ALCOMFT-TR-01-145

Comparing Stochastic Models

Arun Jagota[1], Rune B. Lyngsø[1,⋆], and Christian N. S. Pedersen[2,⋆⋆]

[1] Baskin Center for Computer Science and Engineering, University of California, Santa Cruz, CA 95064, U.S.A. E-mail: {jagota,rlyngsoe}@cse.ucsc.edu
[2] BRICS[⋆⋆⋆], Department of Computer Science, University of Aarhus, Ny Munkegade, DK-8000 Århus C, Denmark. E-mail: cstorm@brics.dk.

Abstract Stochastic models are commonly used in bioinformatics, e.g. hidden Markov models for modeling sequence families or stochastic context-free grammars for modeling RNA secondary structure formation. Comparing data is a common task in bioinformatics, and it is thus natural to consider how to compare stochastic models. In this paper we present the first study of the problem of comparing a hidden Markov model and a stochastic context-free grammar. We describe how to compute their co-emission – or collision – probability, i.e. the probability that they independently generate the same sequence. We also consider the related problem of finding a run through a hidden Markov model and derivation in a grammar that generate the same sequence and have maximal joint probability by a generalization of the CYK algorithm for parsing a sequence by a stochastic context-free grammar. We illustrate the methods by an experiment on RNA secondary structures.

1 Introduction

The basic chain-like structure of the key biomolecules, DNA, RNA, and proteins, allows an abstract view of these as strings, or sequences, over finite alphabets, obviously of finite length. Furthermore, these sequences are not completely random, but exhibit various kinds of structures in different contexts. E.g. a family of homologous proteins is likely to have similar amino acid residues in "equivalent" positions; an RNA sequence will have pairs of complementary subsequences to form base pairing helices. Hence, it is natural to consider applying models from formal language theory to model different classes of biological sequences.

Though not completely random, biological sequences can still possess inherent stochastic traits, e.g. due to mutations in a family of homologous sequences or a lack of knowledge (and computing power) to correctly model all aspects of RNA secondary structure formation. Thus, it is often better to use stochastic

⋆ Supported by grants from Carlsbergfondet and the Program in Mathematics and Molecular Biology
⋆⋆ Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT)
⋆⋆⋆ Basic Research in Computer Science, www.brics.dk, funded by the Danish National Research Foundation

models giving a probability distribution over all sequences, where a high probability reflects a sequence likely to belong to the class of sequences being modeled, instead of formalisms only distinguishing sequences as either belonging to the class being modeled or not. The two most widely used grammatical models in bioinformatics are hidden Markov models [1,3,8,9,16] and stochastic context-free grammars [7,14,15], though other models have also been proposed [13,18]. These two types of stochastic models were originally developed as tools for speech recognition (see [2,12]). One can identify hidden Markov models as a stochastic version of regular languages and stochastic context-free grammars as a stochastic version of context-free languages (see [17] for an introduction to formal languages).

As stochastic models are commonly used to model families of biological sequences, and as a common task in bioinformatics is that of comparing data, it is natural to ask how to compare two stochastic models. In [10] we described how to compare two hidden Markov models, by computing the co-emission – or collision – probability of the probability distributions of the two models, i.e. the probability that the two models independently generate the same sequence. Having the co-emission probability for a pair of probability distributions as well as for each of the distributions with itself, it is easy to compute the L_2- and the Hellinger-distance between the two distributions. In this paper we study the problem of comparing a hidden Markov model and a stochastic context-free grammar. We develop recursions for the co-emission probability of the distributions of the model and the grammar, recursions that lead to a set of quadratic equations. Though quadratic equations are generally hard to solve, we show how to find an approximate solution by a simple iteration scheme. Furthermore, we show how to solve the equivalent maximization problem, the problem of finding a run through the hidden Markov model and derivation in the grammar that generate the same sequence and have maximal joint probability. This is in essence parsing the hidden Markov model by the grammar, so the algorithm can be viewed as a generalization of the CYK algorithm for parsing a sequence by a stochastic context-free grammar. Indeed, in most cases the complexity of our algorithm will be identical to the complexity of the CYK algorithm. Finally we discuss the undecidability of some natural extensions of our results.

The structure of this paper is as follows. In Sect. 2 and 3 we briefly introduce hidden Markov models and stochastic context-free grammars and the terminology we use. In Sect. 4 we consider the problem of computing the co-emission probability of the probability distributions of a hidden Markov model and a stochastic context-free grammar, and in Sect. 5 we develop the algorithm for parsing a hidden Markov model by a stochastic context-free grammar. In Sect. 6 we present an illustrational experiment and in Sect. 7 we discuss the problems occurring when trying to extend the methods presented in Sect. 4 and 5.

2 Hidden Markov Models

A hidden Markov model M is a generative model consisting of n states and m transitions between states, where each state is either silent or non-silent. The model generates a string s over a finite alphabet Σ with probability $P_M(s)$ such that $\sum_s P_M(s) = 1$, i.e. M describes a probability distribution over the set of finite strings over Σ. A hidden Markov model is a *left-right model* if the states can be numbered $1, 2, \ldots, n$ such that all transitions $i \to j$ from state i to state j satisfy $i \leq j$.

A run in M begins in a special *start-state* and continues from state to state according to the *state transition probabilities*, where $a^M_{q,q'}$ is the probability of a transition from state q to q', until a special *end-state* is reached. A state is either a *silent* or a *non-silent* state. Each time a non-silent state is entered, a symbol is emitted according to the *symbol emission probabilities*, where $e^M_{q,\sigma}$ is the probability of emitting symbol $\sigma \in \Sigma$ in state q. When entering a silent state, nothing is emitted. A run thus follows a Markovian path $\pi = (\pi_0, \pi_1, \ldots, \pi_k)$ of states and generates a string $s \in \Sigma^*$ which is the concatenation of the emitted symbols. The probability $P_M(s)$ of M generating s is the probability of following any path and generating s. The probability $P_M(\pi, s)$ of following path $\pi = (\pi_0, \pi_1, \ldots, \pi_k)$ and generating s depends on the subsequence $(\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_\ell})$ of non-silent states on the path π. If the length of s is different from the number of non-silent states along π, the probability $P_M(\pi, s)$ is zero. Otherwise, we get

$$P_M(\pi, s) = P_M(\pi) \cdot P_M(s \mid \pi) = \prod_{i=1}^{k} a^M_{\pi_{i-1},\pi_i} \cdot \prod_{j=1}^{\ell} e^M_{\pi_{i_j},s_j}\,, \quad P_M(s) = \sum_{\pi} P_M(\pi, s).$$

A *partial run* in M is a run that starts in a state p and ends in a state q, which are not necessarily the special start- and end-states. To ease the presentation of the methods in Sect. 4, we introduce the concept of a partial run from p to q *semi-including* q meaning that if q is a non-silent state then no symbol has yet been emitted from q. The probability of a partial run from p to q semi-including q and generating s is thus the probability of generating s on the path from p to the immediate predecessor of q and taking the transition to q. Given a string s and a model M, efficient algorithms for determining $P_M(s)$ and the run in M of maximal probability generating s are known, see e.g. [5,12].

3 Stochastic Context Free Grammars

A context-free grammar G describes a set of finite strings over a finite alphabet Σ, also called a language over Σ. It consists of a set V of non-terminals, a set $T = \Sigma \cup \{\epsilon\}$ of terminals, where ϵ is the empty string, and a set P of production rules $\alpha \to \beta$, where $\alpha \in V$ and $\beta \in (V \cup T)^+$. A production rule $\alpha \to \beta$ means that α can be rewritten to β by applying the rule. A string $s \in \Sigma^*$ can be *derived* from a non-terminal U, $U \stackrel{*}{\Rightarrow} s$, if U can by rewritten to s by a sequence of production rules; s is in the language described by G if it can be derived from a special start non-terminal S. A derivation D of s in G is a sequence of production rules which rewrites S to s. A derivation of s in G is also called a parsing of s in G.

A *stochastic* context-free grammar G is a context-free grammar in which all production rules $\alpha \to \beta$ are assigned probabilities $P_G(\alpha \to \beta)$ such that the sum of the probabilities of all possible production rules from any given non-terminal is one. The resulting stochastic grammar describes a probability distribution over the set of finite string over the finite alphabet Σ, where the probability $P_G(s)$ of deriving s in G (sometimes also referred to as the probability of G generating s) is the sum of the probabilities of all possible derivations of s from S. The probability of a derivation D of s in G, i.e. $P_G(D) = P_G(S \stackrel{D}{\Rightarrow} s)$, is the product of the probabilities of the production rules in the sequence D applied to derive s from S.

Any (stochastic) context-free grammar can be transformed to an equivalent (stochastic) context-free grammar in Chomsky normal form which describes the same language but where the production rules can be split into two sets; a set P_n of non-terminal production rules on the form $U \to XY$, and a set P_t of terminal production rules on the form $U \to \sigma$ and $U \to \epsilon$, where U, X and Y are non-terminals, σ is a symbol in Σ, and ϵ is the empty string. For a stochastic context-free grammar G in Chomsky normal form, we can compute $P_G(s)$ by the inside algorithm, and the most likely parse of s in G by the CYK algorithm, in time $O(|P| \cdot |s|^3)$ where $|P|$ is the number of production rules in G, i.e. $|P_n| + |P_t|$, and $|s|$ is the length of s, see e.g. [5].

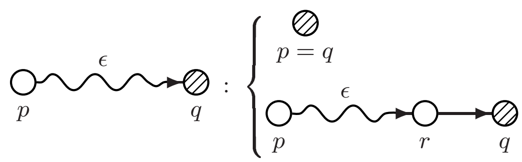
4 Comparing a SCFG and a HMM

In this section we will consider the problem of comparing a stochastic context-free grammar G (in Chomsky normal form) and a hidden Markov model M which generate strings over the same alphabet Σ. More precisely, we will consider the problem of computing the co-emission probability, $C(G, M)$, of the probability distributions of G and M over all strings, i.e. the quantity

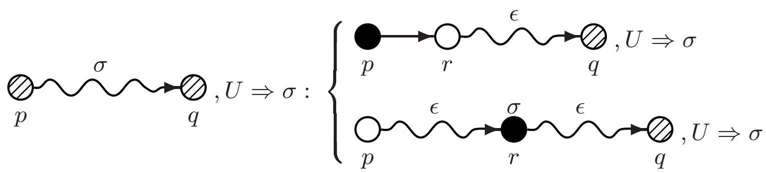
$$C(G, M) = \sum_{s \in \Sigma^*} P_G(s) \cdot P_M(s),$$

which is similar to the definition of the co-emission probability of two hidden Markov models in [10]. This quantity is also often referred to as the collision probability of the probability distributions of G and M, as it is the probability that strings picked at random according to the two probability distributions collide, i.e. are identical. We initially assume that M is an acyclic hidden Markov model, i.e. a left-right hidden Markov model where no states have self-loop transitions. By itself, this is not a very interesting class of hidden Markov models, e.g. a model of this type cannot generate strings of arbitrary length, but the ideas of our approach for computing the co-emission probability for this class of models are also applicable to left-right and general hidden Markov models.

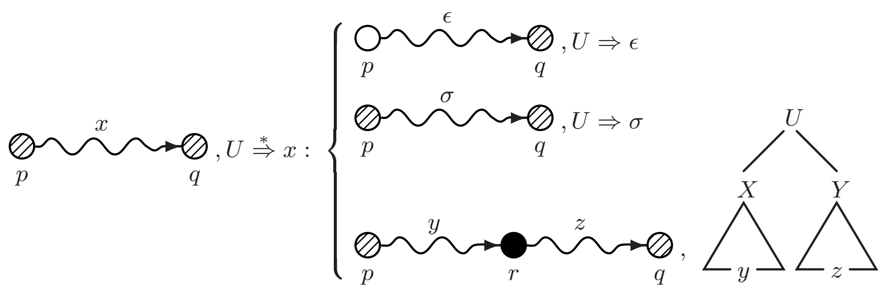
For acyclic hidden Markov models we can use an approach closely mimicking the inside algorithm for computing the probability that a stochastic context-free grammar generates a given string. In the inside algorithm, when computing the probability that a string s is derived in a stochastic context-free grammar,



(a) The A array holds the probabilities of getting from one state p to another state q in M without emitting any symbols.



(b) The B array holds the probabilities of getting from one state p to another state q in M while emitting only one symbol and at the same time generating the same symbol by a terminal production rule for the non-terminal U in G.



(c) The C array holds the probabilities of getting from one state p to another state q in M while emitting any string and at the same time generating the same string from a non-terminal U in G.

Figure 1. Illustration of the individual purposes and recursions of the three arrays used. Hollow circles denote silent states, solid circles denote non-silent states, and hatched circles denote states of any type. Squiggle arrows indicate partial runs of arbitrary length and straight arrows indicate single transitions between states.

track is being kept of the probability that a substring of s is derived from a non-terminal of the grammar. In our algorithm for computing the co-emission probability of G and M we keep track of the probability of deriving the same string from a non-terminal that is generated on a partial run from a state p to a state q semi-including q in M. In our dynamic programming based algorithm we maintain three arrays, A, B, and C, for the following purposes.

– $A(p,q)$ will be the sum of the probabilities of all partial runs from p to q semi-including q that does not emit any symbols, illustrated in Fig. 1(a). I.e. all states on the partial runs, except possibly for q, are silent states.
– $B(U,p,q)$ is the probability of independently deriving a single symbol string from the non-terminal U and generating the same single symbol string on a partial run from p to q semi-including q, illustrated in Fig. 1(b).
– $C(U,p,q)$ is the probability of independently deriving a string from the non-terminal U and generating the same string on a partial run from p to q semi-including q, illustrated in Fig. 1(c).

The purpose of the A and B arrays is to deal efficiently with partial runs consisting of silent states. As all symbols of a string are in a sense "non-silent", this is the main new problem encountered when modifying the inside algorithm to "parse" acyclic hidden Markov models.

The C array is similar to the array maintained in the inside algorithm. The array of the inside algorithm tells us the probability of deriving any substring of the string being parsed from any of the non-terminals of G. Similarly, the C array will tell us the probability of independently generating a sequence on a partial run between any pairs of states and at the same time deriving it from any of the non-terminals of G. It is evident that $C(G,M) = C(S, \mathit{start}, \mathit{end})$, where S is the start symbol of G and *start* and *end* are the start- and end-states of M, as the co-emission probability of G and M is the probability of deriving the same string from S that is generated on a (genuine) run from *start* to *end*. This is assuming that the end-state of M is silent, so that any partial run from *start* to *end* semi-including *end* in M is also a genuine run in M.

Having described the required arrays, we must specify recursions for computing them – and argue that these recursions split the computations into ever smaller parts, i.e. that the dependencies of the recursions are acyclic – to obtain a dynamic programming algorithm computing $C(G,M)$. The A array are the probabilities for getting from the state p to the state q in M along a path that only consists of silent states. In general such a path can be broken down into the last transition of the path and a preceding path only going through silent states. Hence, we obtain the following recursion where the first case takes care of the initialization.

$$A(p,q) = \begin{cases} 1 & \text{if } p = q \\ \displaystyle\sum_{p \leq r < q,\ r \text{ silent}} A(p,r) \cdot a^M_{r,q} & \text{otherwise} \end{cases} \tag{1}$$

The ordering of states referred to in the summation is any ordering consistent with the (partial) ordering of states by the acyclic transition structure. One immediately observes that each entry of the A array requires time at most time $\mathrm{O}(n)$ to compute. Thus the entire A array can be computed in time $\mathrm{O}(n^3)$. However, one can observe that we actually only need to sum over states r with a transition to q in the summation of (1). This observation reduce the time requirements to $\mathrm{O}(nm)$ as each transition is part of $\mathrm{O}(n)$ of the above recursions.

The B array holds the probabilities for getting from the state p to the state q in M by a partial run generating exactly one symbol, and at the same time generating the same symbol from U by a terminal production rule. In general we can partition the partial run into an initial path consisting only of silent states and the remaining partial run starting with the non-silent state emitting the symbol. Hence, we obtain the following recursion where "initialization" is handled by paying special attention to the special case where the initial path of silent states is empty.

$$\begin{aligned} B(U,p,q) = \sum_{(U\to\sigma)\in G} & P_G(U\to\sigma)\cdot \\ & \big(e^M_{p,\sigma}\cdot \sum_{p<r\leq q} a^M_{p,r}\cdot A(r,q) + \sum_{\substack{p<r<q,\\ r \text{ non-silent}}} A(p,r)\cdot B(U,r,q)\big) \end{aligned} \quad (2)$$

To find the time requirements for computing the B array one observes that each non-terminal of G occurs in $\mathrm{O}(n^2)$ entries. Hence, each terminal production rule of G is part of $\mathrm{O}(n^2)$ of the above recursions. Each of these recursions requires time $\mathrm{O}(n)$ to compute. Thus, computing the B array requires time $\mathrm{O}(|P_t|\cdot n^3)$.

The C array should hold the probabilities for getting from the state p to the state q in M by a partial run generating any string, and at the same time generating the same string from a non-terminal U in G. The purpose of the A and B arrays is to handle the special cases where at most a single symbol is emitted on a partial run from p to q. In these cases we do not really recurse on a non-terminal U from G but either ignore G completely or only consider terminal production rules. In the general case where a string with more than one symbol is generated, we need to apply a non-terminal production rule $U \to XY$ from G. Part of the string is then derived from X and part of the string is derived from Y. This leads to the following recursion.

$$\begin{aligned} C(U,p,q) = {} & P_G(U\to\epsilon)\cdot A(p,q) + B(U,p,q) + \\ & \sum_{(U\to XY)\in G} P_G(U\to XY)\cdot \sum_{\substack{p<r<q\\ r \text{ non-silent}}} C(X,p,r)\cdot C(Y,r,q) \end{aligned} \quad (3)$$

The reason for requiring r to be a non-silent state in the last sum is to ensure a unique decomposition of the partial runs from p to q. If we allowed r to be silent we would erroneously include some partial runs from p to q several times in the summation. As we did with the computation of the B array, we can observe that each non-terminal production rule is part of $\mathrm{O}(n^2)$ of the above recursions, and that each recursion requires time $\mathrm{O}(n)$ to compute. Hence, we obtain total time requirements of $\mathrm{O}(|P_n|\cdot n^3)$ for computing the C array. Adding the time requirements for computing the three arrays leads to overall time requirements of $\mathrm{O}(|P|\cdot n^3)$ for computing the co-emission probability of G and M. This is in correspondence with the time requirements of $\mathrm{O}(|P|\cdot |s|^3)$ for parsing a string s by the grammar G cf. Sect. 3.

As previously stated, in order for these recursions to be used for a dynamic programming algorithm, we need to argue that the recursions only refer to elements that in some sense are smaller. I.e. that no recursion for any of the entries of the three arrays depends cyclicly on itself. But this is an easy observation as all pairs of states indexing an array on the right-hand side of the recursions are closer in the ordering of states than the pair of states indexing the array on the left-hand side of the recursions.

The point of always recursing by smaller elements is exactly where we run into problems when trying to extend the above method to allow cycles in M. Even when only allowing the self-loops of left-right models this problem crops up. We can easily modify (1), (2), and (3) to remain valid as equations for the entries of the A, B and C arrays. All that is needed is to change some of the strict inequalities in the summation boundaries to include equality. More specifically, if we assume that only non-silent states can have self-loop transitions (self-loop transitions for silent states are rather meaningless and can easily be eliminated), we need to change (2) and (3) to

$$\begin{aligned} B(U,p,q) = \sum_{(U\to\sigma)\in G} & P_G(U\to\sigma)\cdot \\ & \Big(e^M_{p,\sigma}\cdot \sum_{p\le r\le q} a^M_{p,r}\cdot A(r,q) + \sum_{\substack{p<r\le q,\\ r \text{ non-silent}}} A(p,r)\cdot B(U,r,q)\Big)\,, \end{aligned} \tag{4}$$

$$\begin{aligned} C(U,p,q) = P_G(U\to\epsilon)\cdot A(p,q) + B(U,p,q) + \\ \sum_{(U\to XY)\in G} P_G(U\to XY)\cdot \sum_{p\le r\le q} C(X,p,r)\cdot C(Y,r,q). \end{aligned} \tag{5}$$

For the B array, (4) still refers only to smaller elements. Either the A array or an entry of the B array where the two states are closer in the ordering as r has to be strictly larger than p in the ordering in the last sum. But for the C array we might choose r equal to either p or q in the last sum. Hence, to compute $C(U,p,q)$ we need to know $C(X,p,q)$ and $C(Y,p,q)$ which might in turn depend on (or even be) $C(U,p,q)$.

But, as stated above, (5) still holds as equations for the entries of the C array. For each pair of states, p and q, in M we thus have a system of equations with one variable and one equation (and the restriction that all variables have to attain non-negative values) for each non-terminal of G. Assume that we solve these systems in an order where the distance between the pair of states in the ordering is increasing, i.e. we first consider systems with $p = q$, then systems with q being the successor to p, etc. Then most of the systems will be systems of linear equations. In the equation for a given entry $C(U,p,q)$ the only unknown quantities are the occurrences of $C(X,p,q)$ and $C(Y,p,q)$ corresponding to production rules $U \to XY$ of G. These occurrences have coefficients $P_G(U \to XY)\cdot C(Y,q,q)$ and $P_G(U \to XY)\cdot C(X,p,p)$, respectively, coefficients that have known values if $p < q$. But if $p = q$ the last sum of (5) will lead to a number of terms of the form $C(X,p,p) \cdot C(Y,p,p)$. I.e. the system of equations is quadratic. Hence, for each state with a self-loop transition we need to solve a system of quadratic equations with one variable and one equation for each non-terminal of G. General systems of quadratic equations are hard to solve, see e.g. [6], but the construction proving this requires equations with all terms having coefficients with the same sign. One can immediately observe that in a system of equations based on (5) the left-hand side terms will have coefficients that have opposite sign of the right-hand side terms. Hence, the hardness proof does not relate to the system of quadratic equations we obtain from (5). We have not been able to find any literature on algorithms solving systems of the type derivable from (5). But as all the dependencies are positive we can approximate a solution simply by initializing all entries to the terms depending only on the A and B arrays and then iteratively update each entry in turn. This process will converge to the true solution. We conjecture that this convergence will be very rapid for all realistic grammars and models.

For general hidden Markov models we do not have an ordering of the states of M. Hence, to modify (1), (2), and (3) to hold for general hidden Markov models, the ordering constraints on states should be removed from all the summations. When we do no longer have any ordering of the states, all entries might depend on each other. Thus, we cannot separate the systems of equations for the entries of the C array into independent blocks based on the pair of states indexing the entry of the array. This means that we obtain just one aggregate system of quadratic equations with $|V| \cdot n^2$ variables and equations for the entries of the C array, where $|V|$ is the number of non-terminals in G and n is the number of states in M. However, for the entries of the A and B arrays we still only get systems of linear equations. Actually, the B array can even be computed by simple dynamic programming.

5 Parsing an HMM by a SCFG

Given a stochastic context-free grammar G modeling RNA secondary structures and a hidden Markov model M representing a family of RNA sequences, we can use the method of the previous section to determine the likelihood that the family has a common secondary structure. This being established, one might want to find the most likely structure. In other words, an equivalent to the CYK algorithm for finding the most likely parse of a string, but finding the most likely "parse" of a hidden Markov model, is desirable. Hence, we want to compute

$$\max\{P_G(D) \cdot P_M(r) \mid D \text{ a derivation in } G,\, r \text{ a run in } M, \text{ and } S \overset{D}{\Rightarrow} s_r\}, \quad (6)$$

where s_r is the string generated by the run r, preferably in a way that allows us to easily extract a derivation D and a run r witnessing the maximum.

The basic principles we will use to compute the value of (6) are similar to those used for computing the co-emission probability in the previous section. Thus, the general technique is to find the probabilities of optimal combinations of a derivation from a non-terminal U in G and a partial run between a pair of states p and q in M yielding the same string. Furthermore, this optimal combination is split into a pair of optimal combinations of a derivation from a non-terminal and a partial run between states as illustrated in Fig. 1(c). However, computing the maxima is easier than computing the sums. The main reason for this is that we do not have to consider combinations of derivations and partial runs of arbitrary lengths. The probability of an optimal combination for a particular choice of a non-terminal and pair of states cannot depend cyclicly on itself – the product of two or more probabilities can never be larger than any of these probabilities. A similar principle is used in algorithms for finding shortest paths in graphs with only positive edge weights, cf. [4, chapters 25–26]. Not surprisingly, the methods described in this section will bear strong resemblances to such algorithms.

The description of our method will employ three arrays $A_{\max}$, $B_{\max}$, and $C_{\max}$, similar to the A, B, and C arrays used in previous section. The $A_{\max}(p,q)$ entries hold the maximum probability of any partial run from state p to state q semi-including q not emitting any symbols. But this is just the path of maximum weight in the graph defined by the transitions not leaving a non-silent state in M. We can thus compute the $A_{\max}$ array by standard all-pairs shortest paths graph algorithms in time $\mathrm{O}(n^2 \log n + nm)$, cf. [4, p. 550].

The $B_{\max}(U,p,q)$ entries hold the maximum probability of a combination of a derivation consisting of only a terminal production rule $U \to \sigma$ and a partial run from p to q generating a string consisting only of the symbol σ. This imposes a restriction on the path from p to q preventing the use of standard graph algorithms. Still, we only need to combine transitions from non-silent states with preceding and succeeding partial runs not emitting any symbols, i.e. with entries of $A_{\max}$. However, if we compute the $B_{\max}$ entries directly from the equation

$$B_{\max}(U,p,q) = \max_{\substack{(U\to\sigma)\in G \\ p\to q\in M}} \left\{ P_G(U \to \sigma) \cdot e^M_{r,\sigma} \cdot a^M_{r,s} \cdot A_{\max}(p,r) \cdot A_{\max}(s,q) \right\} \quad (7)$$

we use time $\mathrm{O}(|P_t| \cdot m \cdot n^2)$. This could possibly be the dominating term in the overall time requirements for computing the value of (6), as we will see later.

Hence, we will specify a more efficient way to compute the $B_{\max}(U,p,q)$ entries. If p is a non-silent state the optimal choice of a preceding partial run not emitting any symbols must be the empty run. Thus

$$B_{\max}(U,p,q) = \max_{\substack{(U\to\sigma)\in G \\ p\to r\in M}} \left\{ P_G(U \to \sigma) \cdot e^M_{p,\sigma} \cdot a^M_{p,r} \cdot A_{\max}(r,q) \right\} \quad (8)$$

for all entries of $B_{\max}$ with p non-silent. Having computed these entries, we can now proceed to compute the entries for p silent by

$$B_{\max}(U,p,q) = \max_{\substack{(U\to\sigma)\in G \\ r\in M,\ r \text{ non-silent}}} \left\{ A_{\max}(p,r) \cdot B_{\max}(U,r,q) \right\}. \quad (9)$$

Computing the $B_{\max}$ array this way reduces the time requirements to $\mathrm{O}(|P_t| \cdot n^3)$.

We are now ready to specify how to determine the value of (6), i.e. how to compute the entries of the $C_{\max}$ array. An entry $C_{\max}(U,p,q)$ holds the maximum probability of a combination of a derivation from U in G and a partial run from p to q in M yielding the same string. The following equation for computing an entry of $C_{\max}$ closely follows Fig. 1(c).

$$C_{\max}(U,p,q) = \max \begin{cases} P_G(U \to \epsilon) \cdot A_{\max}(p,q) \\ B_{\max}(U,p,q) \\ \max\big\{ P_G(U \to XY) \cdot C_{\max}(X,p,r) \cdot C_{\max}(Y,r,q) \\ \qquad\qquad\qquad\qquad\qquad \mid (U \to XY) \in G, r \in M \big\} \end{cases} \tag{10}$$

The only exception is that there is no harm in considering the same combination of a derivation and a partial run several times when working with maxima instead of sums. Hence, there is no restriction on the type of the state (denoted by r) that we maximize over in the general case. In an actual implementation one might want to retain the type restriction to speed up the program, though.

Again, this is slightly beyond a shortest path problem. However, (10) gives us a number of inequalities for each of the entries of $C_{\max}$, similar to [4, Lemma 25.3]; $C_{\max}(U,p,q)$ must be at least as large as any of the terms on the right-hand side of (10). Thus, we can use a technique very similar to the relaxation technique discussed in [4, pp. 520–521]; if at any time $C_{\max}(U,p,q) < P_G(U \to XY) \cdot C_{\max}(X,p,r) \cdot C_{\max}(Y,r,q)$ for any $(U \to XY) \in G$ and $r \in M$ we can increase $C_{\max}(U,p,q)$ to this value. This means that we could start by initializing each $C_{\max}(U,p,q)$ to $\max\{P_G(U \to \epsilon) \cdot A_{\max}(p,q), B_{\max}(U,p,q)\}$ and then keep updating $C_{\max}$ by iterating over all possible relaxations until no more changes occur. This process will eventually terminate as no entry can depend cyclicly on itself, as discussed above.

The worst-case time requirements of this scheme are quite excessive, though. Thus, the question of how to order the relaxations so as to compute $C_{\max}$ most efficiently still remains. We propose an approach very similar to Dijkstra's algorithm for computing single-source shortest paths in a graph, cf. [4, Sect. 25.2]. Assume that $\mathcal{S}$ is the set of entries for which we have already determined the correct value, and that we have performed all relaxations combining the correct values of these entries (and initialized all entries using the $A_{\max}$ and $B_{\max}$ arrays as mentioned in the preceding paragraph). Let $C_{\max}(U,p,q)$ be an entry of maximum value not in $\mathcal{S}$. We claim that the current value of $C_{\max}(U,p,q)$ must be correct, i.e. that it cannot be increased. In fact, no entry not in $\mathcal{S}$ can have a correct value larger than $C_{\max}(U,p,q)$. The reason for this is that any relaxation not combining two entries both in $\mathcal{S}$ – which we assumed have already been performed – will involve an entry not in $\mathcal{S}$, and thus with current value at most $C_{\max}(U,p,q)$. As the other entry used in the relaxation can have a value of at most 1, no future relaxations can lead to values above $C_{\max}(U,p,q)$. Hence, we can insert $C_{\max}(U,p,q)$ in $\mathcal{S}$ and perform all relaxations combining $C_{\max}(U,p,q)$ with an entry from $\mathcal{S}$. This idea is formalized in algorithm 1.

So what are the time requirements of algorithm 1? To some extent this of course depends on the choice of data structures implementing the priority queue

Algorithm 1 The algorithm for computing an optimal parse of a hidden Markov model M by a stochastic context-free grammar G.

```
/* Initialization */
PQ = ∅, S = ∅
for all U ∈ G, p, q ∈ M do
  PQ.insert((U, p, q); min{P_G(U → ε) · A_max(p, q), B_max(U, p, q)})
/* Main loop where one entry is fixed at a time */
while PQ is not empty do
  /* Fix the entry with highest probability not yet fixed */
  (U, p, q); x = PQ.deletemax
  S.insert((U, p, q); x)
  /* Combine this entry with all feasible, fixed entries */
  for all X, Y ∈ G with (X → UY) ∈ G and all r ∈ M with (Y, q, r); y ∈ S do
    PQ.increasekey((X, p, r); x · y)
  for all X, Y ∈ G with (X → YU) ∈ G and all r ∈ M with (Y, r, p); y ∈ S do
    PQ.increasekey((X, r, q); y · x)
```

PQ and the set $\mathcal{S}$, but a key observation is that each possible relaxation, i.e. combination of two particular entries, is only performed once, namely when the entry with the smaller value is inserted in $\mathcal{S}$. Hence, algorithm 1 performs $\mathrm{O}(|P_n| \cdot n^3)$ relaxations. One can observe that for each relaxation we need to perform the operation increasekey on PQ, while all other operations on PQ are performed at most $\mathrm{O}(|V| \cdot n^2)$ times. Thus, increasekey is the most critical operation, why we will assume that PQ is implemented by a Fibonacci heap. This limits the time requirements for all operations on PQ to $\mathrm{O}(|P_n| \cdot n^3 + |V| \cdot n^2 \cdot \log(|V| \cdot n))$.

For the set $\mathcal{S}$ we need to be able to insert an element efficiently, and to efficiently iterate through all elements with a particular non-terminal and state. But having already set aside time $\Omega\left(|P|_n \cdot n^3\right)$ for the priority queue operations, the operations on $\mathcal{S}$ do not need to be that efficient. As it turns out, it is actually sufficient to maintain $\mathcal{S}$ simply as a three-dimensional boolean array indexed by (U, p, q). This makes insertion a constant time operation. However, it does not allow for an efficient way to iterate over all elements in $\mathcal{S}$ with a particular non-terminal and state, short of iterating over all elements with that non-terminal and state and test the membership of each individual element. However, this turns out to be sufficiently efficient. In some situations, especially in the beginning when there are only a few elements in $\mathcal{S}$, we might test the membership of numerous elements not in $\mathcal{S}$. But each membership test can be associated with a relaxation involving a particular pair of elements, namely the relaxation that is performed if the test succeeds. Furthermore, for each relaxation we will only test membership twice, once for each element that the relaxation combines. Hence, the total time we spend iterating over elements in $\mathcal{S}$ is $\mathrm{O}(|P_n| \cdot n^3)$. Thus, the time requirements for algorithm 1 is $\mathrm{O}(|P_n| \cdot n^3 + |V| \cdot n^2 \cdot \log(|V| \cdot n))$. Combined with the time complexity of computing the $A_{\max}$ and $B_{\max}$ arrays, this leads to an overall time complexity of $\mathrm{O}(|P| \cdot n^3 + |V| \cdot n^2 \cdot \log(|V| \cdot n))$ for determining the optimal parse of a general hidden Markov model by a stochastic context-free grammar (having computed $A_{\max}$, $B_{\max}$, and $C_{\max}$ it is easy to find the optimal parse by standard backtracking techniques). This should be compared with the time requirements of $\mathrm{O}(|P| \cdot |s|^3)$ for finding the optimal parse of a string s by the CYK algorithm.

In the above description we did not use any assumptions about the structure of the hidden Markov model M. A natural question to ask is thus how much can be gained with respect to time requirements, by restricting our attention to left-right models. That it is not a lot should not be surprising, considering that we are already close to the complexity of the CYK algorithm. However, we can observe that $C_{\max}(U, p, q)$ can only depend on entries $C_{\max}(U', p', q')$ with $p \leq p' \leq q' \leq q$ (where the ordering is with respect to the implicit partial ordering of states in a left-right model). Thus, we can separate (10) into $\mathrm{O}(n^2)$ systems, one for each choice of p and q, that can be solved one at a time in a predefined order. Hence, the priority queue only needs to hold at most $|V|$ elements at any time, reducing the time complexity for finding the optimal parse to $\mathrm{O}(|P| \cdot n^3 + |V| \cdot n^2 \cdot \log |V|)$. More importantly, though, as we only need a priority queue with at most $|V|$ elements and $|V|$ will usually be very small, it might be feasible to replace the Fibonacci heap implementation with implementations that have worse asymptotic complexities but smaller overhead. Thus, if we just implement PQ with an array, scanning the entire array each time a deletemax operation is performed, the complexity of parsing a left-right model only increases to $\mathrm{O}(|P| \cdot n^3 + |V|^2 \cdot n^2)$ with the involved constants being very small.

6 Results

Algorithm 1 has been implemented as the program CStoRM (Comparison of Stochastic (Random) Models) and we are currently working on adding computation of the co-emission probability to the implementation. The implementation is available at `http://www.cse.ucsc.edu/~rlyngsoe/cstorm.tar.gz`. As an illustrating experiment we have used the program to parse the `trna2` profile hidden Markov model that is part of the test suite for the SAM software distribution available at `http://www.cse.ucsc.edu/research/compbio/sam.html`. This model is built from the alignment of *seq1* and *seq2* shown in Fig. 2, with the symbol emission probabilities in each position being a little less than 0.1 for symbols not present in that position in the alignment and the rest of the probability distributed evenly among the symbols present in that position in the alignment. Each match-state has a probability of at least 0.85 – and for positions without gaps in the alignment more than 0.97 – for choosing the transition to the next match-state. The grammar used is the stochastic context-free grammar for general RNA secondary structure presented in [7]. This grammar was also used for predicting the secondary structure of *seq1* and *seq2* shown in Fig. 2.

As is evident from Fig. 2, the maximal parse does a poor job at finding the common structural elements of the two sequences. This might in part be explained by the fact that only about half the base pairs of the structures of each of the sequences is shared with the structure of the other sequence. But

```
seq1 (((((((..((((...  .. ..))))(((((...)))))).......(((((.......))))))))))   )).
     GGGGAUGUAGCUCAGU--GG-UAGAGCGCAUGCUUCGCAUGUAUGAGGCCCCGGGUUCGAUCCCCGGCAUCU---CCA

seq2 (((((((...(((((((((( (.....)))......))))))))......(((((.......))))))))))))....
     CGGCACGUAGCGCAGCCUGG-UAGCGCACCGUCCUGGGGUUGCGGGGGUCGGAGGUUCAAAUCCUCUCGUGCCGACCA

max  .((((((( ((((...  ..   )))))))))))((((((((((((((((((...)))))))))))))))))))...
     GGGCAUGU-GCGCAGU--GG---GCGCACAUGCCUCGGGAUGAGGGGGUCCGAGGUUCGGACCCCCUCAUCCCGACCA
                                                   ↑                          ↑
```

Figure 2. Alignment and predicted secondary structure of the two sequences, *seq1* and *seq2*, used to construct the `trna2` model, and the sequence and secondary structure of the maximal parse of `trna2` aligned according to the states of `trna2` emitting the symbols. In the two positions indicated with an ↑ the sequence of the maximal parse does not match any of the two other sequences.

not even any of the shared base pairs are present in the structure found by the maximal parse. Instead, one can observe that the structure of the maximal parse is much more dense than the structures of *seq1* and *seq2*, with only a few bases not being part of one of the two uninterrupted helices constituting the structure. This is probably an indication of the main problem of using the maximal parse to predict the secondary structure. In each position we can choose a symbol so as to construct a sequence with a structure of very high probability, i.e. the maximal parse seems to be a question of highly probable structures finding matching sequences instead of what we are really looking for, highly probable sequences finding a matching structure. This is further supported by the two positions where the maximal parse disagrees with *seq1* and *seq2*, especially as *seq1* and *seq2* agree in these positions – the sequence obtained by changing the symbols in these two positions to the symbols shared by *seq1* and *seq2* would be roughly 80 times as probable in the hidden Markov model.

Another problem is that the maximum is not very good for discriminating states that exhibit complementarity from states that do not exhibit complementarity. E.g. a state that has probabilities 0.5 for emitting either a `C` or a `G` gets a lower probability if paired with a state identical to itself, than if paired with a state that has probability 0.51 for emitting a `C` and 0.49 for emitting a `A`. However, having the framework of algorithm 1 it is easy to modify the details to accommodate a scoring of combinations of pairs of states and derivations introducing base pairs that captures complementarity better. Furthermore, the co-emission probability will indeed capture that the two `C`/`G`-emitting states exhibit better complementarity than the `C`/`G`-`C`/`A` pair. Thus, a better idea of a common structure might be obtained by looking at the probability that two states emit symbols that are base paired for all pairs of non-silent states, similar to [11,19]. Indeed, as the dependencies of the energy rules commonly used for RNA secondary structure prediction can be captured by a context-free grammar, one can also combine the computation of the co-emission probability as discussed in this paper with the computation of the equilibrium partition function presented in [11] to obtain probabilities for base pairing of positions including both the randomness of base pairing captured by the partition functions as well as the variability of a family of sequences captured by a (profile) hidden Markov model.

7 Discussion

In this paper we have considered the problem of comparing a hidden Markov model with a stochastic context-free grammar. The methods presented can be viewed as natural generalizations of methods for analyzing strings by means of stochastic context-free grammars, or of the idea of comparing two hidden Markov models in [10]. A natural question is thus whether we can further extend the results to comparing two stochastic context-free grammars. If we could determine the co-emission probability of – or just the maximal joint probability of a pair of parses in – two stochastic context-free grammars, we could also determine whether the languages of two context-free grammars are disjoint, simply by assigning a uniform probability distribution to the derivations of each of the variables and asking whether the computed probability is zero. However, a well-known result in formal language theory states that it is undecidable whether the languages of two context-free grammars are disjoint [17, Theorem 11.8.1]. Hence, we cannot generalize the methods presented in this paper to methods comparing two stochastic context-free grammars with a precision that allows us to determine whether the true probability is zero or not.

In [10] we use the co-emission probability to compute the L_2-distance between the probability distributions of two hidden Markov models. Having demonstrated how to compute the co-emission probability between a hidden Markov model and a stochastic context-free grammar, the only thing further required to compute the L_2-distance is the co-emission probability of the grammar with itself. As stated above, the problem of computing the co-emission probability between two stochastic context-free grammars is undecidable, but one could hope that computing the co-emission probability of a stochastic context-free grammar with itself would be easier. However, given two stochastic context-free grammars G_1 and G_2 we can construct an aggregate grammar G' where the start symbols of G_1 and G_2 are chosen with equal probability one half. It is easy to see that the co-emission probability of G' with itself is the sum of the co-emission probabilities of G_1 and G_2 with themselves plus twice the co-emission probability between G_1 and G_2. Hence, computing the co-emission probability of a stochastic context-free grammar with itself, or the L_2- or Hellinger-distances between the probability distributions of a context-free grammar and a hidden Markov model, is as hard as computing the co-emission probability between two stochastic context-free grammars.

In this paper we have presented the co-emission probability as a measure for comparing two stochastic models. However, the co-emission probability has at least two other interesting uses. First, it allows us to use one model as a prior for training the other model, e.g. using the distribution over sequences of a hidden Markov model as our prior belief about the distribution over sequences for a stochastic context-free grammar we want to construct. Secondly, it allows us to compute the probability that two stochastic models have independently generated a sequence s given the two models generate the same sequence. I.e. we can combine two models under the assumption of independence.

References

1. K. Asai, S. Hayamizu, and K. Handa. Prediction of protein secondary strucuture by the hidden markov model. *Computer Applications in the Biosciences (CABIOS)*, 9:141–146, 1993.
2. J. K. Baker. Trainable grammars for speech recognition. In *Speech Communications Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.
3. G. A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
5. R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probalistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
6. J. Håstad, S. Phillips, and S. Safra. A well characterized approximation problem. *Information Processing Letters*, 47(6):301–305, 1993.
7. B. Knudsen and J. Hein. RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15:446–454, 1999.
8. A. Krogh. Two methods for improving performance of an HMM and their application for gene finding. In *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 179–186, 1997.
9. A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.
10. R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen. Metrics and similarity measures for hidden Markov models. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 178–186, 1999.
11. J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990.
12. L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.
13. E. Rivas and S. R. Eddy. The language of RNA: A formal grammar that includes pseudoknots. *Bioinformatics*, 16(4):334–340, 2000.
14. Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22:5112–5120, 1994.
15. D. B. Searls. The linguistics of DNA. *American Scientist*, 80(579–591), 1992.
16. E. L. L. Sonnhammer, G. von Heijne, and A. Krogh. A hidden Markov model for predicting transmembrane helices in protein sequences. In *Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, 1998.
17. T. A. Sudkamp. *Languages and Machines*. Computer Science. Addison-Wesley Publishing Company, Inc., 1998.
18. Y. Uemura, A. Hasegawa, S. Kobayashi, and T. Yokomori. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277–303, 1999.
19. M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244:48–52, 1989.