ 1 + V(i+1,j-1) \end{cases} & \text{otherwise} \end{cases} \tag{2}$$

$$W(i,j) = \begin{cases} 0 & \text{if } j \leq i+3 \\ \max \begin{cases} W(i+1,j) \\ \max\limits_{5 \leq k \leq n} \{V(i,k) + W(k+1,j)\} \end{cases} & \text{otherwise} \end{cases} \tag{3}$$

Based on this recursion, algorithms similar to Algorithms 1 and 2 can be devised for finding a structure with a maximum number of base pairs. The complexity analysis is also similar, this time involving the computation of two sets of $\mathrm{O}\left(n^2\right)$ recursive elements, each taking time $\mathrm{O}\left(n\right)$ to compute. In conclusion, we can find a structure with the maximum number of base pair stackings using about twice the amount of time that was required to find a structure with the maximum number of base pairs, i.e. in time $\mathrm{O}\left(n^3\right)$.

1.2 Energy Based Prediction

Prediction based on maximising number of base pair stacks doesn't is still much too simplistic to consistently provide secondary structures remotely close to the true secondary structures of RNA sequences. In [5] a model of estimating the free energy of an RNA structure based on a decomposition into loops was introduced. This decompositions is illustrated in Fig. 6. Thermodynamics state that the most stable structure of a molecule, or more generally configuration of a system, is the one with minimum free energy. Hence, a predictor of secondary structure will be the structure with minimum free energy.

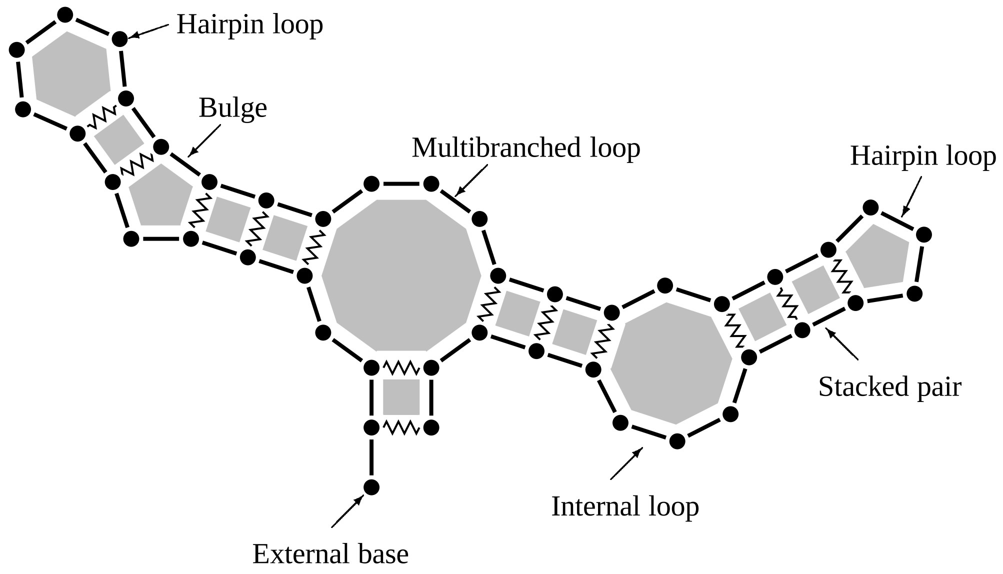


Figure 6: An example decomposition of an RNA secondary structure into its constituent loops. Each loop is represented by a grey polygon, and a representative of each type of loop is indicated.

The model used in [5] postulates that the free energy of an RNA secondary structure $\mathcal{S}$ is simply the sum of independent free energies of each of its constituent loops, i.e. that

$$\text{Energy}(\mathcal{S}) = \sum_{\text{loop} \in \mathcal{S}} \text{Energy}(\text{loop}) . \tag{4}$$

This allows a general recursion scheme for computing the minimum free energy of any structure for an RNA sequence s. As for finding the maximum number of base pair stackings, the central recursive element is the optimum value (i.e. in this context the minimum free energy) that we can obtain for a substring $s[i..j]$ when we require $s[i]$ and $s[j]$ to form a base pair. The base pair $i \cdot j$ closes some loop containing zero or more other base pairs $\{i_l \cdot j_l\}_{1 \leq l \leq k, k \geq 0}$. If the structure on $s[i..j]$ is an optimum structure, then each of the substructures on $s[i_l..j_l]$ for $1 \leq l \leq k$ have to be optimum structures. If we let $vx(i, j)$ denote the minimum free energy taken over all structures on the substring $s[i..j]$, then vx must obey the recursion

$$vx(i,j) = \min_{\substack{k \geq 0 \\ i < i_1 < j_1 < \ldots < i_k < j_k < j}} \left\{ \text{Energy}\,(i \cdot j; i_1 \cdot j_1, \ldots, i_k \cdot j_k) + \sum_{l=1}^{k} vx(i_l, j_l) \right\} , \tag{5}$$

where $\text{Energy}\,(i \cdot j; i_1 \cdot j_1, \ldots, i_k \cdot j_k)$ is the free energy of the loop defined by the base pairs $i \cdot j, i_1 \cdot j_1, \ldots i_k \cdot j_k$. This general recursion does not allow an efficient solution, as we for each substring need to consider $2^{\mathrm{O}(n)}$ possible loops it can close. The main culprit is multibranched loops, as there are only $\mathrm{O}\left(n^2\right)$ possible internal loops, $\mathrm{O}\,(n)$ possible bulges, 1 possible stacked pair, and 1 possible hairpin loop. As we shall presently see, the function assigning energies to multibranched loops has been chosen in such a way that they can be handled efficiently.

Over the years, parameters of the general loop decomposition model have steadily been refined by calorimetric measurements and other experiments, and by optimisation based on known RNA secondary structure. Briefly summarising, the energy functions and their dependencies are

- $eH(i \cdot j)$ is the energy of a hairpin loop closed by the base pair $i \cdot j$. It is a sum of a function depending on the size of the loop, i.e. $j - i$, and stacking interactions between the base pair $i \cdot j$ and the two neighbouring unpaired bases $s[i+1]$ and $s[j-1]$. The size dependence has been experimentally tabulated for small loops, and a general theoretically derived and experimentally fitted formula is used for larger loop sizes. The stacking effect has been experimentally tabulated for all combinations of base pair type and types of the two neighbouring unpaired bases. The energy of certain small loops have been experimentally tabulated depending on the exact base sequence occurring in the loop.

- $eS(i \cdot j, i+1 \cdot j+1)$ is the energy of stacking base pair $s[i] \cdot s[j]$ onto the base pair $s[i+1] \cdot s[j-1]$. It depends on the types of the two base pairs and has been experimentally tabulated for all such.

- $eL(i \cdot j, k \cdot l)$ is the energy of an internal loop or bulge defined by the base pairs $s[i] \cdot s[j]$ and $s[k] \cdot s[l]$ (with $i < k < l < j$). It is a sum of functions depending on the size of the loop, i.e. $k-i+j-l$, the asymmetry of the loop, i.e. $k-i$ and $j-l$, and stacking interactions between the base pair and its two neighbouring unpaired bases for each of the two base pairs in the loop. Size dependence has been experimentally tabulated for small loops and a theoretically derived and experimentally fitted function is used for larger loops. The asymmetry function has a heuristic form with parameters optimised for known structures – it mostly depends on the lopsidedness of the loop, i.e. the difference between the two parameters. Stacking effects have been experimentally tabulated for all combinations of base pair type and types of the two neighbouring unpaired bases. The energy of certain small loops have been experimentally tabulated depending on the exact bases occurring in the loop.

- $eM(i \cdot j; i_1 \cdot j_1, \ldots, i_k \cdot j_k)$ is the energy of a multibranched loop defined by the base pairs $i \cdot j$, $i_1 \cdot j_1$, ..., $i_k \cdot j_k$. It is a sum of stacking interactions between the base pairs and their neighbouring unpaired bases and between base pairs separated by at most one unpaired base, and an affine function of the number of unpaired bases and base pairs in the loop, i.e. a term $a + bk + \sum_{l=0}^{k} c(i_{l+1} - j_l - 1)$ where we for convenience have defined $j_0 = i$ and $i_{k+1} = j$. The stacking interactions have been experimentally tabulated for combinations of base pair types and types of neighbouring unpaired bases and base pairs. The parameters a, b, and c for the loop size dependence have been optimised for known structures.

- Base pairs with neighbouring external bases incur a further stacking contribution similar to the ones seen for the base pairs in the different types of loops. External base pairs not neighbouring a base pair have no contribution to a structures free energy.

The simple affine function for multibranched loops allow these to be handled much more efficiently than if we had to consider each possible multibranched loop independently. The change in energy of adding an extra base pair or unpaired base to a loop does not depend on the location, or even the number, of other base pairs and unpaired bases in the loop, but only on the parameters b and c. This means that we can recursively build the interior of a multibranched loop by adding an unpaired base or a base pair to an existing interior. This is formalised in the recursive element

wx_I in the following set of recursions for determining the minimum free energy of any structure for s. The main recursive element vx captures the free energies of structures where the two flanking bases form a base pair, and wx allows us to add external bases and create multifurcating structures, i.e. structures consisting of two or more independent substructures. For brevity of recursions we have left out the stacking contributions for multibranched loops and external bases. These can be added with a bit of care without significantly changing the time complexity of the resulting algorithm.

$$wx(1,i) = \begin{cases} 0 & \text{if i < 1} \\ \min\{wx(1,i-1), \min\limits_{1\leq j<i}\{wx(1,j-1)+vx(j,i)\}\} & \text{otherwise} \end{cases} \tag{6}$$

$$vx(i,j) = \begin{cases} \infty & \text{if } s[i] \text{ and } s[j] \text{ cannot form a base pair} \\ \min \begin{cases} eH(i,j) \\ eS(i\cdot j, i+1\cdot j-1) + vx(i+1,j-1) \\ \min\limits_{\substack{i<k<l<j \\ k-i+j-l>2}} \{eL(i\cdot j, k\cdot l) + vx(k,l)\} \\ \min\limits_{i<k<j}\{a + wx_I(i+1,k-1) + wx_I(k,j-1)\}\} \end{cases} & \text{otherwise} \end{cases} \tag{7}$$

$$wx_I(i,j) = \begin{cases} \infty & \text{if } j < i \\ \min \begin{cases} b + vx(i,j) \\ c + wx_I(i,j-1) \\ c + wx_I(i+1,j) \\ \min\limits_{i<k<j}\{wx_I(i,k-1) + wx_I(k,j)\}\} \end{cases} & \text{otherwise} \end{cases} \tag{8}$$

We can define a dynamic programming algorithm for solving these recursions in a way similar to what we saw in the previous section. The minimum free energy taken over all possible structures for s will be the final value of $wx(1,n)$. Once we have computed the minimum free energy, we can find a structure with this free energy by backtracking how we obtained the value of $wx(1,n)$, similar to the backtracking of the maximum number of base pairs of Algorithm 2.

Is it more expensive to find a minimum free energy structure than a maximum number of base pairs structure? Computing the wx elements requires computing $\mathrm{O}(n)$ elements each taking time $\mathrm{O}(n)$, i.e. $\mathrm{O}(n^2)$ time. Computing the vx elements requires computing $\mathrm{O}(n^2)$ elements. The most time consuming part of this computation will be minimising over all internal loops, as this requires minimising over all k,l between i and j which takes $\mathrm{O}(n^2)$ time. This results in an overall time complexity of $\mathrm{O}(n^4)$. Usually this is reduced by observing that as all known internal loops are relatively small it is safe to limit the size of loops considered to a constant (usually 30). Even when allowing loops of arbitrary size, with the current form of eL more involved techniques are known for handling internal loop contributions to the vx entries in time $\mathrm{o}\left(n^3\right)$. Using either of these strategies, the most time consuming part of computing a vx entry becomes minimising over all possible ways to split the interior of a multibranched loop into two parts each containing at least one base pair. This has complexity $\mathrm{O}\left(n\right)$, resulting in an overall complexity of $\mathrm{O}\left(n^3\right)$ for computing the vx entries. Finally, there are $\mathrm{O}\left(wx_I\right)$ entries, where the most time consuming part of computing an entry is constructing a structure by joining two substructures which has time complexity $\mathrm{O}\left(n\right)$. In total, computing the minimum free energy can still be done in time $\mathrm{O}\left(n^3\right)$.

Apart from a more refined model allowing much better structure predictions, and advantage of a well parameterised free energy model for RNA secondary structures is that we can borrow from the theory of thermodynamics. The Boltzmann distribution states that the probability of observing a particular configuration $\mathcal{S}$ (e.g. secondary structure) from a set of possible configurations Ω is proportional to $e^{-\,\mathrm{Energy}(\mathcal{S})/kT}$, where T is the absolute temperature and k is a constant. If we can efficiently compute the partition function, i.e. the sum over all possible structures $\mathcal{S}$ of $e^{-\,\mathrm{Energy}(\mathcal{S})/kT}$, then we can efficiently compute the probability of observing any particular structure. Without going into too much detail, this essentially boils down to replacing minimums in Eqs. (6)-(8) with sums (as we need to sum the contribution from all possible choices rather than just choose the optimum) and sums with multiplications (as the exponential of a sum equals the product of the exponentials).

One small problem remains, though. Whereas $\min\{x, x\} = x$, it is not generally the case that $x + x = x$. Therefore we need to be careful only to consider each possible structure exactly once. Eqs. (6)-(8) fail to do this where we combine two wx_I entries, as there in general is more than one possible way to split a structure into two substructures, and where we add unpaired bases in the recursion for wx_I elements, as we can choose first to add an unpaired base to the left and then an unpaired base to the right or vice versa. Recursions can be developed remedying this problem, e.g. replacing Eqs. (7)-(8) with

$$vx(i,j) = \begin{cases} \infty & \text{if } s[i] \text{ and } s[j] \text{ cannot form a base pair} \\ \min \begin{cases} eH(i,j) \\ eS(i\cdot j, i+1\cdot j-1) + vx(i+1,j-1) \\ \min\limits_{\substack{i<k<l<j \\ k-i+j-l>2}} \{eL(i\cdot j, k\cdot l) + vx(k,l)\} \\ a + wx_I(i+1,j-1) \end{cases} & \text{otherwise} \end{cases} \tag{9}$$

$$wx_I(i,j) = \begin{cases} 0 & \text{if } j<i \\ \min \begin{cases} c + wx_I(i,j-1) \\ \min\limits_{i<k<j} \{wx'_I(i,k-1) + b + vx(k,j)\} \\ \min\limits_{i<k<j} \{wx_I(i,k-1) + b + vx(k,j)\} \end{cases} & \text{otherwise} \end{cases} \tag{10}$$

$$wx'_I(i,j) = \begin{cases} 0 & \text{if } j<i \\ \begin{cases} \min\limits_{i\le k<j} \{(k-i)c + vx(k,j)\} \\ c + wx'_I(i,j-1) \end{cases} & \text{otherwise} \end{cases} \tag{11}$$

In this revised equation system wx'_I represents parts of interiors of multibranched loops contributing exactly one base pair to the loop, while wx_I represents possible interiors of multibranched loops, i.e. substructures that would contribute at least two base pairs to the multibranched loop.

For any sequence of reasonable length, there will be so many competing structures, many differing in energy from the optimum structure only by a small amount, that the probability of observing any structure, even the optimum one, is vanishingly small. So if all the information we could obtain from Boltzmann distributions was the probability of a particular structure, it would be of little use. However, $vx(i,j)$ (modified to computation of partition function instead of minimum free energy) holds the partition function for the subsequence $s[i..j]$ under the constraint that $s[i]$ and $s[j]$ form a base pair. Being a bit careful with how the sequence ends are treated, we can set up similar recursions for computing $vx(j,i)$, i.e. the partition function for s with the subsequence $s[i..j]$ extricated and under the constraint that $s[i]$ and $s[j]$ form a base pair. Multiplying $vx(i,j)$ and $vx(j,i)$ and dividing by the full partition function, i.e. $vx(i,j)vx(j,i)/wx(1,n)$, we get the probability of observing a structure containing the base pair $s[i]\cdot s[j]$. This allows us to compute the probability of observing all possible base pairs, still in time O $\left(n^3\right)$

2 Stochastic Context Free Grammars

Grammars denote systems that allow us to grow entities from successive applications of replacement rules, also known as productions. The type of grammar depends on the form the productions are allowed to take. Context free grammars can be specified as quadruples (V, Σ, P, S) where

- V is a finite set of vaiables.
- Σ is a finite set of terminal symbols, also called the alphabet of the grammar.
- P is a set of productions on the form $A \to x$ where $A \in V$ and $x \in (V \cup \Sigma)^*$, i.e. the lefthand side if the production is a variable and the righthand side is a finite (possibly empty) string of variables and terminal symbols.
- $S \in V$ is a distinguished start variable.

Starting from the string S we are now allowed to keep applying productions as long as there are variables in our string. I.e. if our current string is xAy we can replace it with xzy if $A \to z \in P$. The strings we can generate from a grammar are all the finite string containing no variables that can be obtained in this way.

Example 1 (Palindromes) *Consider the grammar G with just the single variable S, alphabet a, b, start symbol S and productions*

$$S \to aSa \mid bSb \mid \epsilon\,, \tag{12}$$

where $|$ denotes a choice between different productions for S and $epsilon$ the empty string (i.e. in quadruple notation $G = (\{S\}, \{a, b\}, \{S \to aSa, S \to bSb, S \to \epsilon\}, S)$). This grammar generates all the even length palindromic strings, i.e. strings that read the same when read forward and backward, of a's and b's. E.g. the string $abba$ can be derived by the sequence of replacements $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$, where each of the productions incidentally are used exactly once.

Example 2 (RNA secondary structure) *The productions of a grammar consists of a single element on the lefthand side, and a string of elements on the right hand side. This structure is very similar to recursions. This observation allow us an easy way to derive a grammar generating RNA sequences by mimicking their secondary structure from Eqs. (6)-(8). Each recursive element gives rise to a variable in the grammar, and each choice in the recursions gives rise to a production. As it turns out, we also need a variable that can generate strings of unpaired bases* *of arbitrary length. With these considerations, we get a grammar with variables $\{V, W, I, U\}$, alphabet $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{U}\}$, productions*

$$W \to \epsilon \mid Wa \mid WV \tag{13}$$
$$V \to bU\bar{b} \mid bV\bar{b} \mid bUVU\bar{b} \mid bII\bar{b} \tag{14}$$
$$I \to V \mid Ia \mid aI \mid II \tag{15}$$
$$U \to \epsilon \mid aU\,, \tag{16}$$

where a denotes an unpaired base and $b, \bar{b}$ denotes a pair of bases forming a base pair. The start variable is W. This grammar doesn't quite reflect the structure of Eqs. (6)-(8), as hairpin loops are not guaranteed to contain at least three unpaired bases and the production reflecting internal loops and bulges, $V \to bUVU\bar{b}$, can also generate a stacked pair. These minor details can be rectified by a slightly more detailed handling of stretches of unpaired bases in the respective types of loops.

Given any context free grammar, algorithms exists for systematically converting it to a context free grammar in Chomsky normal form (CNF) that generates exactly the same strings as the original grammar. In a grammar on CNF, all productions are of one of the following three types:

- $S \to \epsilon$, i.e. the start variable is replaced by the empty string.
- $A \to a$ where $a \in \Sigma$, i.e. a variable is replaced by a terminal symbol.
- $A \to BC$ where $B, C \in V \setminus \{S\}$, i.e. a variable is replaced with two variables, neither of them being the start variable.

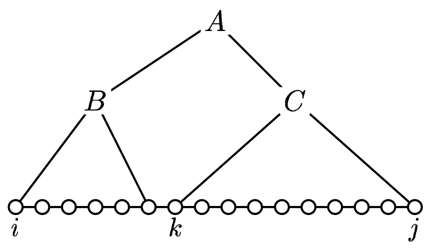


Figure 7: Illustration of the third possibility in Eq. (17). We can replace A with BC where B generates a prefix of the sequence and C generates the remaining suffix of the sequence.

Given a grammar $G = (V, \Sigma, P, S)$ on CNF we can determine whether it can generate a sequence s by an algorithm that in its structure is very similar to the RNA secondary structure prediction algorithms we saw in the previous section. The central element is determining whether substrings of s can be generated starting from single variables of G. If $\alpha(A, i, j)$ denotes whether we can generate $s[i..j]$ starting from the variable $A \in V$, then α obeys the following recursion:

$$\alpha(A, i, j) = \begin{cases} \textbf{true} & \text{if } A \to \epsilon \in P \text{ and } j = i - 1 \\ \textbf{true} & \text{if } j = i \text{ and } A \to s[i] \in P \\ \textbf{true} & \text{if } \exists i < k \leq j, A \to BC \in P : \alpha(B, i, k-1) \wedge \alpha(C, k, j) \\ \textbf{false} & \text{otherwise} \end{cases} \tag{17}$$

Essentially the recursion says that we can generate a subsequence from a variable either if we can generate it directly using one of the productions of P, or if we can replace it with two other variables where the first can generate a prefix of the subsequence and the second can generate the remaining suffix of the subsequence. This last possibility is illustrated in Fig. 7

A stochastic context free grammar is a context free grammar with a function $Pr : P \mapsto \mathbb{R}$ that assigns a probability to each production. We will usually require that the probabilities are normalised for each variable, i.e. that $\forall A \in V : \sum_{A \to x \in P} Pr(A \to x) = 1$. Hence Pr can be seen as defining the probability distribution from which we sample whenever we replace a variable. Given a stochastic context free grammar G on CNF, it is not too difficult to see how we can modify Eq. (17) to compute the maximum probability of any single derivation of a sequence s, or the total probability of generating s by G: simply replace the Boolean values with the appropriate Pr values, replace the $\wedge$ with multiplication and either take the maximum or sum over all possible choices, depending on whether it is the maximum or total probability that needs to be determined.

Transforming a general stochastic context free grammar to a stochastic context free grammar on CNF that generates sequences with the same probabilities is not altogether trivial. For one thing, the way to assign probabilities in the new grammar depends on whether the grammar should give the same maximum probability of deriving a sequence, or the same total probability of deriving a sequence. In the latter case, assigning the right probabilities is in general hard.

Given a grammar and data with either known or unknown derivations in the grammar, we can estimate train the grammar, i.e. find probabilities of the productions making the data more likely to have been generated by the grammar. This is described in more detail in [1, Chapter 9.6]. Essentially this allows us to infer e.g. parameters from a set of known RNA molecules yielding a grammar where the most likely parse of an RNA sequence is a good predictor of its secondary structure.

References

[1] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probalistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[2] S. R. Eddy. How do rna folding algorithms work? *Nature Biotechnology*, 22:1457–1458, 2004.

[3] J. S. Mattick. RNA regulation: a new genetics? *Nature Reviews Genetics*, 5:316–323, 2004.

[4] R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied Mathematics*, 35(1):68–82, July 1978.

[5] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9:133–148, 1981.