An optimized parsing algorithm well suited to RNA folding

Fabrice Lefebvre*

LIX
École Polytechnique
91128 Palaiseau CEDEX
lefebvre@lix.polytechnique.fr

Abstract

The application of stochastic context-free grammars to the determination of RNA foldings allows a simple description of the sub-class of sought secondary structures, but it needs efficient parsing algorithms. The more classic thermodynamic model of folding, popularized by Zuker under the framework of dynamic programming algorithms, allows an easy computation of foldings but its use is delicate when constraints have to be introduced on sought secondary structures. We show here that S-attribute grammars unify these two models and we introduce a parsing algorithm whose efficiency enables us to handle problems until then too difficult or too large to deal with. As a matter of fact, our algorithm is as efficient as a standard dynamic programming one when applied to the thermodynamic model (yet it offers a greater flexibility for the expression of constraints) and it is faster and saves more space than other parsing algorithms used so far for stochastic grammars.

Introduction

In RNA, interactions between nucleotides form base pairs and, seen at a higher level, characteristic secondary structure motifs such as helices, loops and buldges. When multiple RNA sequences must be aligned, both primary structure and secondary structure need to be considered since elucidation of common folding patterns may indicate some pertinent regions to be aligned and vice versa (Sankoff 1985). Several methods have been established for predicting RNA secondary structure. The first method is phylogenetic analysis of homologous RNA molecules. It relies on conservation of structural features during evolution. The second method uses a thermodynamic model of RNA secondary structure to find the structure with the lowest free energy. The underlying thermodynamic model is simple enough to allow implementation of the search in the form of a dynamic programming algorithm (Zuker 1989) whose complexity is $O(n^2)$ in space and $O(n^3)$ if time, where n is the number of bases of the RNA. This method, compared to some others, is fast and does not use too much memory (i.e. the constant in front of both complexities is small). Unfortunately, it has the major drawback of having to rely on optimized thermodynamic models to correctly fold RNAs whose secondary structure is known (Papanicolaou, Gouy, & Ninio 1984), and of allowing the introduction of structural constraints only by means of modified energy models. These hand-crafted optimizations and modifications of thermodynamic energies do not contribute to an increased confidence in returned results. This method is however the only resort when nothing but a single member of a RNA family is available, making phylogenetic approaches impossible. A third method has been recently introduced by Sakakibara *et al.* (Sakakibara *et al.* 1993) for the problems of folding, aligning and modeling families of homologous RNA sequences. It relies on *stochastic context-free grammars* (or stochastic grammars) to model common secondary structures of a given family of RNAs. It is well known that parse trees of a RNA, for an appropriate context-free grammar, can describe secondary structures without pseudo-knots of this RNA (Searls 1992). A stochastic grammar is a context-free grammar whose productions have been assigned probabilities. A stochastic grammar assigns therefore a probability to each parse tree and thus to each secondary structure. The sought secondary structure is obtained from the parse tree having the highest probability. A multiple alignment of RNAs belonging to the family modeled by the grammar might be obtained by identification of nonterminals appearing in the most probable parse trees of those RNAs. The parsing algorithm currently used by Sakakibara *et al.* is derived from the algorithm of Cocke–Younger–Kasami, and though its space complexity may still be $O(n^2)$, it is very memory hungry and the constant in front of the n^2 main order term is very sensitive to the size of the grammar. Even small problems (involving RNAs of 150 bases) have sometimes to be scaled down in order to become manageable on a 128 Mbytes computer (Underwood 1994).

*This work was partially supported by a grant of the GREG project Recherche de Motifs dans les Séquences.

In this paper, we shall show that the classic thermodynamic model can be described in terms of S-attribute grammars. With the help of the context-free formalism underlying these grammars, we shall gain on a more traditional dynamic programming approach the ability to express easily structural constraints (our constraints are slightly different from constraints discussed in (Gaspin, Bourret, & Westhof 1995), where they are used under the framework of *Constraint Satisfaction Problems*). Since it is possible to think of probabilities used by stochastic grammars in terms of attributes, stochastic grammars will also be readily converted into S-attribute grammars, thus unifying the thermodynamic model and the probabilistic model. But stating that an existing model (the thermodynamic model) might be expressed in some new way is useless if this new way turns out to be unmanageable, or at least much less efficient than existing methods, when dealing with usual problems. Therefore we also introduce a parsing algorithm for S-attribute grammars, which is in fact faster than a standard dynamic programming algorithm when applied to S-attribute grammars derived from thermodynamic folding rules. An added benefit of this parsing algorithm is that, when applied to S-attribute grammars directly derived from stochastic grammars used by Sakakibara et al., it consumes much less memory and is significantly faster than their parsing algorithm.

In the next section, we shall define context-free and S-attribute grammars. Then a parsing algorithm for context-free grammars will be introduced in the third section. This algorithm will be the foundation of the parsing algorithm introduced in the fifth section for S-attribute grammars. We discuss some implementation details in the fourth section and we give a S-attribute grammar for the thermodynamic model in the sixth section, together with an example of structural constraint. Some benchmarks and simulation results are provided in the last section.

Context-free and S-attribute grammars

We shall use the following classic definition of context-free grammars.

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

The relation $\rightarrow$, called a **derivation relation**, has a reflexive transitive closure denoted by $\rightarrow^*$. The replacement of a nonterminal A in a string of V^* by a string α such that $A \rightarrow \alpha$ is a **derivation**. A **derivation tree** is a planar representation of a sequence of derivations. The set of strings in T^* derived from S is the language generated by G, denoted by $L(G)$. An empty string is denoted by ϵ. In order to keep the number of derivation trees finite, we shall assume that the grammar is non-circular, that means that no nonterminal A may verify $A \rightarrow^+ A$. We shall also assume that the grammar is epsilon-free (i.e. it has no rule of the form $A \rightarrow \epsilon$). An **ambiguous** grammar is a grammar for which there exists a string of symbols having at least two different derivation trees. Grammars whose derivation trees describe secondary structures will always be ambiguous because a given RNA has potentially several different secondary structures. We shall call parsing the process of finding a derivation tree, or **parse tree** of a string in $L(G)$.

Context-free grammars allow us to give a synthetic description of a set of secondary structures, but they do not allow us to choose one structure among all those different anonymous possible structures. S-attribute context-free grammars (which are a proper subset of attribute-grammars introduced by Knuth in his seminal paper (Knuth 1968)) are an extension of context-free grammars allowing the assignment of a value (called attribute) to every vertex of a derivation tree. With attributes, we may now select derivation trees with a simple criterion. 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 and we will give in the sixth section an S-attribute grammar having more complex attributes. Our context of utilization of S-attribute grammars has led us to the following definition for those grammars.

Definition *An S-attribute grammar is denoted by $G = (T, N, P, S, \mathcal{A}, S_{\mathcal{A}}, F_P)$. This is an extension of the context-free grammar $G = (T, N, P, S)$, where an attribute $x \in \mathcal{A}$ is attached to each symbol $X \in V$, and a string of attributes $\lambda \in \mathcal{A}^*$ to each string $\alpha \in V^*$. $S_{\mathcal{A}}$ is a function from T to $\mathcal{A}$ assigning attributes to terminals. F_P is a set of functions from $\mathcal{A}^*$ to $\mathcal{A}$. A function $f_{A \rightarrow \alpha}$ is in F_P iff $A \rightarrow \alpha$ is in P.*

The attribute λ of a string 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 of a string α derived from A, it returns the attribute x of A. Thus, functions of F_P are responsible for the computation, in a bottom-up way, of the attributes of nonterminals A in derivations $\alpha A \beta \rightarrow^* u$, where u must belong to T^* in order that the attribute of A may be computable.

We shall use thereafter the following classic symbols and conventions to avoid the use of membership qualifiers: A, B, C, D for elements of N; X, Y, Z for elements of V; α, β, γ for elements of V^*; a, b, c for elements of T; u, v, w for elements of T^*; x, y, z for elements of $\mathcal{A}$; λ, μ, ν for elements of $\mathcal{A}^*$.

Syntax analysis of context-free grammars

Usually, a parsing algorithm (or parser) is considered efficient when its time and space complexities are linear functions of input length. In order to achieve this efficiency, parsers such as those generated by YACC only consider a single derivation (and thus a single interpretation) of any parsed string. When the grammar describing the language is ambiguous, those parsers rely on default actions (shift rather than reduce, associativity and priority of operators, ...) to resolve ambiguities. In any case, we can assert that a parsing algorithm useful to the search of secondary structures will need to explore these ambiguities, rather than avoid them, in order to find a secondary structure fulfilling some criterion (minimal energy, maximal probability, etc...). The algorithm at the heart of our parser will thus have to be sufficiently general to deal with almost every context-free grammar. The algorithm we are about to describe belongs to the family of *tabular parsers*. The two best known algorithms stemming from this family are the, not so efficient, algorithm of Cocke–Younger–Kasami (Aho & Ullman 1972) and Earley's algorithm (Aho & Ullman 1972; Earley 1970). While being related to this last one, our algorithm will constitute a more effective basis on which will be naturally grafted a strategy (introduced in the next section) of evaluation and selection of attributed trees. The syntax analysis method behind our algorithm will be called Generalized Common Prefix (GCP) parsing (Nederhof 1993; Voisin 1988). Some concepts used in our presentation of the algorithm will be found elsewhere in a somewhat different form (Kruseman Aretz 1989; Graham, Harrison, & Ruzzo 1980).

Definition *Let $G = (T, N, P, S)$ be a non-circular epsilon-free context-free grammar. We define the set E_{GCP} of items as follows:*

$$E_{GCP} = \left\{ [\{A_1, \ldots, A_n\} \rightarrowtail \alpha] \;\middle|\; n > 0 \;\wedge\; \forall A_i, \exists \beta, A_i \rightarrow \alpha\beta \right\}$$

We shall use the symbol Δ (and its derivations) to range over sets of nonterminals. An item belonging to E_{GCP} can therefore be written $[\Delta \rightarrowtail \alpha]$. We shall also call item the association of an item $[\Delta \rightarrowtail \alpha]$ with an integer i, and we shall denote by $[\Delta \rightarrowtail \alpha, i]$ this association.

We have imposed on the grammar G to be non-circular and epsilon-free. These conditions are not essential for the definition of E_{GCP}, but they are essential for the tabular parser which will be introduced later.

Definition *The parse table of a string $a_1 \ldots a_n$ is a vector of entries $(T_j)_{1 \leqslant j \leqslant n}$. Each entry T_j of a parse table is a set of items $[\Delta \rightarrowtail \alpha, i]$ so constructed as to satisfy the following minimum condition of usefulness:*

$$\forall [\Delta \rightarrowtail \alpha, i] \in T_j, \forall A \in \Delta, \exists \beta, A \rightarrow \alpha\beta \rightarrow^* a_{i+1} \ldots a_j \beta$$

The string $a_1 \ldots a_n$ belongs to $L(G)$ if $S \rightarrow^* a_1 \ldots a_n$. We shall show later on that $a_1 \ldots a_n \in L(G)$ if and only if there exists $[\Delta \rightarrowtail \alpha, 0] \in T_n$ such that $S \in \Delta \;\wedge\; S \rightarrow \alpha$.

We now introduce a relation $\angle$ which will be helpful to determine which symbols are expected at the beginning of derivations.

Definition *The relation $\angle$ on $V \times N$ is defined by:*

$$X \angle A \iff \exists \alpha, A \rightarrow X\alpha$$

The reflexive transitive closure of $\angle$ is denoted by $\angle^*$. Notice that for every X, $X \angle^* X$.

The partial order $<$ on N is defined by:

$$A < B \iff B \rightarrow A$$

Since G is not circular, the partial order $<$ might be extended into a total order by means of a topological sort of nonterminals.

Definition *The item resulting from the extension of an other item by a symbol is given by the $\mathcal{G}oto$ function, classic in parsing theory:*

$$\mathcal{G}oto([\Delta \rightarrowtail \alpha], X) = \begin{cases} [\Delta' \rightarrowtail \alpha X] & \text{if } \Delta' \neq \emptyset, \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

where $\Delta' = \{A \in \Delta \mid \exists \beta, A \rightarrow \alpha X \beta\}$.

For an item $[\Delta \rightarrowtail \alpha, i]$ in T_j to be extended in an item $[\Delta' \rightarrowtail \alpha X, i]$ in $T_{k>j}$, the relation $[\Delta' \rightarrowtail \alpha X] = \mathcal{G}oto([\Delta \rightarrowtail \alpha])$ must hold. if X is a terminal a, then $k = j + 1$ and a must be a_{j+1}. If X is a nonterminal A, this nonterminal must be a reducible nonterminal of an item $[\Delta'' \rightarrowtail \gamma, j]$ of T_k, that is, $A \in \Delta''$ and $A \rightarrow \gamma$. We shall denote by $\mathcal{R}educe$ the function giving the set of reducible nonterminals of an item, and by $\mathcal{E}xpect$ the function returning the set of nonterminals one might expect if an item $[\Delta \rightarrowtail \alpha]$ has to be extended.

Definition *The $\mathcal{R}educe$ function of an item $[\Delta \rightarrowtail \alpha]$ is defined by:*

$$\mathcal{R}educe([\Delta \rightarrowtail \alpha]) = \{A \in \Delta \mid A \rightarrow \alpha\}.$$

The $\mathcal{E}xpect$ function of an item $[\Delta \rightarrowtail \alpha]$, or of an entry T_j, is defined by :

$$\mathcal{E}xpect([\Delta \rightarrowtail \alpha]) = \left\{ D \in N \;\middle|\; \exists A \rightarrow \alpha B \beta,\; A \in \Delta \;\wedge\; D \angle^* B \right\}$$

$$\mathcal{E}xpect(T_j) = \bigcup_{[\Delta \rightarrowtail \alpha, i] \in T_j} \mathcal{E}xpect([\Delta \rightarrowtail \alpha])$$

The set $\mathcal{E}xpect(T_j)$ is the set of nonterminals which might extend at least one item of T_j. Therefore, the set Δ of an item $[\Delta \rightarrowtail \alpha, j]$ must be included in $\mathcal{E}xpect(T_j)$. One must not mistake the set of nonterminals returned by $\mathcal{E}xpect$ for the set of terminals returned by the *FOLLOW* function, classic in parsing theory (Aho & Ullman 1972).

Definition *The item of right-hand side X, obtained from a set Δ of expected nonterminals, is given by:*

$$\mathcal{I}nitial(\Delta, X) = \begin{cases} [\Delta' \rightarrowtail X] & \text{if } \Delta' \neq \emptyset, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

where $\Delta' = \{A \in \Delta \mid X \angle A\}$.

An item $[\Delta \rightarrowtail X, i]$ may exist only if it verifies $[\Delta \rightarrowtail X] = \mathcal{I}nitial(\mathcal{E}xpect(T_i), X)$. Thus every set Δ of an item $[\Delta \rightarrowtail \alpha, i]$ satisfies $\Delta \subset \mathcal{E}xpect(T_i)$. We shall denote the set $\mathcal{E}xpect(T_i)$ by Δ_i. Notice that T_0 does not exist and we shall note $\Delta_0 = \{A \in N \mid A \angle^* S\}$ the set of nonterminals expected at the beginning of a derivation of the start symbol S.

Thanks to those previous definitions, we may now informally explain how entries T_j are built from a string $a_1 \ldots a_n$.

- We first start with an empty T_j.
- Our first moves will try to use the next input symbol a_j in every possible way. First, the terminal a_j might start the right-hand side of an item. Thus, if $[\Delta \rightarrowtail a_j] = \mathcal{I}nitial(\Delta_{j-1}, a_j)$ is defined, we add the item $[\Delta \rightarrowtail a_j, j-1]$ to T_j. Then, a_j might extend an item $[\Delta' \rightarrowtail \alpha, h]$ of T_{j-1}. If $[\Delta \rightarrowtail \alpha a_j] = \mathcal{G}oto([\Delta' \rightarrowtail \alpha], a_j)$ is defined, we add the item $[\Delta \rightarrowtail \alpha a_j, h]$ to T_j.
- Among items which have been added to T_j, some might be reducible. Let $[\Delta'' \rightarrowtail \beta, i]$ be such an item and $A \in \mathcal{R}educe([\Delta'' \rightarrowtail \beta])$ be a reduced nonterminal. Notice that $A \rightarrow^* a_{i+1} \ldots a_j$ and that i is the *left position* of A ($j-1$ was the left position of a_j). Our next moves will try to use A in every possible way. First, A might start the right-hand side of an item. Thus, if $[\Delta \rightarrowtail A] = \mathcal{I}nitial(\Delta_i, A)$ is defined, we add the item $[\Delta \rightarrowtail A, i]$ to T_j. Then, A might extend an item $[\Delta' \rightarrowtail \alpha, h]$ of T_i. If $[\Delta \rightarrowtail \alpha A] = \mathcal{G}oto([\Delta' \rightarrowtail \alpha], A)$ is defined, we add the item $[\Delta \rightarrowtail \alpha A, h]$ to T_j.
- Finally, we repeat the previous step while T_j is not stationary.

Therefore, whether X is a terminal a_j or a nonterminal A, the same kind of moves are performed. Only the value of X and its left position care. We may thus handle reducible items as soon as they are added to T_j by maintaining a list Q of pairs (X, i) of reduced symbols with their left position. This list holds initially the pair $(a_j, j-1)$ and, whenever an item $[\Delta \rightarrowtail \alpha, i]$ such that $A \in \mathcal{R}educe([\Delta \rightarrowtail \alpha])$ is added to T_j, we add (A, i) to the list. Then, while Q is not empty, we just have to pick a pair from this list to know what to do. Even if it is not essential here, we shall structure Q into a priority queue for the sake of attributes which will be dealt with later.

Definition *Let Q be a set of pairs $(X, i) \in V \times \{1, \ldots, n\}$. We say that $(X, i) < (Y, j)$ if $i < j$ or if $i = j$ and $X < Y$. Thus the relation $<$ defines a total order over the set of pairs. We shall denote by $\mathcal{E}xtract$ the function used to both return the maximum pair (X, i) of Q and remove this maximum from Q. We shall denote this extraction by $(X, i) = \mathcal{E}xtract(Q)$.*

The $<$ relation on pairs will also be called a priority relation and the maximum pair will be called the pair having the greatest priority. A more formal presentation of our previous discussion leads to the following tabular algorithm.

Algorithm 1 *Let Q be the previously defined set of pairs (X, i) with its $\mathcal{E}xtract$ function. As soon as an item $[\Delta \rightarrowtail \alpha, i]$ is added to T_j (steps 2 and 3 below), the step $Q := Q \cup \{(X, i) \mid X \in \mathcal{R}educe([\Delta \rightarrowtail \alpha])\}$ must be performed.*
At the beginning, all entries T_j are empty. Let $\Delta_0 = \{A \in N \mid A \angle^* S\}$.
For j going from 1 to n, perform the following steps in that order:

- *Initialize the queue: $Q := \{(a_j, j-1)\}$*
- *Perform the following steps in that order, while Q is not empty:*
  1. *Extraction of the pair having the greatest priority: $(X, i) := \mathcal{E}xtract(Q)$*
  2. *Creation of items beginning with X:*

$$T_j := T_j \cup \left\{ [\Delta \rightarrowtail X, i] \;\middle|\; [\Delta \rightarrowtail X] = \mathcal{I}nitial(\Delta_i, X) \right\}$$

  3. *Extension of an item by X:*

$$T_j := T_j \cup \left\{ [\Delta \rightarrowtail \alpha X, h] \;\middle|\; \exists [\Delta' \rightarrowtail \alpha, h] \in T_i, [\Delta \rightarrowtail \alpha X] = \mathcal{G}oto([\Delta' \rightarrowtail \alpha], X) \right\}$$

- *Compute $\Delta_j := \mathcal{E}xpect(T_j)$.*

The string $a_1 \ldots a_n$ belongs to $L(G)$ if and only if there exists an item $[\Delta \rightarrowtail \alpha, 0] \in T_n$ such that $S \in \mathcal{R}educe([\Delta \rightarrowtail \alpha])$. We then say that the string has been recognized by the algorithm.

Algorithm 1 is a **recognition algorithm**: it allows us to decide whether a given string belongs to $L(G)$. It does not directly give a parse tree of a derivation $S \rightarrow^* a_1 \ldots a_n$. However, it is easy to extend this algorithm to yield parse trees since all necessary information is

left in the parse table after recognition. We now show that algorithm 1 correctly recognizes strings of $L(G)$, and only those strings.

Theorem 1 *When entries T_j are constructed as described in algorithm 1, then an item $[\Delta \rightarrowtail \alpha, i]$ belongs to T_j if and only if $\alpha \rightarrow^* a_{i+1} \dots a_j$ and*

$$\forall A \in \Delta, \exists(\beta, \gamma), S \rightarrow^* a_1 \dots a_i A\gamma \rightarrow a_1 \dots a_i \alpha\beta\gamma$$

Proof: the proof of this theorem takes its inspiration from the proof of Earley's algorithm given in (Aho & Ullman 1972). Necessary and sufficient conditions are both proved by induction (Lefebvre 1995). □

We can thus say that $[\Delta \rightarrowtail \alpha, 0] \in T_n$ and $S \in \mathcal{R}educe([\Delta \rightarrowtail \alpha])$ if and only if $S \rightarrow \alpha \rightarrow^* a_1 \dots a_n$, that is, if and only if $a_1 \dots a_n \in L(G)$. Thus algorithm 1 is correct. The preceding theorem also states that an item is never added to T_j if there is no $a_{j+1} \dots a_n$ such that this item might be used in a parse tree. This property is usually called the *correct prefix property*. Moreover, the form of items used by GCP parsing will never lead to the construction of a given subderivation twice. In some sense, our tabular GCP parsing algorithm is an optimal tabular parsing algorithm.

Theorem 2 *The time complexity of algorithm 1 is $O(n^3)$ and its space complexity is $O(n^2)$. If the grammar G is unambiguous, then algorithm 1 has always the exact same time complexity and space complexity. The time complexity is therefore $O(n^2)$ at worst. If G is $LR(k)$ and not right recursive (i.e. there is no A such that $A \rightarrow^+ \alpha A$), then algorithm 1 runs in linear time and space.*

Proof: The grammar G being acyclic, we remark that, for any given j, each call of the function $\mathcal{E}xtract$ will return a pair (X, i) whose priority is strictly less than the one of the pair previously returned. Steps 1 to 3 will therefore be performed at most $O(j)$ times (there is a bounded number of nonterminals and $i < j$). Each execution of steps 1 and 2 can be performed in time $O(1)$. If step 3 is properly written (see the next section), its execution time is essentially a function of the number of items about to be added to T_j (they may not be actually added if they are already present in T_j). Now, there is a finite number of items of the form $[\Delta' \rightarrowtail \alpha]$. The addition of an index $h < i$ to make an item $[\Delta' \rightarrowtail \alpha, h]$ shows that there are at most $O(i)$ items in T_i likely to be extended in $[\Delta \rightarrowtail \alpha X]$. Each step 3 can thus be performed in time $O(i)$ (we notice at the same time that if there are at most $O(i)$ items in T_i, there are at most $O(n^2)$ items in the whole parse table). Since Δ_j might be computed in $O(j)$ steps, algorithm 1 runs in time $O(n^3)$.

In the case of an unambiguous grammar, we can prove that the algorithm does only try once to add a given item. Thus it has exact same time complexity and space complexity. Moreover, if the grammar is $LR(k)$ and not right recursive, then the number of items in each entry T_j is bounded by a constant. Thus the space complexity is linear. □

Implementation of the algorithm

To help speed up things, the actual implementation of our parsing algorithm will not directly deal with a formal representation of items, $\mathcal{G}oto$ function and such, but rather with numbers, tables and indexes in tables. This numbering scheme is classic but it is not so easy to implement for our GCP parsing algorithm. Although, in principle, the numbering of every item in E_{GCP} and every subset of nonterminals seems sufficient to construct our tables, it is not always desirable. Indeed, for some grammars (this is the case of stochastic grammars we shall use) the pattern of productions in P will lead to a combinatorial explosion of the number of items in E_{GCP} and of the number of subsets of nonterminals returned by a naïve $\mathcal{E}xpect$ function. Fortunately, only the subset of items which may really appear during any parse is really useful. We have observed for every grammar we tested (this includes stochastic grammars) that the size of this subset has always remained acceptable, while the size of E_{GCP} has sometimes exceeded our computation and storage capabilities. In its present version, our implementation of algorithm 1 only number the subset of items which appeared during *actual* parses. Thus, tables are dynamically computed rather than statically computed and, in principle, their computation cost must be added to the proper cost of parsing. But these tables are of bounded size and are very quickly computed. The cost of their dynamic construction will thus be regarded as constant and negligible when compared to the parsing cost of a string $a_1 \dots a_n$.

The addition of items is sped up by using a table which will memoize which items have been added to T_j. Thus a quick look in this table is enough to decide whether a new item has to be added or an existing item has to be modified. Items added to T_j are stored in lists which are functions of the nonterminal which may be used to extend items by application of the $\mathcal{G}oto$ function. Thus there are as many lists as there are symbols. Items which may be extended by several symbols are shared by the corresponding lists.

In order to save memory, useless items are freed as soon as possible. An item becomes useless when it can not be extended, or when its only possible extensions have been computed and reduced. This release of useless items is not so important in the case of context-free grammars, but it will become crucial for S-attribute grammars, where a single item may have several hundred or thousand attributes.

Syntax analysis of S-attribute grammars

In the previous section, we only gave the description of a parsing algorithm which parses a string according to a given context-free grammar. This grammar can be transformed in an S-attribute grammar thanks to the addition of a set of attributes $\mathcal{A}$, a $S_{\mathcal{A}}$ function returning attributes of terminals, and a set F_P of attribute evaluation functions $f_{A\to\alpha}$. Attributes can easily be managed by algorithm 1 with the following changes:

- A string of attributes λ is attached to each item $[\Delta \rightarrowtail \alpha, i]$, λ being the string of attributes attached to α. We denote by $[\Delta \rightarrowtail \alpha, i, \lambda]$ such an association, which we shall call *attributed item* or, when no confusion may arise, item;
- Pairs (X, i) added to Q are now triplets (X, i, x), where x is the attribute attached to X;
- Attribute evaluation functions are now taken into account at the time of reduction of attributed items.

However, these changes are way too simple if we keep in mind the highly ambiguous nature of grammars used for the description of RNA secondary structures. Whereas in algorithm 1, the potentially exponential number of derivation trees (Waterman 1978) was hidden by sharing of common subtrees (it's a characteristic of tabular algorithms), this can no longer be the case with attributed subtrees. Indeed, while an entire class of derivations could, without attributes, be coded under the form of a single item $[\Delta \rightarrowtail \alpha, i]$ belonging to an entry T_j, this same class has now to appear under the form of a set of attributed items $[\Delta \rightarrowtail \alpha, i, \lambda_k]$, where λ_k depends on the derivation k considered. The existence of an exponential number of derivation trees might well translate into the creation of an exponential number of items in T_n. Time and space necessary for parsing and attribution would no longer be polynomial but exponential. This is unacceptable, especially if a single attributed item $[\Delta \rightarrowtail \alpha, 0, \lambda]$ of T_n will be finally accepted (this is the case when one seeks a secondary structure of minimal energy or maximal probability). Fortunately, the final selection criterion of a derivation tree, which is no more than a selection criterion on the attribute of the start symbol S, can generally be recursively applied to all nonterminals of the grammar under the form of constraints which will help us to prune the forest of derivation subtrees. For instance, if the grammar was obtained from a stochastic grammar, a tree of maximal probability may only contain subtrees of maximal probability (i.e. whose root, labeled by a nonterminal, has a maximal "probability" attribute). These constraints will be therefore introduced in a natural way at the level of the choice of triplets (A, i, x) used to create new items. We denote by $\{A, i, x\} := C_A(\{(A, i, y)\})$ the replacement of a set $\{(A, i, y)\}$ of triplets having same A and i by a second set $\{(A, i, x)\}$ deduced from the first with the help of a constraint C_A associated with the nonterminal A. This constraint may for instance take only the triplet whose attribute is the highest, or replace them all by a triplet whose attribute is the sum of the attributes of the original triplets (we shall give in the next section an example of S-attribute grammar with associated constraints). We can now write down a parsing algorithm taking into account attributes and constraints.

Algorithm 2 *Let Q be a set of triplets $(X, i, x) \in V \times \{1, \ldots, n\} \times \mathcal{A}$. The priority of (X, i, x) is the same as the priority of (X, i). Let $\mathcal{E}xtract$ be a function extracting all triplets having the same greatest priority. We denote this extraction by $\{(X, i, x)\} = \mathcal{E}xtract(Q)$. As soon as an attributed item $[\Delta \rightarrowtail \alpha, i, \lambda]$ is added to T_j (step 3 below), the step $Q := Q \cup \{(A, i, x) \mid A \in \mathcal{R}educe([\Delta \rightarrowtail \alpha]) \ \wedge \ x = f_{A\to\alpha}(\lambda)\}$ must be performed.*
At the beginning, all entries T_j are empty. Let $\Delta_0 = \{A \in N \mid A \mathrel{\angle^*} S\}$.
For j going from 1 to n, perform the following steps in that order:

- *Initialize the queue : $Q := \{(a_j, j-1, S_{\mathcal{A}}(a_j))\}$*
- *Perform the following steps in that order, while Q is not empty:*
  1. *Extraction of triplets having the greatest priority $\{(X, i, y)\} := \mathcal{E}xtract(Q)$*
  2. *Selection of triplets by application of the constraint C_X: $Q' := C_X(\{(X, i, y)\})$*
  3. *For every triplet (X, i, x) of Q', perform the two following steps:*
     Creation of items beginning with X :

$$T_j := T_j \cup \Big\{[\Delta \rightarrowtail X, i, x] \;\Big|\; [\Delta \rightarrowtail X] = \mathcal{I}nitial(\Delta_i, X)\Big\}$$

     Extension of an item by X:

$$T_j := T_j \cup \Big\{[\Delta \rightarrowtail \alpha X, h, \lambda x] \;\Big|\; \exists [\Delta' \rightarrowtail \alpha, h, \lambda] \in T_i,\ [\Delta \rightarrowtail \alpha X] = \mathcal{G}oto([\Delta' \rightarrowtail \alpha], X)\Big\}$$

- *Compute $\Delta_j := \mathcal{E}xpect(T_j)$.*

The string $a_1 \ldots a_n$ belongs to $L(G)$ if and only if there exists an item $[\Delta \rightarrowtail \alpha, 0, \lambda] \in T_n$ such that $S \in \mathcal{R}educe([\Delta \rightarrowtail \alpha])$. We then say that the string has been recognized by the algorithm and that it has an attribute λ, not necessarily unique.

If G is unambiguous, then algorithm 2 behaves like algorithm 1, for the time complexity as well as the space complexity. If G is ambiguous, suppose that each nonterminal possesses a constraint returning only a single triplet for every set of triplets passed in parameter (this is the case in practice). Let $r \geqslant 1$ be the

maximum number of nonterminals appearing at the right-hand side of a production of G. Then the space complexity is $O(\max(n^2, n^r))$ and the time complexity is of the order of n times the space complexity, that is $O(\max(n^3, n^{r+1}))$. Grammars used in practice usually verify $r = 2$. In particular, the main order of the time complexity of our algorithm ($O(n^3)$) has always been equal to the main order of the time complexity of the dynamic programming algorithm used by Zuker (Zuker 1989) to find a RNA secondary structure of minimal energy.

Example of S-attribute grammar

With the purpose of comparing algorithm 2 to the classic dynamic programming method used to find folded RNAs of minimum free energy, we retrieved the "Vienna package" by ftp at `ftp.itc.univie.ac.at` in `pub/RNA/ViennaRNA-1.03.tar.Z`. This well known package is programmed in C and implements dynamic programming relations given by Zuker in (Zuker 1989). We then made an exact conversion of energy tables and computation rules embedded in the package into an S-attribute grammar, using a description akin to YACC syntax. We reproduced the core of this description below, leaving out some tables and glue code which are not essential to the understanding. The reader who is not familiar with YACC syntax shall only retain the overall structure without focusing on some minor syntax points.

```
; We first define what our attributes are. 'E' is the
; Energy of structures, 'LS', 'RS', 'EDS' are the length
; of the Left Strand, length of the Right Strand and
; Energy of the internal Double Strand in internal loops
%attribute int E
%attribute unsigned short LS
%attribute unsigned short RS
%attribute int EDS

; Here we define terminals and their attributes
%terminal A     { $$.E = 0; }
%terminal C     { $$.E = 0; }
%terminal G     { $$.E = 0; }
%terminal U     { $$.E = 0; }

; Then we define what pairs of terminals will stand for
; '(' and ')' in rules
%pair A U  %pair U A
%pair G C  %pair C G
%pair G U  %pair U G

; This constraint will be used for all nonterminals which
; do not have their own constraint
%default_constraint
{
 /* Here is the minimization constraint */
 if ($$.E < $!.E)
   $!.E = $$.E;
}

; And now rules and their attribute evaluation functions
RNA : RNA DoubleStrand  { $$.E = $1.E + $2.E; }
RNA : DoubleStrand      { $$.E = $1.E; }
RNA : RNA Base          { $$.E = $1.E; }
RNA : Base              { $$.E = 0; }
DoubleStrand : ( SingleStrand )
{
 int bonus = 0;
 if ($2.len == 4)
   bonus = BonusTetraloop ($2.left, $2.right);
 $$.E = bonus + E_Hairspin ($2.len)
              + E_Mismatch ($1.left, $3.left);
}
DoubleStrand : ( InternalLoop )   {$$.E=$2.E;}
DoubleStrand : ( MultipleLoop )   {$$.E=460+$2.E;}
InternalLoop : DoubleStrand
{
 $$.EDS = $1.E;  $$.LS = $$.RS = 0;
 $$.E = $1.E+E_DblStrand ($1.left-1, $1.right+1);
}
InternalLoop : InternalLoop Base
{
 $$.EDS = $1.EDS;  $$.LS = $1.LS;  $$.RS = $1.RS+1;
 if ($1.LS == 0)
   $$.E = $$.EDS + E_Buldge ($$.RS);
 else {
   if ($1.RS == 0)
      $$.EDS += E_Mismatch ($2.right - $$.RS,
                            $1.left + $$.LS);
   $$.E = $$.EDS + E_Internal ($1.LS, $$.RS)
                 + E_Mismatch ($1.left-1, $2.right + 1);
 }
}
InternalLoop : Base InternalLoop
{
 $$.EDS = $2.EDS;  $$.LS = $2.LS+1;  $$.RS = $2.RS;
 if ($2.RS == 0)
   $$.E =  $$.EDS + E_Buldge ($$.LS);
 else {
   if ($2.LS == 0)
      $$.EDS += E_Mismatch ($2.right - $$.RS,
                            $1.left + $$.LS);
   $$.E = $$.EDS + E_Internal ($$.LS, $2.RS)
                 + E_Mismatch ($1.left - 1, $2.right + 1);
 }
}
%constraint InternalLoop
{
 /* Do not retain internal loops longer than 30 bases */
 if ($$.LS + $$.RS > 30)
   DO_NOT_SELECT;
 /* We ensure that all necessary attribuets are conserved
    when minimizing internal loops energies */
 if ($$.E < $!.E) {
   $!.E=$$.E;     $!.EDS=$$.EDS;
   $!.LS=$$.LS;   $!.RS=$$.RS;
 }
}
MultipleLoop : MultipleLoop DoubleStrand
{ $$.E = 10 + $1.E + $2.E; }
MultipleLoop : DoubleStrand       {$$.E=10+$1.E}
MultipleLoop : MultipleLoop Base  {$$.E=40+$1.E}
MultipleLoop : Base               {$$.E=40;}
Base : A, Base : U, Base : C, Base : G          { }
%constraint Base                                { }
```

```
SingleStrand : SingleStrand Base                    { }
SingleStrand : Base                                 { }
%constraint SingleStrand
{ if ($$.len > 30)  DO_NOT_SELECT; }
```

This description is quite short and clearly shows what the important structural features are when computing a secondary structure of minimal energy. We can assert that our description is of a more declarative nature than the one found in the Vienna Package, where the procedural nature of the C language spread out information in numerous lines, tests and loops. We shall see in the next section that the declarative power of S-attribute grammars is not detrimental to speed, thanks to algorithm 2.

An added benefit of the language formalism is that we can now easily add structural constraints without artificially adjusting energies. If we want to fold a RNA into a cloverleaf-like structure (useful for tRNAs), we just have to replace the following rules:

```
RNA : RNA DoubleStrand   { $$.E = $1.E + $2.E; }
RNA : DoubleStrand       { $$.E = $1.E; }
RNA : RNA Base           { $$.E = $1.E; }
RNA : Base               { $$.E = 0; }
DoubleStrand : ( MultipleLoop )  {$$.E=460+$2.E;}
MultipleLoop : MultipleLoop DoubleStrand
{ $$.E = 10 + $1.E + $2.E; }
MultipleLoop : DoubleStrand          {$$.E=10+$1.E}
MultipleLoop : MultipleLoop Base     {$$.E=40+$1.E}
MultipleLoop : Base                  {$$.E=40;}
```

by new rules which will require the structure to have one multiple loop with the usual three branching stems. We may use for instance the following rules:

```
tRNA : Acceptor SingleStrand      { $$.E = $1.E}
Acceptor : ( Acceptor )
{ $$.E = $2.E + E_DblStrand ($1.left, $3.left); }
Acceptor : ( SingleStrand D SingleStrand
             Anticodon VariableLoop T )
{ $$.E=460+($2.len+$4.len+$6.len)*40
        + $3.E+10 + $5.E+10 + $7.E+10; }
D : DoubleStrand                  { $$.E = $1.E; }
Anticodon : DoubleStrand          { $$.E = $1.E; }
T : DoubleStrand                  { $$.E = $1.E; }
VariableLoop : SingleStrand       {}
```

With this modified grammar, any RNA given in input will fold into a cloverleaf structure. In our opinion, this example will positively show that there are better ways than arbitrarily fine tuning elementary free energies, in order to maximize the proportion of correctly folded tRNAs or rRNAs. We might even envision a tool which would take in input a meta-grammar describing major structural features (forced base pairs, unpaired bases, size of loops, number of stems in some multiple loops, etc...) and basic energy rules and which would automatically output the required S-attribute grammar.

Results

The S-attribute grammar given in the previous section was automatically turned into a parser by a YACC-like tool which we designed for this purpose. Algorithm 2 is the heart of the generated parser, which takes in input RNA strings and which returns the sought parse tree (here, the secondary structure of minimal energy). With the same input, our parser and the Vienna package will both return the same secondary structure. Thus we may now compare the speed of the two programs. Our comparisons were done on a Decserver 2100-500MP and are gathered in the following table:

	Vienna package	Algorithm 2
683 bases RNA		
time in seconds:	35	21
space in Mbytes:	4.5	13.5
994 bases RNA		
time in seconds:	92	62
space in Mbytes:	7.5	23
1667 bases RNA		
time in seconds:	350	292
space in Mbytes:	19	64

Thus our parser is slightly faster than a standard dynamic programming one (and it takes three times more memory), and energy rules are much easier to specify using a single short S-attribute grammar than using hundreds of lines of not so easy to write and debug C code.

The conversion of stochastic grammars into S-attribute grammars being straightforward, we applied our algorithm to some instances of practical problems involving those grammars. When we took a grammar modeling the tRNA family (Sakakibara *et al.* 1994), we found that our algorithm was at least three times faster than the algorithm used by UCSD's team. We expect our algorithm to be at least 5 times faster and to take approximately 10 times less memory on larger problems such as snRNAs of Rebecca Underwood (Underwood 1994). We did not have a chance to see the actual implementation of their algorithm, but we believe that our algorithm is faster and uses much less memory for the following reasons:

- The probabilistic model of folding of Sakakibara et al. requires a large number of nonterminals and productions in grammars (96 nonterminals and 660 productions for a model of tRNAs, about 500 nonterminals and 2,000 productions for a model of snRNAs). The CYK algorithm is very sensitive to huge grammars since it considers every nonterminal and every production at every step, regardless of what has already been parsed, whereas our algorithm will only reduce nonterminals which are coherent with the already parsed string, and will only try to add items which might be needed to pursue the parse. Moreover, our left-factorization of items (we add at most one item $[\Delta \mapsto \alpha]$ for every possible α) heavily contributes to the good behavior of our algorithm.

- The original CYK algorithm needs grammars in Chomsky normal form (right hand sides of rules might only have either one terminal, or one nonterminal, or two nonterminals). Some of the modifications required by the algorithm to handle stochastic grammars (which are not in Chomsky normal form) do not improve efficiency, to say the least. Our algorithm does not require grammars to be in Chomsky normal form.
- Our left to right parse of the input string makes it possible to gradually free items which become useless during the course of parsing.

In this paper, we described a well-known formalism (S-attribute grammars) and an effective parsing algorithm for it. This formalism and this algorithm might be used in many contexts (natural language parsing, programming languages parsing, etc...), and we found out that the search of RNA secondary structures might be one of those contexts. In fact, the use of S-attribute grammars to express the classic thermodynamic model leads to an algorithm which is faster than the standard dynamic programming one, in a framework that allows much more flexible domain modeling. We also noticed that our algorithm was faster and much less memory hungry than algorithms currently used to parse stochastic grammars. The current trend in the domain of RNA folding seems to indicate that automated tools to build grammars from families of RNAs are much needed. In the near future, we believe that a closer unification of thermodynamic and probabilistic models, together with results like those of Eddy and Durbin (Eddy & Durbin 1994), will make things easier.

Acknowledgments

I wish to thank D. Gardy and J.-M. Steyaert for suggestions and comments on the subject.

References

Aho, A. V., and Ullman, J. D. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall, Inc.

Earley, J. 1970. An efficient context-free parsing algorithm. *Communication of the ACM* 13(2):94–102.

Eddy, S. R., and Durbin, R. 1994. RNA sequence analysis using covariance models. *Nucleic Acids Research* 22(11):2079–2088.

Gaspin, C.; Bourret, P.; and Westhof, E. 1995. Computer assisted determination of secondary structures of RNA. *Techniques et science informatiques* 14(2):141–160.

Graham, S.; Harrison, M.; and Ruzzo, W. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems* 2(3):415–462.

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

Kruseman Aretz, F. E. J. 1989. A new approach to Earley's parsing algorithm. *Science of Computer Programming* 12:105–121.

Lefebvre, F. 1995. s-attribute grammars and RNA folding. Technical report, LIX, École Polytechnique, 91128 Palaiseau Cedex, FRANCE.

Nederhof, M.-J. 1993. A multidisciplinary view on PLR parsing. In *Twentieth Workshop on Language Technology 6.*

Papanicolaou, C.; Gouy, M.; and Ninio, J. 1984. An energy model that predicts the correct folding of both the tRNA and the 5S RNA molecules. *Nucleic Acids Research* 12:31.

Sakakibara, Y.; Brown, M.; Mian, I. S.; Sjölander, K.; Underwood, R. C.; and Haussler, D. 1993. The application of stochastic context-free grammars to folding, aligning and modeling homologous RNA sequences. Technical Report UCSC-CRL-94-14, University of California, Santa Cruz, Santa Cruz CA 95064 USA.

Sakakibara, Y.; Brown, M.; Hughey, R.; Mian, I. S.; Sjölander, K.; Underwood, R. C.; and Haussler, D. 1994. Stochastic context-free grammars for tRNA modeling. available by anonymous ftp at `ftp.cse.ucsc.edu` in `/pub/rna`.

Sankoff, D. 1985. Simultaneous solution of the rna folding, alignment and protosequences problems. *SIAM Journal of Applied Mathematics* 45:810–825.

Searls, D. B. 1992. The linguistics of DNA. *American Scientist* 80:579–591.

Underwood, R. C. 1994. Stochastic context-free grammars for modeling three spliceosomal small nuclear ribonucleic acids. Technical Report UCSC-CRL-94-23, PhD thesis, University of California, Santa Cruz, Santa Cruz CA 95064 USA.

Voisin, F. 1988. A botttom-up adaptation of Earley's parsing algorithm. In *Programming Languages Implementation and Logic Programming, International Workshop*, volume 348 of *Lecture Notes in Computer Science*. Springer-Verlag. 146–160.

Waterman, M. S. 1978. Secondary structure of single-stranded nucleic acids. *Studies in Foundations and Combinatorics, Advances in Mathematics Supplementary Studies* 1:167.

Zuker, M. 1989. The use of dynamic programming algorithms in RNA secondary structure prediction. In Waterman, M. S., ed., *Mathematical Methods for DNA Sequences*. CRC Press. chapter 7, 159–184.