Methods in Molecular Biology 1097

Springer Protocols

Jan Gorodkin
Walter L. Ruzzo *Editors*

RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods

Humana Press

Chapter 8

SCFGs in RNA Secondary Structure Prediction: A Hands-on Approach

Zsuzsanna Sükösd, Ebbe S. Andersen, and Rune Lyngsø

Abstract

Stochastic context-free grammars (SCFGs) were first established in the context of natural language modelling, and only later found their applications in RNA secondary structure prediction. In this chapter, we discuss the basic SCFG algorithms (CYK and inside–outside algorithms) in an application-centered manner and use the pfold grammar as a case study to show how the algorithms can be adapted to a grammar in a nonstandard form. We extend our discussion to the use of grammars with additional information (such as evolutionary information) to improve the quality of predictions. Finally, we provide a brief survey of programs that use stochastic context-free grammars for RNA secondary structure prediction and modelling.

Key words SCFGs, CYK algorithm, Inside–outside algorithm, Pfold

1 Introduction

Stochastic context-free grammars (SCFGs), also known as probabilistic context-free grammars (PCFGs), were first invented in the purely computational context of natural language modelling, and to this day are widely used in computational linguistics [1–3]. It is perhaps surprising that SCFGs found applications in RNA secondary structure modelling too – but RNA secondary structure and natural languages have many things in common.

In both cases, our concern is producing a string of symbols from a particular alphabet. In natural languages, the alphabet is made up of the vocabulary of the language; in the case of RNA secondary structure, we have nucleotides in particular structural configurations. In both cases it is also a requirement that the symbols are chosen and connected to each other according to particular rules, which can formally be described by a grammar. The rules can describe connections between distant parts of the string (sentence). For example, in the English language a verb

Jan Gorodkin and Walter L. Ruzzo (eds.), *RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods*, Methods in Molecular Biology, vol. 1097, DOI 10.1007/978-1-62703-709-9_8,

must always be connected to a subject, but they need not be immediately adjacent to each other. In RNA secondary structure, any pairing nucleotide must have its pairing partner somewhere in the structure.

Having a grammar to describe a language enables us to formally ask and answer questions about the language itself: for example, we can determine whether a particular string can be produced in the language or not. We can also generate arbitrary strings that fulfill the criteria of the language. Extending the grammar with probabilities, we can ask how likely a sentence is to be produced in the language, or the probability with which two particular words occur in a particular configuration in a sentence.

In this chapter, we will introduce the basic formalism of these ideas and describe how they can be used for the prediction and modelling of RNA secondary structure. Our goal here is an introductory, "hands-on" approach, with focus on providing an intuitive understanding of the subject and practical examples of its use, rather than formal proofs. For a more rigorous treatment of the subject, the reader is advised to consult the references listed at the end of this chapter.

2 What Is an SCFG?

A grammar (also known as formal grammar or transformational grammar) can be thought of as a framework for producing particular types of strings using "placeholders," which turn into symbols from the alphabet according to predefined rules. A grammar thus consists of: an alphabet (terminal symbols, which make up the words of the language), variables (nonterminal symbols, which function as the "placeholders"), and rewriting rules (productions). A string is a concatenation of terminal and nonterminal symbols. A production is of the form $\alpha \rightarrow \beta$, where in general, α and β can be any strings, with the restriction that α must contain at least one nonterminal symbol. The function of a grammar is to convert, in a stepwise fashion, a single nonterminal symbol (the designated start symbol) into a string of terminals from the alphabet of the grammar.

Definition 1: A sequence generating grammar is a tuple (V, Σ, P, S) where

- V is a finite set of nonterminal symbols (variables), conventionally denoted by uppercase letters
- Σ is a finite set of terminal symbols (alphabet), conventionally denoted by lowercase letters

- P is a finite set of productions of the type $\alpha \to \beta$, where $\alpha \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ and $\beta \in (V \cup \Sigma)^*$.[1]
- $S \in V$ is the start symbol

An example of a grammar is the Knudsen–Hein grammar, which is used in the pfold program for RNA secondary structure prediction.

Example 1: The Knudsen–Hein grammar contains three nonterminal symbols: $\{S, L, F\}$ (of which S is the start symbol), the alphabet: $\{s, d\}$, and six production rules. Alternative productions from the same left-hand side are separated by | ("or").

$$\begin{aligned} S &\to L \mid LS \\ L &\to s \mid dFd \\ F &\to dFd \mid LS \end{aligned}$$

Generating a single s (representing a single unpaired nucleotide) can be done as:

$$S \Rightarrow L \Rightarrow s$$

A small stem-loop could be generated as:

$$S \Rightarrow L \Rightarrow dFd \Rightarrow dLSd \Rightarrow dsSd \Rightarrow dsLd \Rightarrow dssd$$

Note that the derivation connects the two d's to each other grammatically. The first d is like an opening paranthesis with the second d as its closing pair; this is indeed the basis of the dot-bracket representation of RNA secondary structures. In the case of the KH grammar, the semantic interpretation of a derivation is the secondary structure itself.

We will use the grammar from Example 1 throughout this chapter, and will in the following refer to it as the KH grammar for brevity.

Producing a string using a grammar is also known as "deriving" or "parsing" the string. A useful way of representing the parse of a string is a parse tree (*see* Fig. 1). Two derivations of a string differ only if they have different parse trees. If there exists a string that can be produced in more than one way by the grammar, the grammar is said to be *syntactically ambiguous*. As the same RNA sequence can usually fold into many different secondary structures, RNA secondary structure grammars are often syntactically ambiguous by design. A one-to-one correspondence between parse trees and RNA secondary structures is also desirable; in contrast, if the same secondary structure can be represented by more than one parse tree, then the grammar is said to be *semantically ambiguous*.

[1] The symbol * is the Kleene star operator; it is a widely used regular expression and denotes any (possibly empty) string that is produced by concatenating elements drawn from the set. The elements can occur any number of times and in any order in this string.

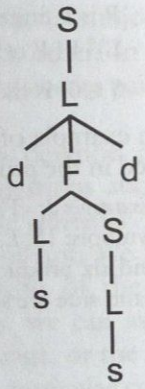

Fig. 1 The parse tree of the derivation of the string *dssd* using the KH grammar. This string can only be derived in one way

Example 2: The KH grammar is syntactically ambiguous. For example, the string *dssdssdssd* can be parsed in two different ways; the details are left to the reader as an exercise. However, the KH grammar is semantically unambiguous, because there is a one-to-one correspondence between RNA secondary structures and parse trees. Chapter 5 in this book discusses the issue of ambiguity in greater detail.

Grammars can be classified on the basis of their production rules into what is known as the Chomsky-hierarchy. Context-free grammars form a class in this hierarchy and are defined by only having a single nonterminal symbol on the left-hand side of all of the production rules. The context-free designation is because one can always rewrite the nonterminal symbol, irrespective of the context in which it occurs. This is a convenient model for RNA secondary structure, where insertions of stem-loops can happen anywhere without affecting the rest of the structure; this is known as "nesting" structures. However, context-free grammars are not naturally suited for modelling pseudoknotted structures. The KH grammar is context-free.

Note that there can be several equivalent grammars that describe the same language and can be converted to each other by rule transformations. This is illustrated for one rule of the KH grammar in Example 3 below.

Example 3: The rule $L \rightarrow dFd$ can be "expanded" to a set of equivalent rules:

$$L \rightarrow NM, \quad N \rightarrow d, \quad M \rightarrow FK, \quad K \rightarrow d.$$

Thus: $L \Rightarrow NM \Rightarrow dM \Rightarrow dFK \Rightarrow dFd$.

It is desirable to use a standard for writing equivalent grammars, such that any general grammar has an equivalent grammar in the standard. These standards are called normal forms, and the most common normal form for context-free grammars is the Chomsky normal form.

Definition 2: A context-free grammar (V, Σ, P, S) is in Chomsky normal form if all of its production rules are of the form:

- $N \to AB$ (rules of this type are called bifurcations) or
- $N \to \sigma$ or
- $S \to \epsilon$

where $N, A, B \in V$, $\sigma \in \Sigma$, ϵ is the empty string and $A, B \neq S$.

It can be proven that every context-free grammar can be written in Chomsky normal form. A grammar in Chomsky normal form is also clearly always context-free. The transformation of a given context-free grammar to Chomsky normal form, however, is not always trivial.

As noted above, the KH grammar provides alternative production rules from the same nonterminals, giving us a choice every time we need to parse a nonterminal. If we associate each of these choices with a probability, such that the total probability of the choices from the same nonterminal is 1, we get a stochastic context-free grammar.

Definition 3: A stochastic grammar is a grammar (V, Σ, P, S) extended with a weight function $w : P \mapsto \mathbb{R}$, where w for each left-hand side is a probability distribution over the possible right-hand sides it can be replaced with. The probability of a derivation is the product of the probabilities of the productions used in that derivation. A stochastic context-free grammar is a context-free grammar that is stochastic.

Derivations with higher probabilities are said to be more consistent with the grammar than derivations with lower probabilities. The probability of a string under the grammar is the sum of the possible derivations of that string under the grammar.

3 SCFG Algorithms

We are now in the position to ask some useful questions about SCFGs:

1. Can a particular string be produced by the grammar?
2. What derivation has the highest probability under the grammar?
3. How can a particular string be produced by the grammar?
4. What is the total probability of all derivations of a particular length?
5. What is the total probability under a grammar describing RNA secondary structure that two particular positions form a pair?

Efficient algorithms have been developed to answer these questions computationally.

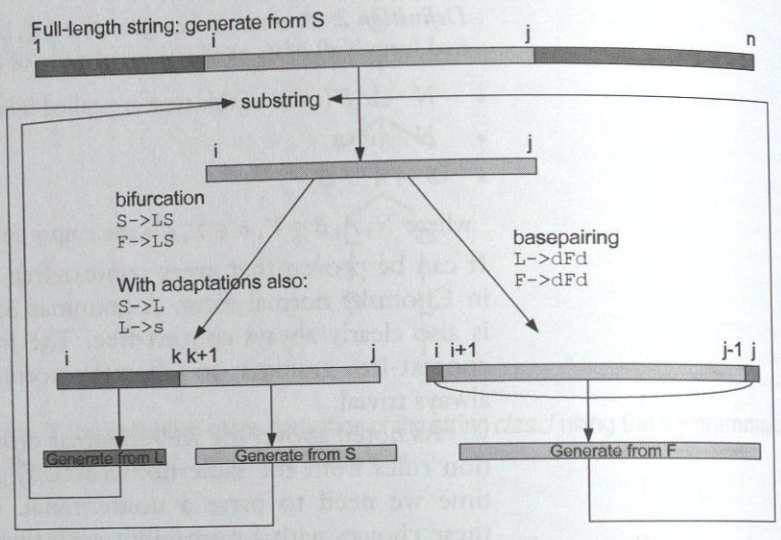

Fig. 2 To determine whether a string can be derived under the KH grammar, one must consider each subsequence recursively. Some subsequences, however, would be calculated several times in a recursive approach; for this reason, dynamic programming is used

The Basic CYK Algorithm

The Cocke-Younger-Kasami algorithm (CYK algorithm) is a parsing algorithm designed to answer the question: Can a string be derived under the grammar? The algorithm is based on the observation that a string can be derived from a nonterminal $N \in V$ if and only if the string can be derived from the right-hand side of at least one of the productions that have N on their left-hand side.

Example 4: To determine whether a string (secondary structure) $c_1 \dots c_n$ can be derived under the KH grammar, we can begin to reason intuitively:

- $c_1 \dots c_n$ can be derived if and only if one can write a sequence of productions under the grammar such that $S \to \dots \to c_1 \dots c_n$.
- Consequently, $c_1 \dots c_n$ can be derived if either a derivation $L \to \dots \to c_1 \dots c_n$ or a derivation $LS \to \dots \to c_1 \dots c_n$ exists.
- For each of those cases, we must examine the possibilities:
  - If $L \to \dots \to c_1 \dots c_n$ exists, then either the rule $L \to s$, or the rule $L \to dFd$ must be used. $L \to s$ requires $n = 1$ and $c_1 = s$. $L \to dFd$ requires that $F \to \dots \to c_2 \dots c_{n-1}$ exists, $c_1 = d$ and $c_n = d$. We need to check which one (if either) is the case.
  - For $LS \to \dots \to c_1 \dots c_n$, we need to check for every $1 \leq k < n$ whether the pair of derivations $L \to \dots \to c_1 \dots c_k$ and $S \to \dots \to c_{k+1} \dots c_n$ exists.

This argument naturally leads to a recursion (*see* Fig. 2). Let the boolean (true/false) array `P[i,j,N]` denote whether $c_i \dots c_j$ can be derived starting from nonterminal $N \in V$ of the context-free

grammar (V, Σ, P, S) (assuming it is converted to Chomsky normal form). The string $c_1 \ldots c_n$ can be produced if and only if `P[1,n,S]` is true, and the value of this element can be determined using the following recursion relation:

$$P[i,j,N] = \begin{cases} \text{TRUE if } (S \to \epsilon) \in P & i = j+1 \text{ and } N = S \\ \text{TRUE if } (N \to c_i) \in P & i = j \\ \text{TRUE if } \exists k : i \le k < j \text{ and } A, B \in V: & \\ \quad (N \to AB) \in P \text{ AND } P[i,k,A] & \\ \quad \text{AND } P[k+1,j,B] & 1 \le i < j \le n \\ \text{FALSE} & \text{otherwise} \end{cases} \quad (1)$$

A direct implementation of this recursion, however, would be very slow, as most cells would have to be visited several times. A computational trick to speed up the evaluation of `P[1,n,S]` is to do the calculations bottom-up (i.e., starting from shorter substrings and progressing to the full-length string), and store the partial results; this is known as dynamic programming. For grammars in Chomsky normal form, the CYK algorithm can be written up directly from the recursion relations:

```
1   initialize all P[i,j,N] to false

2   for i = 1 to n do:
3     for each production N -> c_i
4       set P[i,i,N] = true
5   for each j = 1 to n do:
6     for each i = 1 to j-1 do:
7       for each k = i to j-1 do:
8         for each production N -> AB do:
9           if P[i,k,A] and P[k+1,j,B] then
10            set P[i,j,N] = true

11  if P[1,n,S] then
12    string can be derived under grammar
13  else
14    string can not be derived under grammar
```

It can be shown that the CYK algorithm has a time complexity of $O(n^3|P|)$ and a space complexity of $O(n^2|V|)$, where $|P|$ is the number of rules and $|V|$ is the number of nonterminals in the grammar.

The CYK algorithm can also be applied to context-free grammars that are not in Chomsky normal form (such as the KH grammar as presented above), but this requires more thought. One approach is to convert the grammar to Chomsky normal form;

however, this often requires the creation of more nonterminal symbols and rules, increasing both the time and space complexity of the algorithm. For this reason, the CYK algorithm is typically adapted to match the format of the grammar.

Example 5: An adaptation of the basic CYK algorithm to the KH grammar would be, given an input structure $c_1 \ldots c_n$ (a string of *s*'s and *d*'s):

```
1  initialize all P[i,j,S], P[i,j,L], P[i,j,F] to
   false

2  for i = 1 to n do:
3     if c_i = 's' then set P[i,i,L] = true

4  for each j = 1 to n do:
5     for each i = 1 to j-1 do:
6        if j-i > 2 then
7           if P[i+1,j-1,F] and c_i = 'd'
              and c_j = 'd' then
8              set P[i,j,L] = true
9              set P[i,j,F] = true

10       for each k = i to j-1 do:
11          if P[i,k,L] and P[k+1,j,S] then
12             set P[i,j,F] = true
13             set P[i,j,S] = true

14       if P[i,j,L]
15          set P[i,j,S] = true

16 if P[1,n,S] = true then
17    structure can be produced by the KH grammar
18 else
19    structure cannot be produced by the KH grammar
```

It is important to note that with grammars not in Chomsky normal form, one must pay great attention to the order of calculations. For example, line 14 here should not be placed before line 8; `P[i,j,L]` must be fully evaluated before its value is needed.

The adapted CYK algorithm for the KH grammar is independent of nucleotide sequence, just like the grammar itself. To make the algorithm (and the grammar) more useful, one must introduce nucleotide-dependence. One way of doing this would be to simply introduce an extended alphabet and more production rules into the grammar (e.g., $L \rightarrow s$ would become $L \rightarrow a|c|g|u$, and $N \rightarrow dFd$ would become $N \rightarrow aFu|uFa|cFg|gFc$, assuming that only A-U and G-C pairs are allowed) and adapt the CYK algorithm to this new grammar. Equivalently, one can start the modifications at the algorithm level, as illustrated below in Example 6.

Example 6: To find out if the grammar can produce the input string $c_1 \ldots c_n$ (which now represents a sequence of nucleotides), only two lines in the previous algorithm need to be modified. Firstly, any nucleotide can be single-stranded so we can write:

```
3       set P[i,i,L] = true
```

Secondly, we can force the algorithm not to allow basepairing between nucleotides c_i and c_j unless they are complementary (e.g., A-U, G-C). We will denote complementarity between nucleotides at positions i and j as $c_i \sim c_j$. We can then write:

```
7           if P[i+1,j-1,F] and c_i ~ c_j then
```

The basic algorithm presented here only allows us to answer whether a string can be derived from a grammar at all. In our particular example above, the nucleotide-dependent, KH-grammar adapted CYK algorithm answers the question: Can the KH grammar fold this sequence? It turns out the answer will always be `true`, as any sequence will fold into some structure, in the worst case the single-stranded (unfolded) structure.

The basic CYK algorithm presented in this section might not appear to be very useful for RNA secondary structure prediction, but it does provide a versatile framework where only small modifications are required to produce other useful algorithms.

Example 7: The KH grammar (or alternatively the algorithm) can, for example, be adapted to couple sequence to structure, and answer the question: Can the KH grammar fold this particular sequence into this particular structure? To do this, one can introduce more terminal symbols into the grammar to distinguish between pairing and unpairing nucleotides – so instead of s, we would have s_a, s_c, etc., and instead of d, we would have d_a, d_c etc., and the number of rules would again increase. Alternatively, the algorithm can be modified to produce the same result without an increase in complexity – the details of this are left to the reader as an exercise.

3.2 Highest Probability Parse

In RNA secondary structure prediction, we are not as interested in whether a sequence can fold into a structure as in *what structure* it is most likely to fold into. As discussed previously, the probabilities in the grammar make some structures more likely (more consistent with the grammar) than other structures. The CYK algorithm presented above does not take these probabilities into account – it only gives a binary true/false answer. A natural question is: what is the probability of the highest probability parse of a string under the grammar? In biological terms, this is the probability of the best structure the grammar can produce for that sequence (if the grammar is semantically unambiguous).

The basic CYK algorithm can be easily adapted to reveal this for a stochastic context-free grammar by simply considering the `P[i,j,N]` array to be an array of numbers, replacing `and` operations with products of probabilities, and making decisions on the basis of whether the potential new value is higher than the previous one. The modified algorithm for a stochastic context-free grammar in Chomsky normal form is:

```
1  initialize all P[i,j,N] to 0

2  for i = 1 to n do:
3    for each production N -> c_i
4      set P[i,i,N] = P(N->c_i)

5  for each j = 1 to n do:
6    for each i = 1 to j-1 do:
7      for each k = i to j-1 do:
8        for each production N -> AB do:
9          if P[i,k,A]*P[k+1,j,B]*P(N->AB)
              > P[i,j,N] then
10            set P[i,j,N] =
                 P[i,k,A]*P[k+1,j,B]*P(N->AB)

11 P[1,n,S] is the probability of the best parse
```

For grammars not in Chomsky normal form (such as the KH grammar) the algorithm can be adapted similarly to the examples in Subheading 3.1.

This algorithm still does not return that final structure. However, it is possible to further modify it to provide the actual parse tree of the highest probability parse. One way of doing this is storing which choice was made in an array of backpointers, and backtracking through that array after the value of `P[1,n,S]` is known.

Example 8: Suppose that the following choice is made in the CYK algorithm:

```
9          if P[i,k,A]*P[k+1,j,B]*P(N->AB)
              > P[i,j,N] then
10            set P[i,j,N] =
                 P[i,k,A]*P[k+1,j,B]*P(N->AB)
```

In this case, one could set the cell `T[i,j,N]` in the backpointer array to point to the cells `T[i,k,A]` and `T[k+1,j,B]` indicating the values that were needed to produce the final value in `T[i,j,N]`.

After the highest probability for the full parse, `P[1,n,S]`, is known, one can start from `T[1,n,S]` and trace back through `T`, reading off the

parse tree branch-by-branch, until the full parse tree is produced. This parse tree is the most probable way of parsing the string $c_1 \dots c_n$ under the grammar.[2]

The unmodified (boolean) CYK algorithm is rarely used in RNA secondary structure prediction, for reasons outlined above. The modified CYK algorithm (complete with backtracking), however, is very common and is typically referred to as "the CYK algorithm" without further elaboration.

3.3 The Inside–Outside Algorithm

The CYK algorithm can be used to determine the best parse of the sequence of using the grammar. However, one might also be interested in the *total* probability that the grammar can generate the string $c_1 \dots c_n$ at all. In a similar vein as before, we can talk about the total probability of generating any substring $c_i \dots c_j$, as well as the probability of generating everything outside of $c_i \dots c_j$ in the string $c_1 \dots c_n$.

Let us therefore define inside variables e, which represent the sum of the probabilities of all the possible ways of generating $c_i \dots c_j$ from nonterminal N:

$$e(i,j,N) = P(N \rightarrow c_i \dots c_j) \tag{2}$$

Similarly, we can define outside variables f, which represent the sum of the probabilities of all the possible ways of generating everything outside of $c_i \dots c_j$, given that N generates $c_i \dots c_j$:

$$f(i,j,N) = P(S \rightarrow c_1 \dots c_{i-1} N c_{j+1} \dots c_n) \tag{3}$$

Before we delve into the algorithmic details of how to calculate them, it is worth spending a moment reflecting on the power of the inside and outside variables. They can, in fact, be used to calculate many useful probabilities under the model of the grammar – some examples of this are shown below.

Example 9: In the KH grammar, there are two rules that produce basepairs: $L \rightarrow dFd$, and $F \rightarrow dFd$. The total probability that either one of these rules is used to parse $c_i \dots c_j$ (in the middle of the full string, $c_1 \dots c_n$) is actually the probability that positions i and j form a basepair under the SCFG model:

$$P_d(i,j) = \frac{f(i,j,L)e(i+1,j-1,F)P(L \rightarrow c_i F c_j)}{P(S \rightarrow c_1 \dots c_n)} + \frac{f(i,j,F)e(i+1,j-1,F)P(F \rightarrow c_i F c_j)}{P(S \rightarrow c_1 \dots c_n)} \tag{4}$$

Example 10: In a more general case, the probability of $c_i \dots c_j$ being derived from N is given by the product $e(i,j,N)f(i,j,N)$.

[2] In practice, we need not even store the array of backpointers at all. Starting from $P[1,n,S]$, one can simply do the calculations "backwards" and determine which choice must have been made in each step.

The probability that $c_i \dots c_j$ is derived from an initial application of the particular rule $N \to AB$ is

$$P(N \to AB \to c_i \dots c_j) = \frac{f(i,j,N) \sum_{k=1}^{j-1} P(N \to AB) e(i,k,A) e(k+1,j,B)}{P(S \to c_1 \dots c_n)} \tag{5}$$

As we shall see later, the quantities in this example are used for "training" the grammar (setting its probabilities).

Inside Algorithm

The inside algorithm is identical to the CYK algorithm for finding the highest probability parse, except that all maxima are replaced with sums. Consequently, its time complexity is also $O(n^3|P|)$ and the space complexity is $O(n^2|V|)$.

Outside Algorithm

The outside algorithm can be thought of as employing "reverse logic" compared to the CYK/inside algorithms. For every rule in the grammar $\alpha \to \beta$, in the CYK/inside algorithms we examine the right-hand side matching the nonterminal we want the value of the CYK/inside variable for. In the outside algorithm, we examine the left-hand side instead. Therefore we also start from the longest subsequences ($f(1,n,S) = 1$) and progress to shorter ones.

For a grammar in Chomsky normal forms, the outside algorithm is:

```
1  initialize f[1,n,S] to 1
   and all f[1,n,N] to 0

2  for each i = 1 to n do:
3    for each j = n to i do:
4      for each k = j to n do:
5        for each production N -> AB do:
6              increment f[i,j,A] with
               f[i,k,N]*e[j+1,k,B]*P(N->AB)
7      for each k = 1 to i do:
8        for each production N -> BA do:
9              increment f[i,j,A] with
               f[k,j,N]*e[k,i-1,B]*P(N->BA)
```

The outside algorithm is illustrated in Fig. 3.

In this algorithm, lines 4–6 represent the situation where we embed $c_i \dots c_j$ in a longer subsequence: $c_i \dots c_j c_{j+1} \dots c_k$ (expressed from N), which we then split into two parts: $c_i \dots c_j$ (expressed from A) and $c_{j+1} \dots c_k$ (expressed from B). $f(i,j,A)$ requires that $c_i \dots c_j$ is *excluded* from the complete sequence – so a contribution to $f(i,j,A)$ from this particular splitting involves a combination of the probability of excluding all of $c_i \dots c_k$ ($f(i,k,N)$) but at the same time also expressing the extra needed part $c_{j+1} \dots c_k$ ($e(j+1,k,B)$).

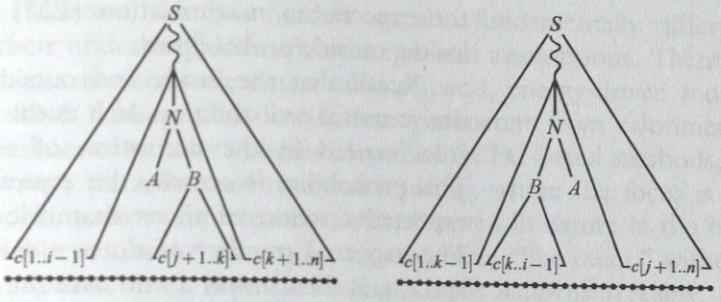

Fig. 3 An illustration of the two parts of the outside algorithm: the algorithm progressively fills out the subsequence not yet derived

Similarly, in lines 7–9 we consider the reverse situation, where $c_i \ldots c_j$ is embedded in $c_k \ldots c_{i-1} c_i \ldots c_j$, and $B \rightarrow c_k \ldots c_{i-1}$ and $A \rightarrow c_i \ldots c_j$. Again, for $f(i,j,A)$ we add the contribution from excluding the longer string $c_k \ldots c_j$ ($f(k,j,N)$) combined with including $c_k \ldots c_{i-1}$ ($e(k,i-1,B)$).

As is the case with the CYK/inside algorithms, the outside algorithm can also be adapted to grammars in non-Chomsky normal forms. However, the order in which operations are carried out requires attention, just as we have seen in the case of the CYK algorithm.

The computational complexity of the outside algorithm can be shown to be the same as the complexity CYK/inside algorithm.

3.3.3 Training a Grammar

So far we have assumed that the probabilities in the grammar are given – i.e., that our model is fully parametrized. But at some point the parameters need to be set so that the grammar best models what it is meant to model. In RNA secondary structure prediction, grammars are usually designed to model structures that are found in nature. Setting the probabilities of the grammar to the optimal values is termed "training" it.

If one has a dataset where the correct parses of some strings are known, the SCFG is trained by simply setting probability of each rule on the basis of the number times it was used in the derivations. In RNA secondary structure prediction, one will typically first obtain the parse trees from an input dataset of structures[3]: this can be done by using the CYK algorithm to parse all the input data (this will reveal the parse tree for every input structure).

If one has a dataset where the parses are not known (for example, if no RNA structures are known at all, or if one assumes that the structures we are trying to predict aren't like the ones already known), it is still possible to train the grammar so it produces the highest probability predictions. This is done through

[3]They can also be sequence-structure pairs, depending on the model.

the expectation maximization (EM) algorithm, which uses the inside–outside probabilities.

Recall that the inside and outside variables can be used to derive useful probabilities, such as the probability that a particular rule is used in the derivation of a subsequence (*see* Eq. 5). This probability is actually the *expectation* of that rule (i.e., the expected fraction of times that rule is used to parse $c_i \ldots c_j$). The expected number of times a rule is used in the parsing of $c_1 \ldots c_n$ is then simply a sum over the expectations for all possible subsequences. The probabilities in the SCFG can then be set from these expectations.

The training process in the EM algorithm is done in an iterative fashion. The details of this are beyond the scope of this chapter, but in short, the steps are as follows:

1. The probabilities are initialized to some (possibly random) values.
2. The expectation step: compute the expectation of each rule using the inside–outside algorithm with the current probabilities.
3. The maximization step: update the probabilities to the expectations determined in step 2.
4. Repeat 2–3 until the probabilities converge (or until some arbitrary cutoff value).

In this way, the grammar learns what probabilities will best describe the input data.

4 Discussion

4.1 SCFGs vs. Thermodynamic Models

The thermodynamic prediction of RNA secondary structure is discussed in Chapters 3 and 4 of this book. While it might seem that the two approaches have little in common, in some ways they are more alike than different. The KH grammar presented in this chapter can actually be thought of as a very simple thermodynamic model in itself: it gives scores between 0 and 1 to every basepair (distinguishing stacking basepairs from opening ones), single-stranded nucleotide or loop. Conversely, Rivas and Eddy have shown that Zuker's thermodynamic model can be converted to an SCFG by obtaining the probabilities of productions from the appropriate thermodynamic constants [4]. Both SCFG-based and thermodynamic models are formulated in terms of the optimization of an objective function: thermodynamic methods minimize free energy, SCFG methods maximize probability. In both cases, the optimization is expressed through recursion relations and implemented through dynamic programming algorithms, with the same $O(n^3)$ computational complexity.

However, the two models are also fundamentally different in their underlying scientific concepts and assumptions. Thermodynamic methods are based on a physical, energy-driven model for RNA folding, and obtain their parameters from calorimetric experiments on short oligonucleotides. SCFG-based methods, in contrast, are a form of machine learning, where the focus is on modelling the complete structures observed in nature in the best possible way, and then producing structures "like them," without assumptions about how the folding actually happens in reality.

SCFG-based methods are also inherently probabilistic. The advantage of this is that the theory of probability and statistics is very well developed in pure mathematics, and all that "machinery" can be applied to SCFGs in a rigorous way. SCFG models can also be extended with other models within the same probabilistic framework, in order to improve the quality of predictions, as we shall see in Subheading 4.2.

On the other hand, thermodynamic methods have the advantage that they work with information that is much easier to relate to. The prediction of the free energy contribution of a particular basepair in an RNA structure, for example, is a clear statement about the physical world that (at least in theory) can be tested experimentally. The probability of the same basepair under an abstract SCFG model is much harder to interpret.

It is also interesting to observe that an SCFG will always produce geometrically distributed loop lengths; this is not true for thermodynamic models.

4.2 Pfold: Extending the SCFG Model with Phylogenies

If we were to fold an arbitrary RNA molecule with the basic KH grammar presented in this chapter, we would quickly find that the predictions are highly inaccurate – significantly worse than thermodynamic predictions of the same sequences. The reason for this is that the KH SCFG, by itself, is a much less sophisticated model for RNA folding than thermodynamic models with hundreds of experimentally measured parameters, including a large number of special cases, such as extremely stable tetraloops. However, the KH grammar was not really meant to be used on its own, but in combination with covariance information derived from an alignment of the sequence with homologous sequences.

One way to think about this is that the SCFG proposes a "first guess" probability distribution over secondary structures – in Bayesian terms, this is the *prior probability distribution* over the structures. If additional information is known (such as mutual information between columns of an alignment, derived using an evolutionary model), the prior distribution of secondary structures can be adjusted ("evaluated") using conditional probabilities, so the final, *posterior probability distribution* assigns a probability to each secondary structure on the basis of how well it matches both the SCFG and the phylogenetic model. This has been done in pfold

by slight modifications to the inside–outside algorithm: in effect, the KH grammar coupled to the evolutionary model is an SCFG that has the columns of the alignment as its terminals. Grammars like this are known as phylo-grammars, and pfold became the first program to use one. The accuracy obtained by pfold in the prediction of the consensus structure of RNA structural alignments is significantly higher than that of purely thermodynamic programs for individual structure predictions of the same sequences. The caveat is that obtaining a good structural alignment is itself an unsolved problem (*see* Chapter 17): the alignment depends on the structure, but the structure is what we are trying to find using the alignment.

Some programs have been written to align and fold RNA sequences at the same time (*see* Subheading 6.3). However, the accurate prediction of RNA secondary structure remains an open problem. In practice, researchers interested in identifying RNA secondary structures bioinformatically will often iterate between different methods and manually adjust the results of each program on the basis of their background knowledge of the particular sequence.

4.3 Problems and Solutions

The main limitation of the SCFG algorithms discussed above is their computational complexity (this complexity, however, is the same as for algorithms based on thermodynamic models). This was prohibitive in the early days of SCFG-based methods; with the advance of more powerful computer hardware, recent efforts have been made to parallelize the algorithms [5], and even building specialized hardware [6].

Another well-known problem that occurs with the use of SCFGs is numerical underflow. The reason for this is the multiplication of a large number of probabilities (numbers between 0 and 1) with each other. Computer architectures can only represent a finite number of digits, resulting in the rounding of numbers with large negative exponents to zero. A common way of solving this problem is to use log-probabilities instead: products become sums, and the addition operation is typically implemented with the use of a lookup table. Another approach has been to use an extended exponent datatype to represent numbers that otherwise would give underflow [5].

Lastly, SCFG-based RNA secondary structure prediction, just like thermodynamic methods, suffers from the issue of optimizing the objective function over a very complex RNA secondary structure space. Structures with very similar probabilities are not necessarily similar, and very similar structures can have very different probabilities. Therefore the definition of the "best" structure being the one with the highest probability (computed by the CYK algorithm) has been called into question. In pfold, the solution of choice has been to optimize the expectation of predicted positions

instead of simply finding the highest probability structure. This optimization is done through an additional specially designed recursion.[4]

5 Recommended Reading

For more details on the formal theory of grammars, we recommend classic computer science textbooks, such as Languages and Machines by Thomas Subkamp [8]. For more on grammars in the context of RNA secondary structure and general sequence analysis, Biological Sequence Analysis by Durbin et al. [9] offers an excellent introduction.

6 Notes

Since their first application in RNA secondary structure modelling, SCFGs have proved to be extremely versatile and have appeared in a large number of popular computational tools. In this section, we provide an overview of the most well-known programs that use SCFGs in RNA secondary structure prediction and modelling.

6.1 Comparative Secondary Structure Prediction

6.1.1 Pfold/PPfold

Pfold [10, 11] uses the KH grammar described in this chapter and couples it to a phylogenetic model. The original pfold webserver is located at http://www.daimi.au.dk/~compbio/pfold/. Pfold takes an RNA alignment as input and returns a consensus secondary structure of the alignment. Each sequence of the alignment is annotated with a "stem-extended" secondary structure. Other outputs of the program include a dotplot over the likelihood of all possible basepairs, the phylogenetic tree calculated for the sequences on the basis of the evolutionary model, and reliability scores for every prediction in the final structure.

Recently, a multithreaded version of pfold has been developed [5]: this program is called PPfold and can be downloaded from http://birc.au.dk/software/ppfold/. PPfold is capable of solving the structure of longer alignments and can be run on any operating system with Java support. Command-line options are available for the advanced user; otherwise, the program has a standalone version with minimal graphical user interface for the selection of the input alignment, and it can also be downloaded as a full-featured plugin to the CLC Workbenches.

[4] A similar approach has recently been applied to a thermodynamic model, see [7].

6.1.2 RNA-Decoder

RNA-Decoder [12], like pfold, not only predicts the secondary structure of alignments, but it also takes into account the known protein-coding context of RNAs. It employs an SCFG together with a set of phylogenetic models designed to describe the overlapping evolutionary constraints.

6.1.3 Xrate

Xrate [13] is an interpreter for phylo-grammars. Its capabilities include maximum likelihood phylogeny, ancestral sequence reconstruction, alignment annotation, and model estimation. It can be downloaded as part of the DART package at http://biowiki.org/DART. Detailed instructions for its installation and use are found on the same site.

6.2 Similarity Search and Gene Finding

6.2.1 Infernal

Infernal [14] is a program to search DNA sequence databases for RNA structure and sequence similarities. It is based on covariance models, which are a special case of SCFGs. Infernal is also discussed in detail in Chapter 9. The popular Rfam database [15, 16] is based on Infernal. The Rfam database can be accessed at http://rfam.sanger.ac.uk/.

6.2.2 tRNAscan-SE

TRNAnscan-SE [17] is one of the several programs developed for detecting noncoding RNA genes in genomes; it detects tRNAs at very high sensitivities using SCFGs. The tRNAscan-SE webserver can be accessed at http://selab.janelia.org/tRNAscan-SE/.

6.2.3 Evofold

Evfold [18] identifies functional RNA structures in multiple-sequence alignments. It is based on a phylo-SCFG and exploits the differences of the substitution process in stem-pairing and unpaired regions to make its predictions. Evofold can be accessed at http://users.soe.ucsc.edu/~jsp/EvoFold/.

6.3 Simultaneous Folding and Aligning of Homologous RNAs

6.3.1 Consan

Consan [19] develops the original Sankoff algorithm [20] for simultaneous folding and alignment; it uses pair stochastic context-free grammars as a unifying framework for scoring pairwise alignment and folding at the same time. There is also a constrained version of the algorithm, which assumes knowledge of a few confidently aligned positions (pins). The pins are selected based on the posterior probabilities of a probabilistic pairwise sequence alignment. The program can be downloaded from http://selab.janelia.org/software/consan/.

6.3.2 Contrafold

Contrafold [21] is based on conditional log-linear models; this class of probabilistic models generalize on SCFGs by using discriminative training. Discriminative models are trained by maximizing conditional likelihood rather than joint likelihood. The Contrafold webserver can be accessed at http://contra.stanford.edu/contrafold/server.html. The program can also be downloaded from the same website.

6.3.3 Stemloc

Stemloc [22, 23] generates pairwise RNA structural alignments based on pair stochastic context-free grammars. Stemloc can be accessed through the DART library at http://biowiki.org/DART. Detailed instructions for its use can be found on the same site.

Acknowledgments

ZS would like to thank Robert Giegerich and Paula Tataru for their comments on the manuscript, and Christine Heitsch and her group at Georgia Tech for useful discussions.

References

1. Chomsky N (1956) Three models for the description of language. IRE Trans Inf Theory 2(3):113–124
2. Younger DH (1967) Recognition and parsing of context-free languages in time n3. Inf Control 10(2):189–208
3. Baker JK (1979) Trainable grammars for speech recognition. Speech communication papers for the 97th meeting of the acoustical society of America, pp 547–550, Boston, MA, 1979
4. Rivas E, Eddy SR (2000) The language of RNA: A formal grammar that includes pseudoknots. Bioinformatics 16(4): 334–340
5. Sükösd Z, Knudsen B, Værum M, Kjems J, Andersen ES (2011) Mulithreaded comparative RNA secondary structure prediction using stochastic context-free grammars. BMC Bioinformatics 12:103
6. Xia F, Dou Y, Zhou D, Li X (2010) Fine-grained parallel RNA secondary structure prediction using SCFGs on FGA. Parallel Comput 36:516–530
7. Lu ZJ, Gloor JW, Mathews DH (2009) Improved RNA secondary structure prediction by maximizing expected pair accuracy. RNA 10:1805–1813
8. Sudkamp TA (2005) Languages and machines: An introduction to the theory of computer science, 3rd edn. Addison Wesley, Reading, MA
9. Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: Probalistic models of proteins and nucleic acids. Cambridge University Press, Cambridge
10. Knudsen B, Hein J (1999) RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics 15(6):446–454
11. Knudsen B, Hein J (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. Nucleic Acids Res. 31(13):3423–3428
12. Pedersen JS, MeyerI, Forsberg R, Simmonds P, Hein J (2004) A comparative method for finding and folding RNA secondary structures within protein-coding regions. Nucleic Acids Res. 32:4925–4936
13. Klosterman P, Uzilov A, Bendana Y, Bradley R, Chao S, Kosiol C, Goldman N, Holmes I (2006) Xrate: a fast prototyping, training and annotation tool for phylo-grammars. BMC Bioinformatics 7(1):428
14. Nawrocki EP, Kolbe DL, Eddy SR (2009) Infernal 1.0: Inference of RNA alignments. Bioinformatics 25:1335–1337
15. Griffiths-Jones S, Moxon S, Marshall M, Khanna A, Eddy SR, Bateman A (2005) Rfam: annotating non-coding RNAs in complete genomes. Nucleic Acids Res. 33: D121–D124
16. Gardner PP, Daub J, Tate JG, Nawrocki EP, Kolbe DL, Lindgreen S, Wilkinson AC, Finn RD, Griffiths-Jones S, Eddy SR, Bateman A (2009) Rfam: updates to the RNA families database. Nucleic Acids Res. 37: D136–D140
17. Lowe TM, Eddy SR (1997) tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. Nucleic Acids Res. 25:955–964
18. Pedersen JS, Bejerano G, Siepel A, Rosenbloom K, Lindblad-Toh K, Lander ES, Kent J, Miller W, Haussler D (2006) Identification and classification of conserved RNA secondary structures in the human genome. PLoS Comput Biol 2(4):e33
19. Dowell RD, Eddy SR (2006) Efficient pairwise RNA structure prediction and alignment using sequence alignment constraints. BMC Bioinformatics 7:400
20. Sankoff D (1985) Simultaneous solution of the RNA folding, alignment and protosequence problems. SIAM J Appl Math 45(5): 810–825

21. Do CB, Woods DA, Batzoglou S (2006) Contrafold: RNA secondary structure prediction without physics-based models. Bioinformatics 22(14):90–98
22. Bradley RK, Pachter L, Holmes I (2008) Specific alignment of structured RNA: Stochastic grammars and sequence annealing. Bioinformatics 24(23): 2677–2683
23. Holmes I (2005) Accelerated probabilistic inference of RNA structure evolution. BMC Bioinformatics 6:73