A grammar-based unification of several alignment and folding algorithms

Fabrice Lefebvre*

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

Abstract

We show in this paper that many popular models of folding and/or alignment may be described by a new formalism: multi-tape S-attribute grammars (MTSAG's). This formalism relieves the designer of biological models of implementation details which may hinder his inventiveness. To complete this formalism, we designed and implemented a tool which, given a MTSAG, will output an efficient parser for this grammar. We also show that MTSAG's offer a new, efficient and useful way to handle stochastic context-free grammars and tools that generate them.

Introduction

The multiple alignment of biological sequences (DNAs, RNAs, proteins) is one of today's essential tools in molecular biology. Alignments are used in database searching or when trying to find subsequences which carry a significant portion of the biological function of homologous sequences.

The multiple alignment of DNAs or proteins is often carried out by some variations of affine gap penalty models (Myers 1991; Thompson, Higgins, & Gibson 1994) or, more recently, by HMM's (Krogh *et al.* 1994; Baldi *et al.* 1994). Since every HMM is tailored to a particular family of homologous sequences, HMM's have the distinctive advantage of producing good multiple alignments without resort to expensive computational methods. There is an active field of research in connection with the training of HMM's.

The situation is not so bright for RNAs. In fact RNA sequences that share a common function and structure can appear to be unalignable until the common structure is recognized because base pairs introduce long range dependencies which are not captured by the standard methods of alignment. In the past, there were some attempts to align RNAs through the alignment of their common structural features (Hoffmann & O'Donnel 1982; Shapiro 1988; Margalit *et al.* 1989; Shapiro & Zhang 1990; Chevalet & Michot 1992; Jiang, Wang, & Zhang 1994) on the assumption that existing folding algorithms were returning essentially accurate structures (Zuker 1989; Le *et al.* 1989). Unfortunately, this assumption is unrealistic. Moreover, most algorithms which align structures are based on variations of the concept of tree-edit, and are not well suited to align tens of sequences. On the other hand, when a good multiple alignment of RNAs is available, there exists reliable methods to find an embedded common structure (Han & Kim 1993; Grate 1995; Cary & Stormo 1995). There remains the problem of obtaining this good multiple alignment. It is now widely admitted that no good multiple alignment of homologous RNAs may be obtained without a good model of their common structural features, and conversely, no good model of their common structural features may be obtained without a good multiple alignment. Thus the RNA folding and alignment problem looks like the chicken and egg problem. Nevertheless, some methods exist to simultaneously fold and align RNAs (Sankoff 1985; Bafna, Muthukrishnan, & Ravi 1995) but they have prohibitive costs in time and, which is worse, in space.

In this context, the Covariance Models (CM's) (Eddy & Durbin 1994) and the more general Stochastic Context-Free Grammars (SCFG's) (Sakakibara *et al.* 1994) look very promising because, once constructed for a given family of sequences, they allow one to discriminate, fold and align new sequences belonging to this family. While an actual procedure has been given to automatically construct a CM from a set of unaligned, unfolded, homologous sequences, no such procedure exists as yet for SCFG's (to the best of my knowledge).

We shall see in this paper that most popular models (including HMM's, SCFG's and CM's) of alignment and/or folding of DNAs, RNAs or proteins share a common representation in terms of a new formalism:

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

Multi-Tape S-Attribute Grammars (MTSAG's). This new formalism offers a new framework to biologists, clearer and more elegant than the never-ending dynamic programming relations with which traditional methods are usually expressed. But this formalism is not only of use for the description of old or new methods. We designed and implemented a tool which, given a MTSAG, will automatically generate the C source of an efficient C parser which is able to compute alignments and foldings. The speed and memory requirements of such generated parsers stand the comparison with programs manually written from dynamic programming relations.

We shall also describe in this paper two different ways to rewrite a SCFG into a MTSAG. The first way involves only a trivial rewriting which has the sole consequence of allowing the use of our automatic parser generator. The second way harnesses the full power of MTSAG's and should foster new developments. For instance, we shall describe here a method which automatically build SCFG's from sets of unaligned, unfolded RNAs.

We shall first give some definitions and notations before introducing the parsing algorithm for MTSAG's and we shall then give some applications of MTSAG's to edit-distance based alignment methods and to SCFG's. Finally, we shall give a few highlights of our results on some other methods.

Definitions

In this section, our main concern will be to define a special "m-tape" alphabet which will handle sequence alignments, and then a "m-tape" extension of context-free grammars which will handle structures of alignments.

Definition (m-tape index). *A m-tape index $\vec{i}$ is a vector of $\mathbb{Z}^m$. The notation $\vec{i}^{(k)}$ will denote the k-th component of a m-tape index $\vec{i}$ and it will stand for the value of $\vec{i}$ on the k-th tape. We shall say that a m-tape index $\vec{i}$ is inferior to a m-tape index $\vec{j}$ if this order stands on every tape, and we shall denote this relation by $\vec{i} \leqslant \vec{j}$.*

The m-tape index $\vec{1}$ has the value 1 on all tapes. The sum of two m-tape indexes $\vec{i}_1$ and $\vec{i}_2$ is a m-tape index $\vec{j}$ defined by $\vec{j}^{(k)} = \vec{i}_1{}^{(k)} + \vec{i}_2{}^{(k)}$. Thus, $\vec{i} \leqslant \vec{j}$ also means $\vec{j} - \vec{i} \geqslant \vec{0}$.

Definition (m-tape input string). *Given m alphabets $\Sigma^{(i)}$ $(1 \leqslant i \leqslant m)$, a m-tape input string is a vector of m strings $(a^{(1)}_{\vec{1}^{(1)}} \dots a^{(1)}_{\vec{n}^{(1)}}, \dots, a^{(m)}_{\vec{1}^{(m)}} \dots a^{(m)}_{\vec{n}^{(m)}})$, where each string $a^{(i)}_{\vec{1}^{(i)}} \dots a^{(i)}_{\vec{n}^{(i)}}$ belongs to $(\Sigma^{(i)})^*$. We shall denote the set of such defined input strings by $\langle \Sigma^* \rangle = \bigotimes_{i=1\dots n} \Sigma^{(i)*}$.*

As a shorthand, any m-tape input string $(a^{(1)}_{\vec{1}^{(1)}} \dots a^{(1)}_{\vec{n}^{(1)}}, \dots, a^{(m)}_{\vec{1}^{(m)}} \dots a^{(m)}_{\vec{n}^{(m)}})$ may be denoted by $a_{\vec{1}} \dots a_{\vec{n}}$. Substrings of $a_{\vec{1}} \dots a_{\vec{n}}$ will be denoted by $a_{\vec{i}} \dots a_{\vec{j}} = (a^{(1)}_{\vec{i}^{(1)}} \dots a^{(1)}_{\vec{j}^{(1)}}, \dots, a^{(m)}_{\vec{i}^{(m)}} \dots a^{(m)}_{\vec{j}^{(m)}})$ with the usual conventions that $\vec{1} \leqslant \vec{i}$, $\vec{j} \leqslant \vec{n}$ and, if $\vec{i}^{(k)} > \vec{j}^{(k)}$, $a^{(k)}_{\vec{i}^{(k)}} \dots a^{(k)}_{\vec{j}^{(k)}} = \epsilon$ (the empty string).

Example 1. *$(abba, dcd)$ is a 2-tape input string on $\Sigma^{(1)} = \{a, b\}$ and $\Sigma^{(2)} = \{c, d\}$. We shall also write this 2-tape input string as $\binom{abba}{dcd}$, which is a somewhat more natural notation in the context of alignments. This 2-tape input string $a_{\binom{1}{1}} \dots a_{\binom{4}{3}} = \binom{abba}{dcd}$ has a 2-tape input substring $a_{\binom{2}{1}} \dots a_{\binom{3}{2}} = \binom{bb}{dc}$.*

Definition (m-tape alphabet). *A m-tape alphabet Σ is a product of m alphabets $\Sigma^{(i)}$ augmented with the empty string: $\Sigma = \bigotimes_{i=1\dots m} (\Sigma^{(i)} \cup \{\epsilon\})$.*

Definition (m-tape alignment). *An element $a_1 \dots a_l$ of the free monoid Σ^*, generated by formal concatenation of m-tape elements of Σ, is called a m-tape alignment of length l. The empty alignment of Σ^* is denoted by ϵ.*

Definition (ϵ-deletion). *Given any m-tape alignment $a_1 \dots a_l$, we get a m-tape input string $a_{\vec{1}} \dots a_{\vec{n}}$ by concatenation of symbols of the projection of $a_1 \dots a_l$ on every tape.*

$$\Sigma^* \longrightarrow \langle \Sigma^* \rangle$$
$$a_1 \dots a_l \longrightarrow a_{\vec{1}} \dots a_{\vec{n}} = \langle a_1 \dots a_l \rangle$$

Example 2. *The 2-tape input string $\binom{abba}{dcd}$ may be defined as an ϵ-deletion of the alignments $\langle \binom{\epsilon}{d}\binom{a}{\epsilon}\binom{b}{\epsilon}\binom{b}{c}\binom{a}{d} \rangle$ or $\langle \binom{a}{\epsilon}\binom{b}{d}\binom{\epsilon}{c}\binom{b}{\epsilon}\binom{a}{d} \rangle$.*

Every alignment comes from a single m-tape input string, but the converse is not true and in fact, as soon as $m > 1$, the number of alignments of m strings of roughly the same size increases exponentially with this size. The definition of alignment we just gave is a generalization of concepts which may be found in (Myers 1991; Searls & Murphy 1995). The main difference here is that we explicitly identify alignments as being strings over a special m-tape alphabet.

Searls did show that the alignment of two strings according to some edit-distance may be carried out by some simple 2-tape nondeterministic finite automaton (NFA) whose transitions are weighted (Searls & Murphy 1995). An alignment is obtained from a sequence of transitions of the NFA which reads all of the input tapes and which has a minimal total weight. Now, we noticed that the set of alignments recognized by any of Searls' NFA's is in fact a regular language over what we call a 2-tape terminal alphabet, and that this regular

$$
\begin{array}{rcll}
start & \longrightarrow & frame0 & (0) \\
frame0 & \longrightarrow & frame0 \begin{bmatrix} X \\ X \end{bmatrix} \mid \begin{bmatrix} - \\ - \end{bmatrix} & (0) \\
 & \mid & frame0 \begin{bmatrix} X \\ Y \end{bmatrix} & (2) \\
 & \mid & frame1 \begin{bmatrix} - \\ X \end{bmatrix} \mid frame2 \begin{bmatrix} X \\ - \end{bmatrix} & (1) \\
frame1 & \longrightarrow & frame1 \begin{bmatrix} X \\ X \end{bmatrix} & (1) \\
 & \mid & frame1 \begin{bmatrix} X \\ Y \end{bmatrix} & (3) \\
 & \mid & frame0 \begin{bmatrix} X \\ - \end{bmatrix} \mid frame2 \begin{bmatrix} - \\ X \end{bmatrix} & (3) \\
frame2 & \longrightarrow & frame2 \begin{bmatrix} X \\ X \end{bmatrix} & (1) \\
 & \mid & frame2 \begin{bmatrix} X \\ Y \end{bmatrix} & (3) \\
 & \mid & frame0 \begin{bmatrix} - \\ X \end{bmatrix} \mid frame1 \begin{bmatrix} X \\ - \end{bmatrix} & (3)
\end{array}
$$

Figure 1: In this weighted left-regular grammar, weights are written in parentheses after each group of productions having the same weight. Later on, weights will be turned into attribute evaluation functions.

language may actually be described by a regular grammar whose productions are weighted. The weighted regular grammar of figure 1 is a simple translation of a frame-maintenance aligner given in (Searls & Murphy 1995). As regular grammars are a proper subset of context-free grammars, we found natural to generalize this idea of alignment to m-tape (i.e. the terminal alphabet is a subset of a m-tape alphabet) context-free grammars (MTCFG's) and their recognizing devices, namely m-tape nondeterministic pushdown automata (NPDA's). Weighted transitions of NFA's are easily translated into weighted pop-transitions of NPDA's. An alignment is obtained from a sequence of pop-transitions of a NPDA which reads all of the input tapes and which has an optimal (minimal for some problems, maximal for others, etc...) total weight.

What does this generalization of m-tape NFA's and regular grammars to m-tape NPDA's and context-free grammars bring ? Note that the traditional 1-tape NPDA model (or context-free grammar model) may describe unknotted foldings of RNAs (Vauchaussade de Chaumont 1985; Searls 1992) and that the m-tape NFA model may handle alignments. Hopefully, the generalization of both models to m-tape NPDA's will handle all methods which can be described by one of the two previous models **plus** methods which try to simultaneously fold and align RNAs. Figure 2 shows how alignments and structures may be deduced from a single m-tape derivation.

As NPDA's lack NFA's ability to be easily written and understood, we shall rather work with a m-tape context-free grammar than with an equivalent NPDA. This choice will not hamper the NFA case because a regular grammar is as easy to write and understand as

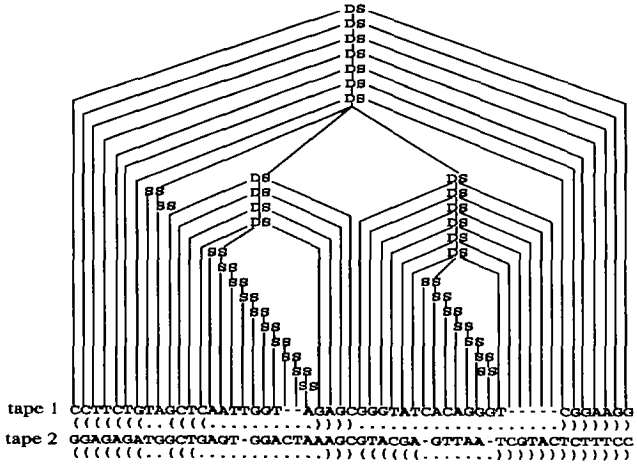


Figure 2: Derivation tree of an alignment of two RNAs. The underlying grammar may be easily recovered. Base pairings are infered from derivations of DS (Double-Strand) and they are given below each tape. Notice that a double-strand has been defined as a substructure whose ends must be paired on at least one tape, whereas a single-strand (SS) may only have unpaired bases on both tapes.

an equivalent NFA (figure 1). We shall use the following definition of m-tape context-free grammars.

Definition (m-tape context-free grammar). *A m-tape context-free grammar $G = (V_T, V_N, P, S)$ consists of a finite set of terminals V_T such that V_T* **is a subset of a m-tape alphabet**, *a finite set of nonterminals V_N such that $V_N \cap V_T = \emptyset$, 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 of the grammar. Each production in P has the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in V^*$. A is the left-hand side of the production and α its right-hand side.*

The reflexive transitive closure of $\rightarrow$ will be denoted by $\rightarrow^*$. A **derivation tree** is a planar representation of a sequence of derivations (replacements of a nonterminals A in a string of V^* by strings α such that $A \rightarrow \alpha$) and it is a result of **parsing**. The set of m-tape input strings $L(G) = \{\langle u \rangle \in \langle V_T^* \rangle \mid S \rightarrow^* u\}$ is the language generated by G. Notice that $L(G)$ is not the set of derivable strings but the set of ϵ-deleted such strings. This distinction disappears in the single tape case.

Example 3. *The following toy MTCFG will align two properly parenthesized strings interspersed with a's:*

$$S \rightarrow \begin{bmatrix} (\\ (\end{bmatrix} S \begin{bmatrix}) \\ ) \end{bmatrix} \mid \begin{bmatrix} a \\ a \end{bmatrix} \mid \begin{bmatrix} a \\ - \end{bmatrix} \mid \begin{bmatrix} - \\ a \end{bmatrix} \begin{bmatrix} - \\ - \end{bmatrix} \mid SS$$

In this MTSAG, the structure defined by parentheses must be the same on both tapes, but substrings of a may be aligned with gaps (denoted by $-$ in terminals

instead of ϵ, because a dash is appropriate, and even expected, in the context of alignments).

We shall sometimes assume that a grammar G is **proper**. This means that no production or nonterminal is useless, no nonterminal A may verify $A \rightarrow^+ A$, and no rule may have the form $A \rightarrow \epsilon$. Hence the number of derivation trees of a given string stays finite. This is not a severe restriction because it is always possible to convert a context-free grammar G' into an equivalent proper context-free grammar G by means of simple transformations (Aho & Ullman 1972). Moreover, derivation trees of G have the desirable property of merely being derivation trees of G' from which empty sub-derivations have been removed and circular derivations $A \rightarrow^+ A$.

Definition (projected grammar).
Let $G = (V_T, V_N, P, S)$ be a proper m-tape context-free grammar. For every tape i ($1 \leqslant i \leqslant m$), define the projected grammar $G^{(i)}$ as the conversion of the grammar $(V_T^{(i)}, V_N, P^{(i)}, S)$ into a proper grammar, where $V_T^{(i)}$ and $P^{(i)}$ are the sets of values on tape i of all the elements of V_T and P respectively.

Example 4. *The toy MTCFG of example 3 has the same projected grammar on both tapes :*

$$S \rightarrow (\ S \ ) \ \mid \ (\ ) \ \mid \ a \ \mid \ SS$$

Projected grammars are useful for the study of the complexity of our parsing algorithm as a function of the ambiguity of MTCFG's. An **ambiguous** grammar is a grammar for which there exists a string having at least two different derivation trees. All grammars we shall use to find alignments or foldings will be ambiguous, because of the exponential number of such alignments or foldings in the size of input strings. But grammars may be more or less ambiguous, and thus lead to more or less efficient parsers. We shall see later that it is easier to parse strings for grammars which have some unambiguous projections than for grammars which have no unambiguous projection.

We said earlier that we could evaluate a cost for each alignment or folding produced by a NPDA, thanks to weights on pop-transitions. This cost-evaluation step is essential for the determination of an optimal cost alignment or folding. We could have converted weighted pop-transitions of NPDA's into weighted grammar productions, but we preferred to use the more general mechanism of synthesized attributes, or S-attributes which, together with MTCFG's, give us m-tape S-attribute context-free grammars, or MTSAG's. S-attributes are attributes which are assigned to every vertex of a derivation tree and which are computed from the bottom of a derivation tree (i.e. every terminal has a known S-attribute) to its root by means of attribute evaluation functions associated to grammar productions. Thanks to these functions, the computation of the final attribute of the derivation tree does not have to rely on a fixed, predetermined, operation (summation, multiplication, ...), as it would have been the case with weighted productions. In our implementation, attribute evaluation functions are C functions. We have already shown the effectiveness of S-attributes with our adaptation of the thermodynamic model of folding to context-free grammars (Lefebvre 1995).

Definition (m-tape S-attribute grammar).
A m-tape S-attribute grammar is denoted by $G = (V_T, V_N, P, S, \mathcal{A}, S_{\mathcal{A}}, F_P)$. It is an extension of a m-tape context-free grammar $G = (V_T, V_N, P, S)$, 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 V_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 (hence the bottom-up computation of attributes).

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

Syntax analysis for MTSAG's

A generalization of Cocke–Younger–Kasami's algorithm (CYK) would be an easy algorithm to parse m-tape input strings. This algorithm has a time complexity of $O(n^3)$ and a space complexity of $O(n^2)$ when only one tape of size n is considered (Aho & Ullman 1972). A generalization to m tapes, each of size n, would lead to an algorithm having a complexity of $O(n^{3m})$ in time and $O(n^{2m})$ in space [1]. What makes CYK's algorithm not so appealing (besides the fact that it requires grammars in Chomsky normal form) is that

[1] Intuitively, a 1-tape parsing algorithm has to deal with substrings of the input string. If the input string has n symbols, there are $O(n^2)$ such substrings, hence the $O(n^2)$ space complexity. And since each substring may be split in two at an unknown location between both ends (to handle rules $A \rightarrow B\,C$), this gives us the $O(n^3)$ time complexity. Since the same reasoning may be applied independently to each tape, we get an informal proof of the complexities stated above

these complexity bounds are actually reached for every useful MTSAG. While we won't be able to lower complexity bounds for general MTSAG's, we want to take advantage of the fact that some MTSAG's have unambiguous or even left-regular projections, and of the fact that some states of the parser are visited only rarely, or in a systematically incomplete manner (for instance, each frame of figure 2 is visited one third of the time).

To fulfill these goals and overcome the limitations of CYK's algorithm, we generalized our parsing algorithm for 1-tape MTSAG's (Lefebvre 1995). The implementation of this 1-tape parsing algorithm proved to be fast and memory efficient compared to some implementations of other parsing algorithms, and it has the desirable property of having lower complexity bounds when the input grammar is unambiguous ($O(n^2)$ time and space), or even LR(k)-like ($O(n)$ time and space). While most modifications made to the original presentation of the algorithm were quite straightforward[2], we give here a description of the modified algorithm for the sake of completeness.

Definition (items).
Let $G = (V_T, V_N, P, S, \mathcal{A}, S_{\mathcal{A}}, F_P)$ be a proper MTSAG. We define the set E_{GCP} of items as follows:

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

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 note $[\Delta \rightarrowtail \alpha, \vec{\imath}, \lambda]$, and also call item, the association of an item $[\Delta \rightarrowtail \alpha]$ with a m-tape index $\vec{\imath}$ and a string of attributes λ, each attribute of λ being associated in sequence to a corresponding symbol of α.

Definition (parse table, useful items). *The parse table of a m-tape input string $a_{\vec{1}} \ldots a_{\vec{n}}$ is a vector of entries $(T_{\vec{\jmath}})_{\vec{0} < \vec{\jmath} \leqslant \vec{n}}$. Each entry $T_{\vec{\jmath}}$ of a parse table is a set of items $[\Delta \rightarrowtail \alpha, \vec{\imath}, \lambda]$ so constructed as to satisfy the following minimum condition of usefulness:*

$$\begin{aligned} \forall [\Delta \rightarrowtail \alpha, \vec{\imath}, \lambda] \in T_{\vec{\jmath}}, \forall A \in \Delta, \exists (\beta, \gamma, u, v), \\ S \rightarrow^* uA\gamma \rightarrow u\alpha\beta\gamma \rightarrow^* v\beta\gamma \;\wedge \\ \langle u \rangle = a_{\vec{1}} \ldots a_{\vec{\imath}} \;\wedge \\ \langle v \rangle = a_{\vec{1}} \ldots a_{\vec{\jmath}} \end{aligned}$$

[2]But we must acknowledge that this formal similarity between our original 1-tape parsing algorithm and the current m-tape parsing algorithm hides numerous implementation issues which we can't discuss in this paper. For instance, our general m-tape parsers had to be as fast as our original one-tape-only parsers when $m = 1$.

This minimum condition of usefulness means that an item is never added to an entry $T_{\vec{\jmath}}$ if it has no chance of being used in a derivation tree of an input string $a_{\vec{1}} \ldots a_{\vec{n}}$ whose already analyzed substring is $a_{\vec{1}} \ldots a_{\vec{\jmath}}$. This is the key to the lower parsing complexities when some projections of the MTSAG are unambiguous.

Definition (Reduce, $\Delta_{\vec{\jmath}}$). *We shall denote by* Reduce *the function returning the set of reducible nonterminals of an item $[\Delta \rightarrowtail \alpha]$:*

$$\mathsf{Reduce}\big([\Delta \rightarrowtail \alpha]\big) = \{A \in \Delta \mid A \rightarrow \alpha\}.$$

Once $T_{\vec{\jmath}}$ has been constructed, the set of nonterminals which may lead to the extension of at least one item of $T_{\vec{\jmath}}$ is denoted by $\Delta_{\vec{\jmath}}$ and is equal to :

$$\Delta_{\vec{\jmath}} = \Big\{ D \in V_N \;\Big|\; \exists [\Delta \rightarrowtail \alpha, \vec{\imath}, \lambda] \in T_{\vec{\jmath}}, \exists A \rightarrow \alpha B \beta, \exists \gamma, \; A \in \Delta \;\wedge\; B \rightarrow^* D\gamma \Big\}$$

Notice that $T_{\vec{0}}$ does not exist, so we shall denote the set of nonterminals expected at the beginning of a derivation by $\Delta_{\vec{0}} = \{A \in V_N \mid \exists \beta, S \rightarrow^* A\beta\}$.

By definition of $\Delta_{\vec{\jmath}}$, the set Δ of left-nonterminals of an item $[\Delta \rightarrowtail \alpha, \vec{\jmath}, \lambda]$ must be included in $\Delta_{\vec{\jmath}}$ so that every nonterminal $A \in \mathsf{Reduce}\big([\Delta \rightarrowtail \alpha]\big)$ may be used by at least one item of $T_{\vec{\jmath}}$. Hence the following definition for the functions responsible for the creation and the extension of items.

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

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

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

The item resulting from the extension of an item by a symbol is given by the following Goto *function:*

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

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

We shall say that a symbol X has a left-position $\vec{\imath}$ and a right-position $\vec{\jmath}$ if $X \rightarrow^* u \;\wedge\; \langle u \rangle = a_{\vec{\imath}+\vec{1}} \ldots a_{\vec{\jmath}}$. Such a symbol X is used during parsing in the following way:

- If X is a terminal a, then $a = a_{\vec{\imath}+\vec{1}} \ldots a_{\vec{\jmath}}$. The attribute of a is $x = S_{\mathcal{A}}(a)$. Then, X may either create an item $[\Delta \rightarrowtail X, \vec{\imath}, x]$ such that $[\Delta \rightarrowtail X] = \mathsf{Initial}(\Delta_{\vec{\imath}}, X)$, or an item $[\Delta \rightarrowtail \alpha X, \vec{h}, \lambda x]$ such that $[\Delta' \rightarrowtail \alpha, \vec{h}, \lambda] \in T_{\vec{\imath}}$ and $[\Delta \rightarrowtail \alpha X] = \mathsf{Goto}\big([\Delta' \rightarrowtail \alpha], X\big)$. Once $\Delta_{\vec{\imath}}$ and all items of $T_{\vec{\imath}}$ have been considered for extension, no other operation is possible with X.

- If X is a nonterminal A, it must come from the reduction of an item $[\Delta' \rightarrowtail \alpha', \vec{i}, \lambda'] \in T_{\vec{j}}$. The attribute of A is $x = f_{A\rightarrow\alpha}(\lambda')$. In the same way as with terminals, X may either create items $[\Delta \rightarrowtail X, \vec{i}, x]$ or items $[\Delta \rightarrowtail \alpha X, \vec{h}, \lambda x]$. The newly created items may in turn be reduced into other nonterminals which will lead to the creation of other items, etc...

A nonterminal A whose left-position and attribute are $\vec{i}$ and x may influence the attribute x' of an other nonterminal A' whose left-position is $\vec{i}'$ if (necessary condition) $\vec{i} > \vec{i}'$ or if $\vec{i} = \vec{i}'$ and $A' \rightarrow^+ A$. If $(A, \vec{i}, x)$ and $(A', \vec{i}', x')$ are simultaneously reducible from items already present in $T_{\vec{j}}$, then $(A, \vec{i}, x)$ must be reduced before $(A', \vec{i}', x')$ if we want to let it a chance to influence the attribute x' of $(A', \vec{i}', x')$. For that reason, we introduce the following priority queue.

Definition (priority queue). *Let Q be a set of triplets $(X, \vec{i}, x)$. We say that $(X, \vec{i}, x) < (X', \vec{i}', x')$ if $\vec{i} < \vec{i}'$ or if $\vec{i} = \vec{i}'$ and $X \rightarrow^+ X'$. The relation $<$ defines a partial order over the set of such triplets. This relation is extended into a total order on triplets, hence there always exists a maximum triplet in Q. We shall denote by $(X, \vec{i}, x) := \mathsf{Extract}(Q)$ the extraction of the maximum triplet from Q. The $<$ relation on triplets will also be called a priority relation and the maximum triplet will be called the greatest priority triplet.*

The reduction of an item $[\Delta \rightarrowtail \alpha, \vec{i}, \lambda] \in T_{\vec{j}}$ into $\big(A, \vec{i}, f_{A\rightarrow\alpha}(\lambda)\big)$ means that we found a sub-derivation $A \rightarrow \alpha \rightarrow^* u$ with $\langle u \rangle = a_{\vec{i}+\vec{1}} \ldots a_{\vec{j}}$. Many sub-derivations of $a_{\vec{i}+\vec{1}} \ldots a_{\vec{j}}$ from A are usually possible and, during parsing, we must expect several reductions into $\big(A, \vec{i}, f_{A\rightarrow\alpha}(\lambda)\big)$ with a different λ each time. We can't store and use all such reductions independently or else our algorithm would turn into an exponential time and space algorithm. Now, we must remember that, in reality, we are looking for sub-derivations which are optimal according to some criterion (maximum probability, minimum energy, minimum edit-distance, ...). Thus, for every possible A, $\vec{i}$, $\vec{j}$, we only have to optimize the attribute x of $(A, \vec{i}, x)$ (note that $\vec{j}$ is implicit in the triplet because the triplet appears while we are computing $T_{\vec{j}}$). To that effect, we define a constraint $\mathcal{C}_A$ associated with A which, given two triplets $(A, \vec{i}, x)$ and $(A, \vec{i}, x')$, will replace those two triplets by a single triplet $(A, \vec{i}, x'')$ with x'' deduced from x and x'. For instance, if attributes are probabilities and if we seek the most probable derivation tree, then $\mathcal{C}_A$ will return $\big(A, \vec{i}, \max(x, x')\big)$.

A more formal presentation of these remarks leads to the following algorithm.

Algorithm 1.
Let Q be the previously defined set of triplets $(X, \vec{i}, x)$. At the beginning, all entries $T_{\vec{j}}$ are empty. For $\vec{j}$ going from $\vec{0}$ (excluded) to $\vec{n}$, in increasing order (if $\vec{j}_1 < \vec{j}_2$, then $\vec{j}_1$ is processed before $\vec{j}_2$), perform the following steps:

1. *Initialize the queue:*
$$Q := \Big\{ \big(a, \vec{i}, S_A(a)\big) \;\Big|\; \vec{i} < \vec{j} \wedge\ a = a_{\vec{i}+\vec{1}} \ldots a_{\vec{j}} \in V_T \Big\}$$
2. *Extract the triplet having the greatest priority:*
$(X, \vec{i}, x) := \mathsf{Extract}(Q)$
3. *Creation of items beginning with X:*
$$T_{\vec{j}} := T_{\vec{j}} \cup \Big\{ [\Delta \rightarrowtail X, \vec{i}] \;\Big|\; [\Delta \rightarrowtail X] = \mathsf{Initial}(\Delta_{\vec{i}}, X) \Big\}$$
4. *Extension of items by X:*
$$T_{\vec{j}} := T_{\vec{j}} \cup \Big\{ [\Delta \rightarrowtail \alpha X, \vec{h}, \lambda x] \;\Big|\; \exists [\Delta' \rightarrowtail \alpha, \vec{h}, \lambda] \in T_{\vec{i}},\ [\Delta \rightarrowtail \alpha X] = \mathsf{Goto}\big([\Delta' \rightarrowtail \alpha], X\big) \Big\}$$
5. *For every item $[\Delta \rightarrowtail \alpha, \vec{i}, \lambda]$ added to $T_{\vec{j}}$ by steps 3 and 4 above, and for every $A \in \mathsf{Reduce}\big([\Delta \rightarrowtail \alpha]\big)$, compute $(A, \vec{i}, x) := \big(A, \vec{i}, f_{A\rightarrow\alpha}(\lambda)\big)$. If there already exists a triplet $(A, \vec{i}, x') \in Q$, replace this triplet by $\mathcal{C}_A\big((A, \vec{i}, x), (A, \vec{i}, x')\big)$. If no such triplet exists, add $(A, \vec{i}, x)$ to Q.*
6. *Repeat steps 2 to 6 while Q is not empty;*
7. *Compute $\Delta_{\vec{j}}$.*

The string $a_{\vec{1}} \ldots a_{\vec{n}}$ belongs to $L(G)$ if and only if there exists an item $[\Delta \rightarrowtail \alpha, \vec{0}, \lambda] \in T_{\vec{n}}$ such that $S \in \mathsf{Reduce}\big([\Delta \rightarrowtail \alpha]\big)$ (proof not shown). We say that the string has been recognized by the algorithm and that it has an attribute λ.

Proposition 1 (1-tape complexity). *Let G be a proper 1-tape MTSAG and let $r \geqslant 1$ be the maximum number of nonterminals appearing at the right-hand side of a production of G. For a tape of length n, the time and space complexities of the previous parsing algorithm are, in order of decreasing constraints on G:*

- *Equal and at most $O(n)$ if G is LR(k) and not right-recursive (this encompasses left-regular grammars);*
- *Equal and at most $O(n^2)$ if G is unambiguous;*
- *$O(n^{r+1})$ and $O(n^r)$ for a generic proper MTSAG.*

Proposition 2 (m-tape complexity). *Let G be a proper m-tape MTSAG. The time complexity of our parsing algorithm on G is equal to the product of the parsing complexities of the same algorithm applied on each tape i with each projected grammar $G^{(i)}$. The same kind of result holds for space complexities. Hence*

the time complexity is at most $O(n^{m(r+1)})$, and the space complexity is at most $O(n^{mr})$, for m-tapes of size n

In practice, MTSAG's that we used verified $r \leqslant 2$ and thus the time and space complexities of our parsers for those grammars were respectively $O(n^{3m})$ and $O(n^{2m})$ at most. We shall give in this paper some examples of useful 2-tape grammars having a LR(1), non right-recursive, first projected grammar, and an ambiguous second projected grammar. On these grammars, our parsing algorithm has a $O(n^4)$ time complexity and $O(n^3)$ space complexity if both tapes have n symbols. Thus the introduction of projected grammar is not solely of theoretical interest, but also of practical interest.

In the above complexities, we omitted a factor which is a linear function of the grammar size. This factor has been hidden in the O notation because the grammar is usually fixed, while n is increasing. When this is not the case (for SCFG's for example), we will explicitly add this factor to our complexities.

Availability

An essential aspect of MTSAG's is the ability to easily generate efficient parsers from grammars. On the basis of the tool we had already written to generate parsers from 1-tape S-attribute grammars, we designed a new tool, MTSAG2C, which automatically generates the C source of a parser from a given MTSAG. The syntax used by our tool is YACC-like (figure 4). The generated parser is able to read tapes (thanks to a lexical analyzer provided by the user), parse tapes, and then output a single derivation tree which satisfies constraints given in the MTSAG. Those constraints often are maximization or minimization constraints on integers or reals. The user may, on the basis of generated parsers, build training algorithms such as the Expectation-Maximization algorithm used with SCFG's. This last algorithm is especially easy to implement because it only involves updates of tables of probabilities based on found derivation trees.

Source code and documentation of MTSAG2C, our parser generator, are available by anonymous ftp, together with some examples, from `ftp://lix.polytechnique.fr/pub/lefebvre`. This directory will be regularly updated as we bring improvements to MTSAG2C and its documentation. The code is ANSI C and is portable among various UNIX platforms (only tested on PCs, Suns and Alphas). The code of MTSAG2C should not be too hard to understand, thanks to the literate programming tool we used in the course of development.

Results

When writing MTSAG's, it would be too tedious to write down all rules for every possible m-tape terminal, or every possible pair of symbols. Thus we introduced the concept of parentheses of finite order, declared in the MTSAG with a `%parentheses`. For instance,

```
%parenthesis .  A U C G
```

declares that the dot . range over any of the four fundamental ribonucleotides. Likewise,

```
%parenthesis ( A U C G    ) U A G C
```

declares that any occurrence of an open-parenthesis on some tape in a rule, followed by a closing parenthesis on the same tape, will be replaced in sequence by all Watson-Crick base pairs. Thus, the following rule:

```
DoubleStrand : [( .] DoubleStrand [) _]
```

defines a nonterminal *DoubleStrand* which ends with a base pair AU, UA, GC or CG on the first tape, and which has a deleted final base on the second tape of the alignment. We have no room to describe further the syntax used by MTSAG2C, but the reader will notice in our examples that it is very like YACC's syntax especially for rules or attribute evaluation functions (which are like the semantic actions of YACC).

frame-maintenance aligner

The frame maintenance aligner given in figure 1 was readily converted into the following MTSAG2C-ready MTSAG:

```
%nb_tapes 2          %empty _     %token C G T A
%attribute "int" v   %parentheses .  C G T A
%terminal_attribs{}  %default_nonterminal_attribs <v>
%default_constraint  { if ($$.v < $!.v) $!.v = $$.v; }
%%
frame0 : frame0 [.  .]  { if ($2[1].left[0] != $2[2].left[0])
                            $$.v=$1.v+2;
                          else $$.v=$1.v; }
       | frame1 [_ .]   { $$.v = $1.v + 1; }
       | frame2 [.  _]  { $$.v = $1.v + 1; }
       | [_ _]          { $$.v = 0; } ;

frame1 : frame0 [.  _]  { $$.v = $1.v + 3; }
       | frame1 [.  .]  { if ($2[1].left[0] != $2[2].left[0])
                            $$.v = $1.v + 3;
                          else $$.v = $1.v + 1; }
       | frame2 [_ .]   { $$.v = $1.v + 3; } ;

frame2 : frame0 [_ .]   { $$.v = $1.v + 3; }
       | frame1 [.  _]  { $$.v = $1.v + 3; }
       | frame2 [.  .]  { if ($2[1].left[0] != $2[2].left[0])
                            $$.v = $1.v + 3;
                          else $$.v = $1.v + 1; } ;
```

Since every projection of this MTSAG is left-linear, parsing may be done by our algorithm in $O(n_1 n_2)$ time

and space, where n_1 and n_2 are the respective sizes of each input tape.

We compared the parser generated from this grammar with a program handwritten from the dynamic programming relations deduced from the grammar. Our results are gathered in the following table (all tests were done on an Alpha 2100-500MP):

	MTSAG	handwritten
2 tapes of 125 bases		
time in seconds:	0.68	0.01
space in Mbytes:	0.9	0.2
2 tapes of 250 bases		
time in seconds:	2.7	0.05
space in Mbytes:	3.4	0.7
2 tapes of 500 bases		
time in seconds:	10.9	0.25
space in Mbytes:	13.5	2.9

Both programs obviously have quadratic time and space complexities. The parser is forty times slower than the handwritten program, and it takes roughly 5 times more memory. This result is not surprising because our parsers are optimized for ambiguous grammars and they always drag structures which are there to handle ambiguities, and which are useless in this case. And the parser can't automatically detect that the grammar is unambiguous, because this problem is provably unsolvable for context-free grammars (Salomaa 1973). Note that it took us only ten minutes to write the MTSAG as opposed to more than two hours to write *and debug* a program from dynamic programming relations. MTSAG's are still useful for the rapid prototyping of new edit-distance based alignment algorithms because the generated code will never be as fast as hand-optimized code for such simple grammars. On the plus side, the parser correctly notices that each frame is used one third of the time, and it allocates memory for used frames only, whereas our handwritten program unconditionally allocates one full matrix for each frame.

The previous MTSAG may be "speeded up" if we know that we shall only try to align relatively similar sequences. Such sequences seldom are desynchronized of more than a few bases in alignments. The following constraint consider that the desynchronization between both tapes can't exceed 20 bases ($$[i].len is the length on tape i of the m-tape input substring derived from the nonterminal whose attribute $$ is being computed).

Example 5.

```
%default_constraint { if (labs($$[1].len-$$[2].len)>20)
                        DO_NOT_SELECT;
                      if ($$.v < $!.v) $!.v = $$.v; }
```

With this constraint, the alignment of two sequences 500 bases long takes 2.5 seconds instead of 10.9 seconds and memory requirements drop to 9.1 Mbytes instead of 13.5 Mbytes. We see in this example that it is quite easy to introduce **optimizing constraints** (constraints which optimize the time and memory taken to solve a particular problem), which is not necessarily the case with handwritten programs.

Stochastic Context-Free Grammars

The first and obvious way to apply MTSAG's to SCFG's is to consider a 1-tape MTSAG having the same rules and vocabulary as a given SCFG. Probabilities of rules are turned into attribute evaluation functions. For this type of MTSAG, we get a parser which has a $O(n^3 l)$ time complexity and a $O(n^2 l)$ space complexity, where l is the number of nonterminals in the grammar and n is the number of bases of parsed RNAs (Grate 1995). Since a SCFG is often designed to get an accurate model of a family of RNAs (with a view to the discrimination and alignment of new RNAs), l and n are roughly equal to the average length of RNAs of this family. This turns parsing into a $O(n^4)$ time and $O(n^3)$ space process. Thus SCFG's are interesting to fold, align and discriminate RNAs, but they suffer from huge grammars which are nearly impossible to manage manually. This is why tools have been designed to automatically build a SCFG from a high-level description of the structure of the family of RNAs one wish to model. This approach has the additional drawback of having to program and maintain tools which cannot always be described in papers. The use of such tools lengthens development cycles since every change in the grammar implies a recompilation step (figure 3(a)).

The second and probably most interesting way to apply MTSAG's to SCFG's is to use a 2-tape MTSAG and transfer on the first tape the high-level description of the structure of a family of RNAs, and on the second tape the RNA to be folded and aligned. All conversion rules used by the SCFG generating tool are then written down as a single, fixed, MTSAG. This has the additional benefit of shortening the development cycle (cf. figure 3(b)). Furthermore, the interpretation of the high-level description of the first tape is only a function of the MTSAG. Thus it is easy to document and change this interpretation. We shall give later an example of such a change.

To illustrate this point we transformed a SCFG used by Haussler's team to model tRNAs (Sakakibara *et al.* 1994). This grammar has 97 nonterminals and 660 rules, and it is quite hard to read and understand directly. Once turned into a high-level description suitable for the first tape of a MTSAG, it looked like the first tape of the 2-tape input string of figure 5(a). The MTSAG which was used to "interpret" this high-level

description is given in figure 4. This MTSAG has a non right-recursive LR(k) projection on the first tape, thus the parser generated has the same time and space complexities as the parser generated from the original SCFG: $O(n^3l)$ in time and $O(n^2l)$ in space, where n is the length of the second tape and l is the length of the first tape instead of the number of terminals of the original SCFG.

One of the advantages of MTSAG's is that one does not have to generate and compile another parser every time one modifies the high-level description of the family (figure 3(b)). Thus the following procedure may be used to learn SCFG's (this procedure is an adaptation of the procedure used by Eddy and Durbin to learn their CM's from initially unaligned and unfolded RNAs):

1. Use a MTSAG adaptation of any folding algorithm (Sankoff, Zuker) to get a rough (and even wrong) initial folding. Convert this folding to a suitable first tape (replace single strands by '*' for instance);
2. Use Dirichlet priors to estimate probabilities;
3. Align and fold all RNAs with this 2-tape MTSAG;
4. Optimize probabilities from results of the previous step and repeat the previous step until probabilities converge;
5. Use a comparative analysis algorithm on alignments returned by step 3 to get a new approximation of the common structural features of all RNAs. Then convert this approximation to a suitable first tape;
6. Repeat steps 2 to 6 until the first tape converges.

We compared the parser generated from a 1-tape version of the already cited 97 nonterminals SCFG (this parser already proved to be quite fast (Lefebvre 1995)) to the parser generated from the 2-tape version of this SCFG (figures 5(a) and 4). Tests were done on an Alpha 2100-500MP

	1-tape	2-tape
83 bases tRNA		
time in seconds:	0.45	0.33
space in Mbytes:	1.8	0.9

Intuitively, the 2-tape version is faster and takes less memory because it has to handle a small, constant, number of nonterminals (7 in our example) only, whereas the 1-tape parser has to handle a number of nonterminals which is a linear function of the length of modeled RNAs (97 here). And the 2-tape parser can be made still faster by adding optimizing constraints like the one in example 5 which says that alignments cannot be too desynchronized. With these optimizing constraints, which are really difficult to add to the original SCFG, the same 83 bases tRNA is parsed in 0.23 seconds and 0.8 Mbytes.

The same principles were applied to the SCFG U1 for snRNAs described by Underwood (Underwood 1994). snRNAs are more than twice longer than tRNAs and Underwood's SCFG was generated by a more sophisticated tool (i.e. the equivalent 2-tape MTSAG is more complex). Our 1-tape parser analyzed a 162 bases snRNA in 97 Mbytes and 78 seconds, whereas our 2-tape parser took only 15 Mbytes (more than 6 times less memory) and 28 seconds.

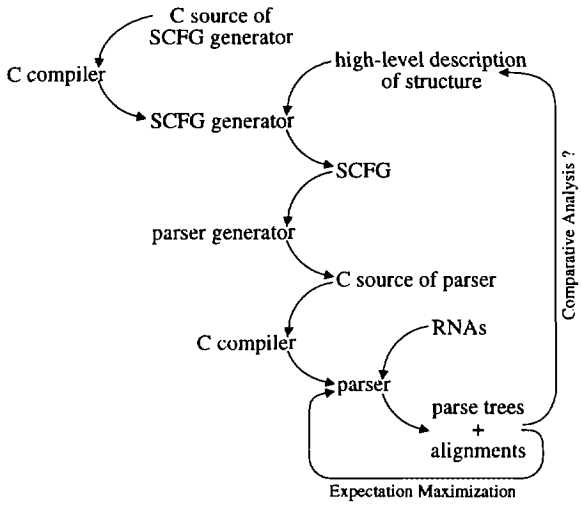
Application to some other models

For space considerations, we shall not give many details in the following examples. However, the full source code of cited MTSAG's is retrievable on our ftp server.

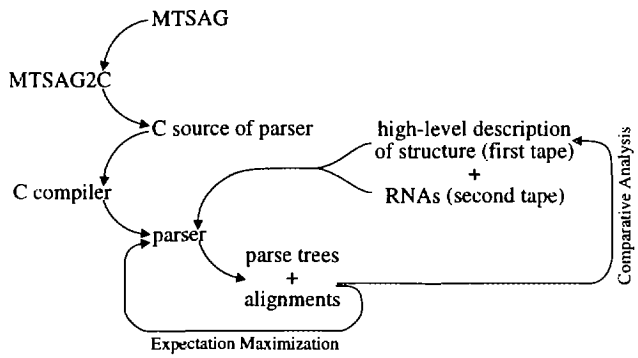
The principles and advantages of 2-tape MTSAG's for SCFG's also apply to HMM's and CM's:

- Both projections of a 2-tape MTSAG describing HMM's are left-regular. Thus parsing complexities fall to the awaited $O(mn)$ time and space.
- The various types of nodes of CM's may easily be encoded in a 2-tape MTSAG, actual nodes being relegated to the first tape. If the 2-tape MTSAG is well written, its projection on the first tape is LR(k), and we get back the complexities stated by Eddy and Durbin for the alignment process of their CM's.

A few years ago, Sankoff gave a dynamic programming method to align RNAs and simultaneously fold them into equivalent structures (Sankoff 1985). It was the first report of an optimal method to fold and align two RNAs using a thermodynamic model of secondary structures and a linear gap penalty model of alignments. A simple 2-tape generalization of the 1-tape MTSAG given last year for the thermodynamic model (Lefebvre 1995) easily captured Sankoff's model. Since both projections of this 2-tape MTSAG are highly ambiguous, our parsing complexities are the same as Sankoff's complexities ($O(n^6)$ in time and $O(n^4)$ in space). While theoretical complexities are the same, real requirements favor our strategy. Our 1-tape parser was already twice as fast as a manual implementation of the dynamic programming recurrences given by Zuker, with less than three times more memory. But our 2-tape parser based on Sankoff's model is more than fifty times as fast as a manual implementation of Sankoff's recurrences, still with less than three times more memory. This difference is so big that we even thought for a moment that we had bugs in our testing code. A thorough analysis revealed that this difference stems from the fact that dynamic programming relations given in papers are usually implemented with a bottom-up strategy, whereas our parsers use a bottom-up strategy with top-down filtering.



(a) 1-tape MTSAG.



(b) 2-tape MTSAG. It should be emphasized that the parser does not have to be changed when the high-level description is evolving.

Figure 3: Development cycle of a MTSAG implementation of SCFG's. It has been suggested (Grate 1995) that a comparative analysis of alignments resulting from parsing may be used to build a new SCFG or a new high-level description of it. With 2-tape MTSAG's, this kind of feedback is as easy to implement as the feedback designed for CM's by Eddy and Durbin (Eddy & Durbin 1994).

```
%nb_tapes 2
%empty _                %token C G U A '(' ')' '.' '*'          %parentheses . C G U A
%attribute "double" p   %terminal_attribs {/* code not included */}   %default_nonterminal_attribs <p>
%default_constraint { if ($$.p > $!.p) $!.p = $$.p; }
%constraint AddBases { if ($$[2].len>max_add_len ($$[1].left-1)) DO_NOT_SELECT; }
%%
RNA           : FixedStrand | VariableStrand | DSEndedRNA                { $$.p = $1.p; }
              | DSEndedRNA FixedStrand | DSEndedRNA VariableStrand       { $$.p = $1.p + $2.p; } ;

DSEndedRNA    : DoubleStrand                  { $$.p = $1.p; }
              | RNA DoubleStrand              { $$.p = $1.p + $2.p; } ;

DoubleStrand  : ['(' ./"("] RNA [')' ./")"]   { $$.p = $2.p + proba_ds ($1[1].left,$1[2].left[0],$3[2].left[0]); } ;

VariableStrand : AddOrDelete                  { $$.p = $1.p; }
              | VariableStrand AddOrDelete    { $$.p = $1.p + $2.p; } ;

AddOrDelete   : ['*' .] AddBases              { $$.p = $2.p + proba_quit ($1[1].left, $2[2].right[0]); }
              | ['*' .]                       { $$.p = proba_eat ($1[1].left, $1[2].left[0]); }
              | ['*' _]                       { $$.p = proba_skip ($1[1].left); } ;

AddBases      : AddBases [_ .]                { $$.p = $1.p + proba_stay ($1[1].left-1, $1[2].right[0]); }
              | [_ .]                         { $$.p = proba_add ($1[1].left - 1, $1[2].left[-1]); } ;

FixedStrand   : ['.' .]                       { $$.p = proba_ss ($1[1].left, $1[2].left[0]); }
              | FixedStrand ['.' .]           { $$.p = $1.p + proba_ss ($2[1].left, $2[2].left[0]); } ;
```

Figure 4: In this MTSAG, we are maximizing an attribute p (which is in fact the log of a probability), as can be seen in %default_constraint. All functions proba_*sth* are constructed from values given as attributes of symbols of the first tape. Besides, their first parameter is the left position on the first tape of the nonterminal whose attributes are being computed. The nonterminal *DoubleStrand* matches pairs of parentheses on the first tape with symbols on the second tape. These symbols are labelled with parentheses (thanks to /"(" and /")") to get an aesthetically pleasing output when printing parse trees. *AddOrDelete* and *AddBases* deal with variable length single strands marked by * on the first tape. *FixedStrand* deals with fixed length single strands marked by . on the first tape. The constraint on *AddBases* means that the number of symbols which may be inserted for every * of the first tape is bounded by a constant which is a function of the * considered.

```
(((((((..((((********)))).(((((.......)))))********((((.......))))))))))).
CCUUCUGUAGCUCAAUUGGUAGAGCAUGUGACUGUAGAGUAUGCGGGUAUCACAGGGUCGCUGGUUCGAUUCCGGCCGGAAGG
```

(a) unaligned 2-tape input string.

```
(((((((..((((********)))).(((((.......)))))*******----------*((((.......))))))))))).
CCUUCUGUAGCUCAAUUGGUAGAGCAUGUGACUGUAGAGUAUGC--GG-GUAUCACAGGGUCGCUGGUUCGAUUCCGGCCGGAAGG
```

(b) 2-tape alignment of the previous 2-tape input string.

Figure 5: unaligned and aligned version of a 2-tape input string. It is obvious that the first tape of this 2-tape input-string has a cloverleaf-like structure. This structure has two single strands which may have a variable length around 8 bases. The second tape is the RNA DY6050 extracted from a well known freely available compilation of tRNAs (Steinberg, Misch, & Sprinzl 1993)

To overcome the high complexities of the previous model, where both RNAs are initially unfolded and unaligned, some authors have tried to align and fold one RNA against already folded RNAs (Bafna, Muthukrishnan, & Ravi 1995; Corpet & Michot 1994). Once again, MTSAG's are able to successfully capture the subtleties of these methods. In fact, we just have to replace the folded RNA or the family of folded RNAs, against which we are about to align and fold new RNAs, by a special first tape where paired and unpaired bases are explicitly identified, by parentheses and dots for instance. Real distributions of underlying RNA bases are turned into attributes of those new special symbols. Then, it is straightforward to write MTSAG's which take into account the folding constraints and scoring methods of Bafna *et al.* (base pairs on the second tape must match base pairs on the first tape) or Corpet and Michot (base pairs on the first tape need not match base pairs on the second tape, but matching is favored). Then, we just have to run the parser generated from the chosen MTSAG with an initially unaligned, unfolded RNA on the second tape. Our parsers have the same complexities for these two models as programs written by the authors from their dynamic programming relations. For the model of Bafna *et al.*, the parser automatically "sees" an optimization which had to be uncovered by the authors. We compared the RNAlign tool of Corpet *et al.* to the parser automatically generated from a MTSAG version of their model. Both programs took 16 Mbytes and 50 seconds to align a tRNA against a family of already aligned and folded tRNAs).

Conclusion

We showed in this paper that there are several reasons why MTSAG's (especially 1-tape or 2-tape MTSAG's) are an interesting way to handle several popular models of alignment and/or folding. In the case of SCFG's, a new way to describe structures has been introduced in the form of 2-tape MTSAG's and special first tapes. This new way alleviates the need for specialized SCFG building tools and for recompilations of parsers every time the model is changed (only the first tape has to be changed). Moreover, parsers generated by MTSAG2C are fast and memory efficient, and still faster and more memory efficient with the addition of suitable optimizing constraints.

We also gave a sketch of a method to build stochastic models from unaligned, unfolded RNAs. However, divide and conquer methods may be a prerequisite for long RNAs (Corpet & Michot 1994; Grate 1995). We are trying to apply MTSAG's to these methods.

MTSAG's also fork new directions of investigation in parsing theory since, to the best of my knowledge, they have not been studied before in their full generality. For instance, note that MTCFG's generate a superset of context-free languages : there exists a 2-tape MTCFG which recognizes the context-sensitive "pseudo-knot language" $\{a^n b^m c^n d^m \mid n, m \geqslant 0\}$ (Brown 1995) if both input tapes are constrained to be the same (i.e. the associated 2-tape NPDA is really a 2-headed 1-tape NPDA):

$$S \to \left[\begin{smallmatrix} a \\ - \end{smallmatrix}\right] S \left[\begin{smallmatrix} c \\ a \end{smallmatrix}\right] I \mid \left[\begin{smallmatrix} b \\ - \end{smallmatrix}\right] S \mid \left[\begin{smallmatrix} - \\ - \end{smallmatrix}\right]$$
$$I \to \left[\begin{smallmatrix} d \\ b \end{smallmatrix}\right] I \left[\begin{smallmatrix} - \\ d \end{smallmatrix}\right] \mid \left[\begin{smallmatrix} - \\ c \end{smallmatrix}\right] I \mid \left[\begin{smallmatrix} - \\ - \end{smallmatrix}\right]$$

Acknowledgments

I wish to thank D. Gardy and J.-M. Steyaert for useful suggestions and comments. I am grateful to L. Grate for data on UCSC's SCFG experiments and F. Corpet for providing me with the source of the RNAlign tool.

References

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

Bafna, V.; Muthukrishnan, S.; and Ravi, R. 1995. Computing similarity between RNA strings. In *Proceedings of the Sixth Symposium on Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, 1–16. Springer-Verlag.

Baldi, P.; Chauvin, Y.; Hunkapillar, T.; and McClure, M. 1994. Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences of the USA* 91:1059–1063.

Brown, M. 1995. RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. Available by anonymous ftp at ftp.cse.ucsc.edu in /pub/rna. To be presented at Pacific Symposium on Biocomputing Jan. 1996.

Cary, R. B., and Stormo, G. D. 1995. Graph-theoretic approach to RNA modeling using comparative data. In *Third International Conference on Intelligent Systems for Molecular Biology*, 75–80. AAAI Press.

Chevalet, C., and Michot, B. 1992. An algorithm for comparing RNA secondary structures and searching for similar substructures. *Computing Applications in the Biosciences* 8(3):215–225.

Corpet, F., and Michot, B. 1994. RNAlign program: alignment of RNA sequences using both primary and secondary structures. *Computing Applications in the Biosciences* 10(4):389–399.

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

Grate, L. 1995. Automatic RNA secondary structure determination with stochastic context-free grammars. In *Third International Conference on Intelligent Systems for Molecular Biology*, 136–144. AAAI Press.

Han, K., and Kim, H. 1993. Prediction of common folding structures of homologous RNAs. *Nucleic Acids Research* 21:1251–1257.

Hoffmann, C. M., and O'Donnel, M. J. 1982. Pattern matching in trees. *Journal of the ACM* 29(1):68–95.

Jiang, T.; Wang, L.; and Zhang, K. 1994. Alignment of trees - an alternative to tree edit. In *Proceedings of the Fifth Symposium on Combinatorial Pattern Matching*, volume 807 of *Lecture Notes in Computer Science*, 75–86. Springer-Verlag.

Krogh, A.; Brown, M.; Mian, I. S.; Sjölander, K.; and Haussler, D. 1994. Hidden markov models in computational biology: Applications to protein modelin. *Journal of Molecular Biology* 235:1501–1531.

Le, S.-Y.; Owens, J.; Nussinov, R.; Chen, J.-H.; Shapiro, B.; and Maizel, J. V. 1989. RNA secondary structures: comparison and determination of frequently recurring substructures by consensus. *Computing Applications in the Biosciences* 5(3):205–210.

Lefebvre, F. 1995. An optimized parsing algorithm well suited to RNA folding. In *Third International Conference on Intelligent Systems for Molecular Biology*, 222–230. AAAI Press.

Margalit, H.; Shapiro, A. B.; Oppenheim, A. B.; and Maizel, J. V. J. 1989. Detection of common motifs in RNA secondary structure. *Nucleic Acids Research* 17:4829–4845.

Myers, E. W. 1991. An overview of sequence comparison algorithms in molecular biology. Technical Report 29, University of Arizona. Available by anonymous ftp at ftp.cs.arizona.edu in /people/gene/PAPERS.

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. *Nucleic Acids Research* 22:5112–5120.

Salomaa, A. 1973. *Formal Languages*. Academic Press, Inc.

Sankoff, D. 1985. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics* 45(5):810–825.

Searls, D. B., and Murphy, K. P. 1995. Automata - theoretic models of mutation and alignment. In *Third International Conference on Intelligent Systems for Molecular Biology*, 341–349. AAAI Press.

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

Shapiro, B. A., and Zhang, K. 1990. Computing multiple RNA secondary structures using tree comparisons. *Computing Applications in the Biosciences* 6(4):309–318.

Shapiro, B. A. 1988. An algorithm for comparing multiple RNA secondary structures. *Computing Applications in the Biosciences* 4(3):387–393.

Steinberg, S.; Misch, A.; and Sprinzl, M. 1993. Compilation of tRNA sequences and sequences of tRNA genes. *Nucleic Acids Research* 21:3011–3015.

Thompson, J.; Higgins, D. G.; and Gibson, T. J. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research* 22:4673–4680.

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.

Vauchaussade de Chaumont, M. 1985. Nombre de strahler des arbres, langages algébriques et dénombrement de structures secondaires en biologie moléculaire. Master's thesis, Universit'e de Bordeaux I.

Zuker, M. 1989. On finding all suboptimal foldings of an RNA molecule. *Science* 244(7):48–52.