Vol. 18 Suppl. 2 2002
Pages S250–S259

An approximate matching algorithm for finding (sub-)optimal sequences in S-attributed grammars

J. Waldispühl, B. Behzadi and J.-M. Steyaert

LIX, École Polytechnique, Palaiseau, 91128, France

Received on April 8, 2002; accepted on June 15, 2002

ABSTRACT

Motivation: S-attributed grammars (a generalization of classical Context-Free grammars) provide a versatile formalism for sequence analysis which allows to express long range constraints: the RNA folding problem is a typical example of application. Efficient algorithms have been developed to solve problems expressed with these tools, which generally compute the optimal attribute of the sequence w.r.t. the grammar. However, it is often more meaningful and/or interesting from the biological point of view to consider almost optimal attributes as well as approximate sequences; we thus need more flexible and powerful algorithms able to perform these generalized analyses.

Results: In this paper we present a basic algorithm which, given a grammar G and a sequence ω, computes the optimal attribute for all (approximate) strings $\omega' \in L(G)$ such that $d(\omega, \omega') \leq M$, and whose complexity is $\mathcal{O}(n^{r+1})$ in time and $\mathcal{O}(n^2)$ in space (r is the maximal length of the right-hand side of any production of G). We will also give some extensions and possible improvements of this algorithm.

Keywords: S-Attribute Grammar, Approximate matching, Dynamic Algorithm.

Contact: waldispu@lix.polytechnique.fr

INTRODUCTION

An important question in biology is structure prediction; it concerns RNA sequences as well as proteins and a number of methods, algorithms and software packages have been proposed and developed in the last ten years. Two main theoretical approaches are classical: the first one is based on automata and formal language theory (see Myers (1995); Myers and Miller (1989); Zuker and Sankoff (1984); Zuker (1989a)) and the second one relies on heuristics inspired from simulated annealing, neural networks or genetic algorithms. Some authors have proposed mixed approches combining pieces of the methods mentionned above by means of (now classical) constraint programming (see Gaspin *et al.* (1995)). In this paper we propose to revisit the formal language approach; Zuker and Sankoff (1984) proposed initially to compute the best possible free energy of a structure by dynamic programming, and Searls made an important contribution showing that such constraints could be expressed in terms of context-free grammars (Searls, 1992; Vauchaussade de Chaumont, 1985). Finally, E.W.Myers (Myers, 1995; Myers and Miller, 1989) introduced the question of best approximation for a given structure. Some years ago, F.Lefebvre (Lefebvre, 1995, 1996) proposed a very general framework to express and formalize quite general context-free, regular, probabilistic constraints, etc. by developping S-attributed grammars specialized to biological problems: this was a significant extension of Knuth's original work (Knuth, 1968). However, Lefebvre's algorithm and software only compute the optimal structure on a given sequence. It is well-known that suboptimal structures are of interest as well as suboptimal approximate sequences behaving similarly to a given model. Therefore it would be interesting to extend the S-attributed grammatical framework to these two problems, thus generalizing Zuker's (Zuker, 1989a) and Myers's (Myers, 1995) previous works. Recently, El-Mabrouk and Raffinot (2002) have presented an algorithm that searches for all approximate matches of a structure that could be described by a restricted regular expression and a kind of mirror mechanism. Since we use plain context-free grammars to describe the possible structures, our approach is more powerful and flexible. The aim of our algorithm is to find the optimal (or suboptimal) structure rather than report each potential one. This optimization could be done because we extend our formalism to S-Attributed grammars. Moreover, insertion, deletion and mutation costs are defined in a flexible way which allows us to define precisely an alignment. The main advantage of our framework is that it allows to formally specify a number of local and global constraints and separetely chemical or physical properties (or attributes) related to energy, similarity, evolution or whatever, and then to let the system produce (by means of a general mechanism) the particular program that will solve the optimization constraint problem. In Section 2, we recall a number of definitions and facts related to S-attributed grammars. In Section 3, we present in detail the basic parsing algorithm which, given a formal description (grammar and criterion) and a sequence, returns a set of approximate sequences to the input which optimize the input criterion. This so-called AMSAG algorithm is proved to be correct and its complexity is proved to be $\mathcal{O}(n^{r+1})$ in time and $\mathcal{O}(n^2)$ in space, as one could expect, r being the length of righthand side strings in the productions. In Section 4, we discuss some extensions in order to produce approximate sequences whose attribute value is suboptimal; this does not increase dramatically the algorithmic complexity while giving useful information. We also propose a heuristic strategy in order to enhance time complexity, when the optimization is done on several criteria; we show that some preprocessing allows important savings in complexity. Finally, some experimental results are presented in Section 5, to illustrate the effective behavior of our algorithm on real biological sequences.

CONTEXT-FREE AND S-ATTRIBUTED GRAMMARS

We recall in this section the basic definitions of context-free grammars and of their natural extension to S-attributed grammars. These definitions are adapted to the fact that we use this formalism to model biological properties of sequences.

Definition 1. (**Context-free grammar**)
A context-free grammar $G = (V_T, V_N, P, S)$ consists of a finite set of terminals V_T, a finite set of nonterminals V_N such that $V_T \cap V_N = \emptyset$, a finite set of productions (rewriting rules) P and a start symbol $S \in V_N$. Let $V = V_T \cup V_N$ denote the vocabulary for the grammar. Each production in P has the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in V^\star$. A is the left-hand side of the production and α its right-hand side.

The transitive closure of the derivation relation $\rightarrow$ is denoted $\overset{\star}{\rightarrow}$. A derivation tree is the planar representation of a sequence of derivations $A \rightarrow \alpha$ such that $A \in V_N$ and $\alpha \in V^\star$. The set of strings in $V_T^\star$ derived from S is called the language generated by G and it is denoted by $L(G)$. The empty string is denoted by ε.

In order to keep the number of derivation trees finite for a given word $\omega \in L(G)$, we assume that the grammar is non-circular, which means that no nonterminal A may verify $A \overset{+}{\rightarrow} A$. We also assume that the grammar is epsilon-free (i.e. it has no rules of the form $A \rightarrow \varepsilon$). An ambiguous grammar is a grammar for which there exists a string of symbols having at least two different derivation trees. For example, the grammar whose derivation trees describe t-RNA secondary structures has to be ambiguous because a given RNA has potentially several different secondary structures. Parsing is the process of finding a derivation tree for a string in $L(G)$, which is called the parse tree of the sequence.

S-attributed context-free grammars, which are a proper subset of attributed-grammars introduced by Knuth in his seminal paper (Knuth, 1968), are an extension of context-free grammar allowing the assignment of a value (called attribute) to every vertex of a derivation tree.

Definition 2. (**S-attributed grammar**)
An S-attributed grammar, denoted by $G = (V_T, V_N, P, S, \mathcal{A}, S_\mathcal{A}, F_P)$, is an extension of the context-free grammar $G = (V_T, V_N, P, S)$; an attribute $x \in \mathcal{A}$ is attached to each symbol $X \in V$, and a string of attributes $\lambda \in \mathcal{A}^\star$ to each string $\alpha \in V^\star$. $S_\mathcal{A}$ is a function from V_T to $\mathcal{A}$ assigning attributes to terminals. F_P is a set of functions from $\mathcal{A}^\star$ to $\mathcal{A}$. A function $f_{A\rightarrow\alpha}$ is in F_P for every production $A \rightarrow \alpha$ in P.

The attribute of a string $\alpha \in V^\star$, denoted by λ_α, is the concatenation of the attributes of the symbols in α. When a function $f_{A\rightarrow\alpha}$ is applied to the attribute λ_α derived from A it returns the attribute $x = f_{A\rightarrow\alpha}$ of $\mathcal{A}$. Thus, the functions of F_P determine the bottom-up computation of the attributes of nonterminals A in derivations $\alpha A \beta \overset{\star}{\rightarrow} u$, where u must belong to $L(G)$ so that the attribute of A be computable.

Example 1. We consider the following ambiguous context-free grammar for arithmetical expressions.

$$S \rightarrow S + S$$
$$S \rightarrow S \times S$$
$$S \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.$$

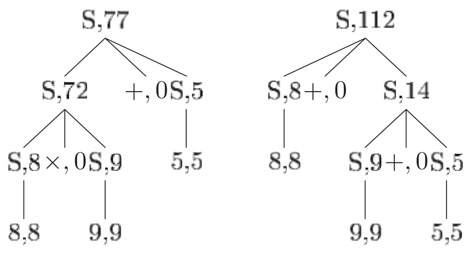
We extend it into an S-attributed grammar such that each attribute stores the value of the arithmetical expression corresponding to the derivation subtree rooted at this node.

$$\mathcal{A} = \mathbb{N}$$

$$S_\mathcal{A}(0) = 0$$
$$\vdots$$
$$S_\mathcal{A}(9) = 9$$
$$S_\mathcal{A}(+, \times) = 0$$

$$F_P = \left\{ \begin{array}{l} f_{S\rightarrow S+S}(x0z) = x + z \\ f_{S\rightarrow S\times S}(x0z) = x \times z \\ f_{S\rightarrow a\in V_T}(x) = x \end{array} \right\}$$

Thus, the sequence $8 \times 9 + 5$ can be parsed in different ways corresponding to different derivation trees, hence producing different values



As previously noted, grammars used for biological sequence analysis are ambiguous because they aim at describing the space of all possible configurations. Attributes are designed to give an evaluation of the quality for each of them which gives a choice criterion. For example, if the attribute of a vertex is an energy or a probability, the criterion may be the selection of the derivation tree with the lowest energy or the highest probability at the root. (But attributes are not restricted to simple real values in general even if this paper deals mainly with this kind). The selection of the best possible value for an attribute is done by means of a function called optimization constraint.

DEFINITION 3. (**Optimization constraint**) Let G be an S-attributed grammar and let $a_1 \cdots a_n$ be a string of L(G). Let x be the attribute of a derivation $A \to \alpha_1 \to \cdots \to \alpha_k \to a_{i+1} \cdots a_j$ and let x' be the attribute of another derivation $A \to \alpha'_1 \to \cdots \to \alpha'_k \to a_{i+1} \cdots a_j$. The optimisation constraint $\mathcal{C}_\mathcal{A}$ associated with the nonterminal A takes as an input the attributes x and x' and returns another attribute x''. We note $x'' = \mathcal{C}_\mathcal{A}(x, x')$.

In practice, $\mathcal{C}_\mathcal{A}$ will be a comparison function between x and x' which returns the optimal attribute (the lowest energy, if attributes are free energy). Finally, we denote by λ_ω the optimal attribute of a string $\omega \in L(G)$. The grammars which are used in this paper are all supposed to satisfy a suboptimality hypothesis: *every optimal attribute is obtained when all the attributes associated with the right-hand side variables of the production are themselves optimal.*

THE AMSAG ALGORITHM

In this section we describe the main algorithm that we propose for best approximate parsing of sequences according to an S-attributed grammar. Then we prove its correctness and analyze its complexity.

Notations and Definitions

In most cases, we will use Myers's notations (Myers, 1995). Let u and v be two words of $V_T^\star$. An alignment of size N of u and v consists of two strings $a_1 a_2 \dots a_N = u$ and $b_1 b_2 \dots b_N = v$ where $a_i, b_i \in (V_T \cup \{\epsilon\})$ with the condition that $(a_i, b_i) \neq (\epsilon, \epsilon)$. We introduce a real positive cost function $\sigma : (V_T \cup \{\epsilon\}) \times (V_T \cup \{\epsilon\}) \to \mathbb{R}^+$ to define the score (or cost) of an alignment: $\delta(u, v) = \sum_i \sigma(a_i, b_i)$. The distance between u and v which is written as $d(u, v)$ is defined as the minimum score for all possible alignments between them. Let $x, y \in V_T$, then $\sigma(x, y)$ refers to *MutCost(x,y)*, $\sigma(x, \epsilon)$ to *DelCost(x)* and $\sigma(\epsilon, x)$ to *InsertCost(x)*, respectively the mutation, deletion and insertion costs used in the program of Figure 1. In the program we use $DelCost(\omega)$, where $\omega \in V_T^\star$, which is a natural extention of $DelCost(x)$ for a terminal x. Furthermore, we assume that $\sigma(x, y) \le \sigma(\epsilon, y) + \sigma(x, \epsilon)$ which is natural. Let $\omega = \omega_1 \cdots \omega_n$ be a word of $V_T^\star$ such that $\omega_i \in V_T$ for all $i \in \{1 \cdots n\}$, and $\omega_{ij} = \omega_i \cdots \omega_j$ be a substring of ω. We define the function $Best(A, i, j, e)$ which returns the optimal attribute value of any string ω' derived from A whose distance from ω_{ij} is at most e. Formally: $Best(A, i, j, e) = max\{\lambda_{\omega'} |$ for all derivation sequences $A \overset{\star}{\to} \omega'$ where $d(\omega', \omega_{ij}) \le e\}$. We will note by $G = (V_T, V_N, P, S, \mathcal{A}, S_\mathcal{A}, F_P)$ the S-attribute grammar and will thoroughly use the following convention on variables:

- $u, v \in V^\star$
- $\alpha \in \mathcal{A}^\star$
- $\Gamma \in P$

For a production rule $\Gamma = A \to w$ and a string $u = u_1 \cdots u_n$ with $u_i \in V$, we define the following functions :

- *LeftSide*: $P \to V_N$ such that *LeftSide*$(\Gamma) = A$
- *RightSide*: $P \to V^\star$ such that *RightSide*$(\Gamma) = w$
- *head*: $V^\star \to V$ such that *head* $(u) = u_1$
- *tail*: $V^\star \to V^\star$ such that *tail*$(u) = u_2 \cdots u_n$

The variables of the grammar are topologically sorted according to the order relation induced by the set of the unit productions($A \to B$ means $B < A$).

Algorithm description

The input of the problem consists of an S-attributed grammar G, a string ω and $MaxErr$ an upper bound for the distance. Here we suppose that $MaxErr$, $MutCost(x, y)$, $DelCost(x)$ and $InsertCost(x)$ are integers for all $x, y \in V_T$. The AMSAG algorithm is based on dynamic programming and fills dynamically an array $Best[V, i, j, NbErr]$ of real numbers. $Best[V, i, j, NbErr]$ stores the optimum value of the attribute for a string ω' (derived from V) whose distance from ω_{ij} is at most $NbErr$.

The AMSAG algorithm finds the optimal attribute. In order to find the approximate string corresponding

```
ALGORITHM 1. AMSAG($G$, $\omega$, $MaxErr$)
0.  Var $Best$ : Array [$V$, $1..N+1$, $0..N$, $0..MaxErr$] of real
1.  For $NbErr \leftarrow 0$ to $MaxErr$ do
2.    For $len \leftarrow 0$ to $N$ do
3.      For $i \leftarrow 1$ to $N - len + 1$ do
4.        $j \leftarrow i + len - 1$
5.        For $A \in V$ do
6.          $Best[A, i, j, NbErr] = -\infty$
7.          For $\Gamma \in P$ and $LeftSide(\Gamma) = A$ do
8.            $Cur\_i \leftarrow i$ , $Cur\_j \leftarrow j$, $Cur\_err \leftarrow NbErr$, $Cur\_V \leftarrow A$
9.            FindAlignment($\Gamma$, $RightSide(\Gamma)$, $\epsilon$, $i$, $j$, $NbErr$)
```

Fig. 1. AMSAG: core algorithm

```
ALGORITHM 2. FindAlignment($\Gamma$, $u$, $\alpha$, $i$, $j$, $NbErr$)
1.  $A \leftarrow LeftSide(\Gamma)$
2.  If ($NbErr < 0$) Then Abort
3.  If ( $i > j$ ) Then
4.    $i \leftarrow 1$ , $j \leftarrow 0$
5.  $a \leftarrow head(u)$
6.  $v \leftarrow tail(u)$
7.  If ($v = \epsilon$ ) Then
8.    If ($a \in V_T$) Then
9.      If ($NbErr \geq d(a, \omega_{ij})$) Then
10.       $Best[Cur\_V, Cur\_i, Cur\_j, Cur\_err]$ =
            $\mathcal{C}_A(Best[Cur\_V, Cur\_i, Cur\_j, Cur\_err], f_\Gamma(\alpha \cdot \lambda_a))$
11.     Else
12.       $Best[Cur\_V, Cur\_i, Cur\_j, Cur\_err]$ =
            $\mathcal{C}_A(Best[Cur\_V, Cur\_i, Cur\_j, Cur\_err], f_\Gamma(\alpha \cdot Best[a, i, j, NbErr]))$
13.  Else
14.    If ($a \in V_T$) Then {
15.      FindAlignment($\Gamma$, $v$, $\alpha \cdot \lambda_a$, $i$, $j$, $NbErr$ - $InsertCost(a)$)
16.      For $k \leftarrow i$ to $j$ do
17.        If ($a = \omega_k$) Then
18.          FindAlignment($\Gamma$, $v$, $\alpha \cdot \lambda_a$, $k+1$, $j$, $NbErr$ - $DelCost(\omega_{ik})$)
19.        Else
20.          FindAlignment($\Gamma$, $v$, $\alpha \cdot \lambda_a$, $k+1$, $j$, $NbErr$ - $DelCost(\omega_{ik})$ - $MutCost(a, \omega_k)$)
21.    }
22.    Else
23.      For $k \leftarrow i-1$ to $j$ do
24.        For $e \leftarrow 0$ to $NbErr$ do
25.          If ($Best[a, i, k, e] > -\infty$)
26.            FindAlignment($\Gamma$, $v$, $\alpha \cdot Best[a, i, k, e]$, $k+1$, $j$, $NbErr$ - $e$)
```

Fig. 2. AMSAG: recursive computation of $BEST$

to the optimal value and the corresponding derivation sequence of this string in the grammar in addition to the value itself, we use an additional array to keep in memory the rule and the corresponding factorization of the ω_{ij} for the $Best[V, i, j, NbErr]$ (this information will be refered to as *cut points*). Whenever we update this

cell with a better value, we change accordingly the cut points array. The memorized dynamic implementation of the algorithm does not fill the useless cells of the array $Best$: thus it has a better execution time. The fact that $Best[V, i, j, NbErr] \leq Best[V, i, j, NbErr + k]$ (for a $k > 0$) leads us to another improvement in time: whenever $Best[V, i, j, NbErr] = Best[V, i, j, NbErr + k]$, we store $-\infty$ in $Best[V, i, j, NbErr + k]$ since it is useless, which decreases notably the execution time.
The core algorithm, represented in Figure 1, is inspired by Myers's one. Each cell $Best[V, i, j, NbErr]$ is progressively filled according to the length of ω_{ij} and the error bound $NbErr$, by a call to the function FindAlignment described in Figure 2. The input of FindAlignment has six different fields: a production rule noted Γ, a suffix of the right-hand side of Γ noted u, an attribute string α which is the attributes of the prefix of the right-hand side already processed in the recursive calls, two indices i and j representing the substring ω_{ij} of the input string ω, and an error bound $NbErr$. By this call, FindAlignment finds recursively the alignments of the strings derived by the right-hand side of production Γ and the substring (ω_{ij}) in which we accept an error bounded by $NbErr$. In the consecutive recursive calls the right-hand side elements of the production rule Γ are assigned to a prefix of the ω_{ij} one by one. FindAlignment$(\Gamma, u, \alpha, i, j, NbErr)$ produces all possible alignments of $head(u)$ with a prefix of ω_{ij} and calls itself recursively. A negative $NbErr$ in line 2 means that the alignment has exceeded the error bound and so this function call is aborted. Now let $NbErr$ be non-negative. If the right-hand side of the production rule Γ has been scanned completely (i.e. u is the empty word ε), the corresponding attribute of this alignment will be compared to the value already stored in $Best$ and will replace it if it has a better score (lines 7 to 12). Otherwise we have 2 cases : If $head(u)$ (noted a) is a terminal (lines 14 to 20), we consider all the possible alignments between a and the prefixes of ω_{ij} (including the empty one at line 15) and we update the error bound by substracking the cost of the alignment (last field of the arguments given to FindAlignment). In the other case (lines 23 to 26), a is a nonterminal and we simply use the attributes already stored in $Best$.

Proof

We first state without formal proof a rather natural result that can be argued geometrically.

LEMMA 1. *Given two words w and w' with a factorization of $w = w_1w_2$, there exists a factorization of $w' = w'_1w'_2$ such that $d(w, w') = d(w_1, w'_1) + d(w_2, w'_2)$.*

We now prove the main result which validates the algorithm:

PROPOSITION 1. *Let $\lambda(A, \omega, e)$ be the optimal value of the attribute of a derivation sequence $A \overset{\star}{\rightarrow} \omega'$ for all sequences ω' such that $d(\omega, \omega') \leq e$. Then, there is a factorization $\omega_1 \cdots \omega_n = \omega$ where some substrings ω_k can possibly be empty, such that $\lambda(A, \omega, e) = max_{A\rightarrow\alpha\in P}\{f_{A\rightarrow\alpha}(\lambda(\alpha_1, \omega_1, e_1) \cdots \lambda(\alpha_n, \omega_n, e_n)) \mid e_1 + \cdots + e_n \leq e\}$*

PROOF. Let $A \rightarrow \alpha = \alpha_1 \cdots \alpha_n \overset{\star}{\rightarrow} \omega'$ be the first production of the derivation sequence with optimal attribute $\lambda_{\omega'}$. Then, each λ_{α_k} is also optimal and we have a decomposition $\omega' = \omega'_1 \cdots \omega'_n$ such that $\lambda_{\alpha_k} = \lambda_{\omega'_k}$. Assume that $d(\omega, \omega') \leq e$, then by Lemma 1 there exists a decomposition $\omega = \omega''_1 \cdots \omega''_n$ such that $d(\omega, \omega') = d(\omega''_1, \omega'_1) + \cdots + d(\omega''_n, \omega'_n)$. We have also $\lambda_{\omega'_k} \leq \lambda(\alpha_k, \omega''_k, d(\omega''_k, \omega'_k)) = \lambda_k$, so $\lambda_{\omega'} \leq f_{A\rightarrow\alpha}(\lambda_1 \cdots \lambda_n)$. But, we note that $f_{A\rightarrow\alpha}(\lambda_1 \cdots \lambda_n) \in \{f_{A\rightarrow\beta}(\lambda(\beta_1, \omega_1, e_1) \cdots \lambda(\beta_m, \omega_m, e_m)) \mid e_1 + \cdots + e_m \leq e\}$ and then $\lambda_{\omega'} \leq max_{A\rightarrow\alpha\in P}\{f_{A\rightarrow\alpha}(\lambda(\alpha_1, \omega_1, e_1) \cdots \lambda(\alpha_n, \omega_n, e_n)) \mid e_1 + \cdots + e_n \leq e\}$. For the opposite inequality, we note that all the attributes in $\{f_{A\rightarrow\alpha}(\lambda(\alpha_1, \omega_1, e_1) \cdots \lambda(\alpha_n, \omega_n, e_n)) \mid e_1 + \cdots + e_n \leq e\}$ come from words ω'' such that $A \overset{\star}{\rightarrow} \omega''$ and $d(\omega, \omega'') \leq e$. Then, it is clear that $\lambda(A, \omega, e)$ is at least better than every attribute in this set. We conclude that $\lambda_{\omega'} \geq max_{A\rightarrow\alpha\in P}\{f_{A\rightarrow\alpha}(\lambda(\alpha_1, \omega_1, e_1) \cdots \lambda(\alpha_n, \omega_n, e_n)) \mid e_1 + \cdots + e_n \leq e\}$ which completes the proof.

Finally, we note that $FindAlignment(\Gamma, u, \alpha, i, j, NbErr)$ recursively produces all factorizations of ω_{ij} with $|u|$ parts such that the cost of the final alignment is at most $NbErr$: the proof is done by induction.

Complexity analysis

The complexity of the algorithm in space is $\mathcal{O}((|V_T| + |V_N| \times Max\_Err)n^2)$. By preprocessing, $d(a, \omega_{ij})$ is computed and stored in an array in time $\mathcal{O}(n^2)$. For a production in which the length of the right-hand side is r $FindAlignment$ chooses $r - 1$ cut points to produce all possible decompositions. So $FindAlignment$ is processed in $\mathcal{O}(n^{r-1})$ and the total complexity of algorithm in time will be $\mathcal{O}(|P|.(Max\_Err)^r.n^{r+1})$, hence $\mathcal{O}(n^{r+1})$ for a given grammar G and Max_Err.

THE VARIANTS OF THE ALGORITHM

We propose two variants closely related to this basic algorithm, which could be useful to analyze the biological sequences.

The kth optimal value

In practice, one is often interested not only in the optimal attribute but also in the best sub-optimal k values. An immediate result of suboptimality hypothesis is that every

kth optimal attribute is obtained when all the attributes associated with the right-hand variables of the production are themselves one of the k optimal values. In order to solve the k optimal value problem in our algorithm we have to store the sorted list of k optimal values in $Best[V, i, j, NbErr]$ and the rest of the algorithm will keep the same. Thus the space complexity is multiplied by k and the time complexity is multiplied by $\mathcal{O}(k^r.log(k))$.

Multi-criteria optimization

An additional parameter. We now discuss the trade-off between the attribute value of a (sub-)optimal solution and its distance to the input sequence. It can be natural to consider that a string with a lower attribute value can be more meaningful if it is very close to the input one. Therefore, we introduce a valuation function $\mathcal{V} : V_T^\star \times V_T^\star \rightarrow \mathbb{R}$ which gives a score between two strings. The criteria take into account here are the value of the attribute of the output string and its distance to the input string. Hence, we define another function $\mathcal{W} : \mathcal{A} \times \mathbb{R} \rightarrow \mathbb{R}$ such that $\mathcal{W}(\lambda_{\omega'}, d(\omega, \omega')) = \mathcal{V}(\omega, \omega')$. By extension, this function can also be applied to every possible alignment between ω and ω' as $\mathcal{W}(\lambda_{\omega'}, \delta(\omega, \omega'))$. We propose in this paper $\mathcal{W}(\lambda_{\omega'}, d(\omega, \omega')) = \frac{\lambda_{\omega'}}{d(\omega,\omega')+1}$, although any function which increases with $\lambda_{\omega'}$ and decreases with d(ω,ω') would be convenient.

A heuristic sampling approach. An additional property on $\mathcal{W}$ will allow further simplification. Let us make the following assumption (ω_i is now a substring and not only a terminal):
Let $S \rightarrow \alpha$ be a production rule of G such that $\alpha = \alpha_1 \cdots \alpha_n$ and let us consider two decompositions of ω and ω' into n parts: $\omega = \omega_1 \cdots \omega_n$ and $\omega = \omega'_1 \cdots \omega'_n$. Let F be the value function of the attribute associated with $S \rightarrow \alpha$. Thus $\mathcal{W}(\lambda_\omega, \delta(\omega, \omega')) = \mathcal{W}(F(\lambda_{\omega'_1} \cdots \lambda_{\omega'_n}), \delta(\omega_1, \omega'_1) + \cdots + \delta(\omega_n, \omega'_n))$ is optimal if and only if each $\mathcal{W}(\lambda_{\omega'_i}, \delta(\omega_i, \omega'_i))$, where $i \in 1 \cdots n$, is optimal.

This allows to use a dynamic programming algorithm to fill the array $Best[A, i, j]$ such that each cell $Best[A, i, j]$ stores the couple $(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}))$ which optimizes $\mathcal{W}(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}))$.

Although this property does not hold in general, it is possible to keep enough information in each cell if we can find a lower bound $T(A, i, j)$ for each $\mathcal{W}(Best[A, i, j])$: then we will be able to decide if a pair $(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}))$ must be stored or rejected. In this case, we can build a dynamic algorithm which stores in each cell $Best[A, i, j]$ the list of all candidates. Without loss of generality, assume that $\mathcal{W}(\lambda, \delta) = \frac{\lambda}{\delta+1}$. Let $\omega \in L(G)$, therefore λ_ω is defined and we have the inequality: $\mathcal{W}(\lambda_{\omega'}, d(\omega, \omega')) \geq \lambda_\omega$, if ω' is the optimal word.

Let us assume that there exists a constant $K \leq 1$ such that for a given grammar G and two *string* factorizations, $\omega = \omega_1 \cdots \omega_r$ and $\omega' = \omega'_1 \cdots \omega'_r$, if there is $i \in \{1 \cdots r\}$ such that $\mathcal{V}(\omega_i, \omega'_i) < K \cdot \lambda_{\omega_i}$ then $\mathcal{V}(\omega, \omega') < \lambda_\omega$. Then $T(A, i, j) \geq K \cdot \lambda_{\omega_{ij}}$. Hence we have the following algorithm.

Algorithm. This algorithm operates in two stages. A preprocessing stage computes the bounds $T(A, i, j)$ for each $A \in V_N$ and $0 \leq i \leq j \leq N$, and the core of our algorithm is dedicated to fill the array $Best[A, i, j]$ with the list of candidate pairs $(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}))$. The input is formed by an S-attributed grammar G, a word $\omega = \omega_1 \cdots \omega_N$ and a value for K.
Pairs $(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}))$ are computed in the same way as in the basic algorithm. We produce all possible alignments between the right-hand side of Γ and an ω_{ij}. A score $\mathcal{W}(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}))$ is computed for each alignment. Only the alignments with a score greater than the preprocessed threshold $T(A, i, j)$ are kept in the array. In the case when two alignments correspond to the same approximation ω'_{ij}, we only keep the one of greatest cost $\mathcal{V}(\omega_{ij}, \omega'_{ij})$. Hence, we only need keep in memory 3-tuples $(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}), \omega'_{ij})$, as shown in Figure 5. Finally, line 7 could be substitued by the procedure shown in Figure 5.

APPLICATIONS

We have applied our algorithm to real biological sequences in order to check its complexity and efficiency. The grammar used for the tests (a rather naive one) is described in paragraph 1 while experimentations are developped in paragraph 2.

Grammar description

In order to perform our tests, we have used a grammar describing a simplified thermodynamic model of RNA folding without pseudo-knot. To compute the free energy of a folding, only basepair are considered and not stacking-pairs as in Zuker's model. Obviously, the destabilisation free energy is also assigned to each substructure as end loops, bulges, interiors loop and multi-loops, but in order to simplify our model they are restricted to be affine functions. All the parameters have been computed by interpolating the values used in Lefebvre (1995). In this grammar, the maximal length r of the right-hand side of any production in P is 3. In the same way, mutation, deletion and insertion costs have been set to 1 for any parameter. Formally:

$$\sigma(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$$

where $x, y \in V_T \cup \{\varepsilon\}$.

```
Preprocessing:
    1. Compute the Array $Best$ with basic algorithm and a maximal error cost
       of 0,
    2. $T(A, i, j) \leftarrow K \cdot Best[A, i, j, 0]$.
```

Fig. 3. Multi-criteria optimization: preprocessing

```
Core:
    0. Var $Best$ : Array [V, 1..N+1, 0..N] (different from the $Best$ of Preprocessing)
    1. For len $\leftarrow$ 0 to N do
    2.     For i $\leftarrow$ 1 to N - len + 1 do
    3.         j $\leftarrow$ i + len - 1
    4.         For $A \in V$ do
    5.             Best[A, i, j] = $\emptyset$
    6.             For ($\Gamma \in P$ and LeftSide($\Gamma$) = $A$) do
    7.                 Compute all the ($\lambda_{\omega'_{ij}}$,$\delta(\omega_{ij}, \omega'_{ij})$) and add them into the list
                         $Best[A, i, j]$ if $\mathcal{W}(\lambda_{\omega_{ij}}, \delta(\omega_{ij}, \omega'_{ij})) \geq T(A, i, j)$.

Output: Choose in $Best[S, 1, N]$ the optimal pair ($\lambda_{\omega'}$,$\delta(\omega, \omega')$).
```

Fig. 4. Multi-criteria optimization: core algorithm

Experimental results

Computational time. Our software has been runned on a Dec-Alpha with a 667 Mhz processor. Yet, only the basic algorithm is implemented but it is enough to check the theorical complexity. Tests have been performed on a data set of 8 real RNA sequences (see Table 1). The graphics present the results in two different ways: The first one (Figure 6) displays the running time of the algorithm versus the input string length with a fixed maximal error cost, and the second one (Figure 7) displays the running time versus the maximal error cost with a fixed input string length. The use of a logarithmic scale clearly shows that the computational behavior is polynomial and at least as good as the theorical complexity. A curve parallel to the theorical complexity is displayed with dotted lines in Figures 6 and 7, in order to show that the pratical complexity could be cleary better than the expected one. With this grammar, the complexity with a fixed error bound is closer to $\mathcal{O}(n^3)$ than $\mathcal{O}(n^4)$; similarly when the input length is fixed the complexity is closer to $\mathcal{O}(NbErr^2)$ than $\mathcal{O}(NbErr^3)$.

Example. The aim of this paragraph is to prove that our algorithm could be useful to analyse biological sequences. We don't pretend to solve here the biological problem in its totality, but rather to illustrate the capacity of our formalism and software. The algorithm has been run on a data set of 92 t-RNA of Escherichia coli from Lowe and Eddy (1997). We show an example characteristic of the results obtained in Figures 8 and 9. A folding is represented by a well bracketed word where two associated brackets correspond to a basepair. Our software displays the output as a string associated with a folding and aligned with the input string. We use the following notations for an alignment: a lower case letter corresponds to a mutation of a nucleic acid, while '-' indicates the empty letter (and then an insertion or deletion according to whether it belongs to the input or the output string). The real folding is also shown in order to give a reference.

Table 1. Data set used for complexity tests

organism	type	length
C. elegans	U18 snoRNA	65
Arabidopsis	tRNA	74
Homo sapiens	Y RNA	85
E. coli	tRNA	94
X. leavis	U16 snoRNA	103
Homo sapiens	scRNA	121
M. pneumoniae	4.5S RNA homolog	135
Mus musculus	snRNA	150

Obviously, because of the simplicity of our model,

```
0. Compute all the 3-tuple $(\lambda_{\omega'_{ij}}, \delta(\omega_{ij}, \omega'_{ij}), \omega'_{ij})$ and add them to $Best[A, i, j]$
   according to:
1.    If there is a 3-tuple $(\lambda', \delta', \omega'_{ij})$ in $Best[A, i, j]$ Then
           replace it by $(\lambda, \delta, \omega'_{ij})$ if and only if $\delta < \delta'$.
2.    If $Best[A, i, j]$ doesn't contain any 3-tuple $(\lambda', \delta', \omega'_{ij})$ Then add $(\lambda, \delta, \omega'_{ij})$.
```

Fig. 5. Multi-criteria optimization: improvment

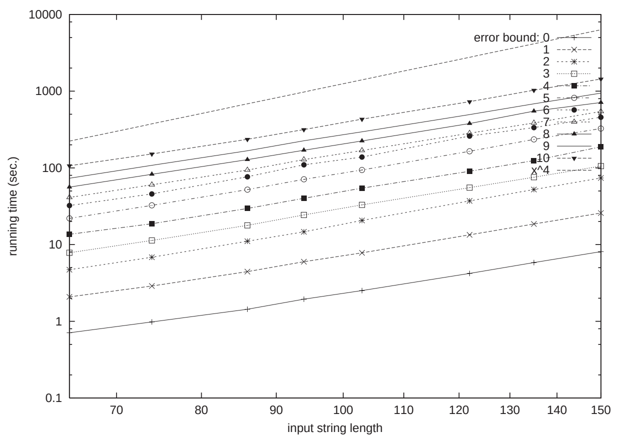


Fig. 6. Running time versus input length with fixed error bound (curve with dotted lines is parallel to x^4).

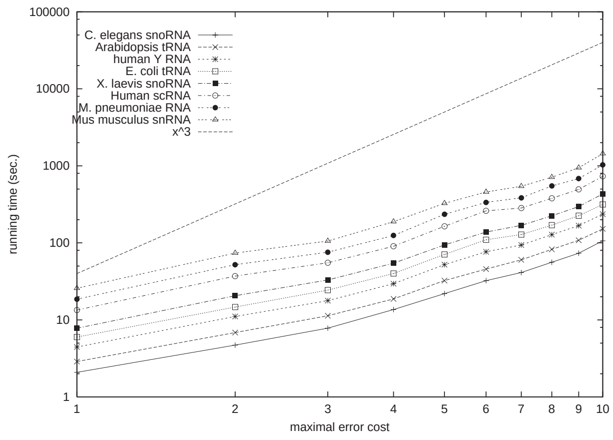


Fig. 7. Running time versus maximal error cost with fixed length(curve with dotted lines is parallel to x^3).

when the maximal error is 0 the real folding is not necessarily found. However, when we increase the bound, a stable folding center on the corresponding anticodon appears (TGC for Figure 8 and GGC for Figure 9). This phenomenon occurs on more than 80% of the data set. The two t-RNA sequences shown here are associated with the same amino acid (Alamine). We note that even if the predicted structure are different when the error bound is 0, a common folding appears when the error bound is increased.

DISCUSSION

The application domain of AMSAG could be large. As a prediction tool, we can use it to detect an optimal folding which could lead the real one. Effectively, during the folding process an attractive configuration could have a major influence on the final structure. In the same time, finding sub-optimal foldings could allow us to evaluate the stability of a configuration. For example, if the optimal structure is noticeably different from the sub-optimal ones, it seems reasonable to think that it does not correspond to the real folding. Finally, AMSAG could be an useful tool for classifying biological sequences. The detection of similar derivation trees on different sequences could be an important criterion to sort the data. When the deletion cost is linear, the complexity of our algorithm is largely improved by adding a preprocessing stage which provides information about the occurences of the terminals in the input sting. The improved algorithm is $\mathcal{O}(n^{r'})$ where r' is the maximum number of the non-terminals in the right-hand side of any production rule in the grammmar. In our simple model of RNA secondary structures we reduce the complexity by a factor of n by this improvement.

CONCLUSION

As was shown in Lefebvre (1995, 1996), S-attributed grammars provide a powerful and an adaptable formalism for sequence analysis. The work presented here in this paper naturally extends the application domain of this theory. The development of efficient algorithms dedicated to S-attributed grammars gives us useful tools for understanding and classifying biological data. We have started the implementation of these algorithms in order to use them as a prediction tool. Other improvements are also necessary for studying large sequences: we have developed, with M.Nguyen-The, a statistical model for energy which allows to restrict to foldings of more than

```
real folding:
(((((((--((((--------))))-(((((-------)))))-----(((((-------))))))))))))----
GGGGCTATAGCTCAGCTGGGAGAGCGCCTGCTTTGCACGCAGGAGGTCTGCGGTTCGATCCCGCATAGCTCCACCA

AMSAG with an error bound of 0:
--(((----((((--------)))))))(((---)))----((((--((((((-------))))--))))))----
GGGGCTATAGCTCAGCTGGGAGAGCGCCTGCTTTGCACGCAGGAGGTCTGCGGTTCGATCCCGCATAGCTCCACCA
GGGGCTATAGCTCAGCTGGGAGAGCGCCTGCTTTGCACGCAGGAGGTCTGCGGTTCGATCCCGCATAGCTCCACCA

energy: -8.96, bound: 0

AMSAG with an error bound of 5:
(((((((((((------((((((((((((((((---))))))))))))----------)))))))))))))))----
GGGGCTATAGCTCAGCTGGGAGAGCGCCTGCGTTGCACGCAGGCGCTCTGCGGTTCGATCCCGCTATAGCCCCACCA
GGGGCTATAGCTCAGCTGGGAGAGCGCCTGCtTTGCACGCAGGaGgTCTGCGGTTCGATCCCGC-ATAGCtCCACCA

energy: -30.84, bound: 5
```

Fig. 8. Results for Ecoli-K12.trna2-AlaTGC

```
real folding:
GGGGCTATAGCTCAGCTGGGAGAGCGCTTGCATGGCATGCAAGAGGTCAGCGGTTCGATCCCGCTTAGCTCCACCA
(((((((--((((--------))))-(((((-------)))))-----(((((-------))))))))))))----

AMSAG with an error bound of 0:
---------((((--------))))-(((((((---)))))))-----(((((-------)))))-----------
GGGGCTATAGCTCAGCTGGGAGAGCGCTTGCATGGCATGCAAGAGGTCAGCGGTTCGATCCCGCTTAGCTCCACCA
GGGGCTATAGCTCAGCTGGGAGAGCGCTTGCATGGCATGCAAGAGGTCAGCGGTTCGATCCCGCTTAGCTCCACCA

energy: -10.52, bound: 0

AMSAG with an error bound of 5:
(((((----(((((((((((((((((((((((((---))))))))))))----------))))))))))))))))))-
GGTGGCTATAGCTCAGCTGGGAGAGCGCTTGCATGGCATGCAAGCGCTCAGCGGTTCGATCCCAGCTGAGCTCCACCA
GG-GGCTATAGCTCAGCTGGGAGAGCGCTTGCATGGCATGCAAGaGgTCAGCGGTTCGATCCC-GCTtAGCTCCACCA

energy: -34.36, bound: 5
```

Fig. 9. Results for Ecoli-K12.trna70-AlaGGC

the average energy; this model is currently being checked.

REFERENCES

Aho,A. and Peterson,T. (1972) A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.*, **4**, 305–312.

Aho,A., Sethi,R. and Ullman,J. (1988) *Compilers. Principles, Techniques, and Tools*. Addison Wesley.

El-Mabrouk,N. and Raffinot,M. (2002) Approximate matching of secondary structures. In *Proceedings of the 6th Annual International Conference on Computational Molecular Biology*.

Gaspin,C., Bourret,P. and Westhof,E. (1995) Détermination assistée par ordinateur de structures secondaires d'arn. *Technique et Science Informatique*, **14**, 141–160.

Knuth,D. (1968) Semantic of context-free languages. *Mathematical Systems Theory*, **2**, 127–145. Correction: Mathematical Systems theory (1971) **5**, 95–96.

Lefebvre,F. (1995) An optimized parsing algorithm well-suited to rna folding. In Press,A. (ed.), *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*. pp. 222–230.

Lefebvre,F. (1996) A grammar-based unification of several alignment and folding algorithms. In Press,A. (ed.), *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*. pp. 143–154.

Lowe,T. and Eddy,S. (1997) trnascan-se: a program for improved detection of transfer rna genes in genomic sequence. *Nucleic Acids Res.*, **25**, 955–964.

Lyon,G. (1974) Syntax-directed least-errors analysis for context-free languages: a pratical approach. *Comm. of the ACM*, **17**, 5–37.

Myers,E. and Miller,W. (1989) Approximate matching of regular expressions. *Bull. Math. Biol.*, **51**, 5–37.

Myers,E.W. (1995) Approximately matching context-free languages. In Baeza-Yates,R. and Manber,U. (eds), *Proceedings of the 2nd South American Workshop on String Processing*. Valparaíso, Chile, pp. 38–52.

Sankoff,D. (1985) Silmutaneous solution of the rna folding, aligment and protosequence problems. *SIAM J. Appl. Math.*, **45**, 810–825.

Searls,D.B. (1992) The linguistics of dna. *American Scientists*, **80**, 579–591.

Vauchaussade de Chaumont,M. (1985) *Nombre de Strahler des arbres, langages algébriques et dénombrement de structures secondaires en biologie moléculaire*, Master's thesis, Université de Bordeaux I.

Waterman,M. (1984) General methods of sequence comparison. *Bull. Math. Biol.*, **46**, 473–501.

Younger,D. (1967) Recognition and parsing of context-free language in time n^3. *Information and Control*, **20**, 189–208.

Zuker,M. (1989a) On finding all suboptimal foldings of an rna molecule. *Science*, **244**, 48–52.

Zuker,M. (1989b) The use of dynamic programming algorithms in rna secondary structure prediction. In Waterman,M. (ed.), *Mathematical Methods for DNA Sequences*, Chapter 7, CRC press, pp. 159–184.

Zuker,M. and Sankoff,D. (1984) Rna secondary structures and their prediction. *Bull. Math. Biol.*, **46**, 591–621.