An Efficient Parser Well Suited to RNA Folding

Fabrice Lefebvre
LIX, École polytechnique

26 Juin 1995

[summary by Fabrice Lefebvre]

1. Introduction

In RNA, interactions between nucleotides form base pairs and, seen at a higher level, characteristic secondary structure motifs such as helices, loops and bulges. These motifs are of great interest to biologists. Though the secondary structure of RNA is much simpler than its tertiary structure, it remains difficult to compute because the number of secondary structures of an RNA of n bases grows exponentially with n [9]. Several methods have been established for folding RNAs, that is predicting RNA secondary structure. The first method is phylogenetic analysis of homologous RNA molecules. It relies on conservation of structural features during evolution. Some people are trying to apply a grammar formalism to this method [3]. The second method uses a simplified thermodynamic model of RNA secondary structure to find the structure with the lowest free energy. The third method has been recently introduced by Haussler *et al.* [6] and it relies on *stochastic context-free grammars* (SCFGs) to model common secondary structures of a given family of RNAs. Our parser has been designed to express easily the latest two methods.

2. Folding and S-attribute grammars

It is well known that secondary structures without pseudo-knots of an RNA may be seen as derivation trees of this RNA for a suitably defined context-free grammar (CFG) [7]. We might for instance use the following grammar with terminals A, C, G, U:

$$E \to \epsilon \mid AE \mid CE \mid GE \mid UE \mid AEUE \mid UEAE \mid GECE \mid GEUE \mid CEGE \mid UEGE$$

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

Definition 1. A CFG $G = (T, N, P, S)$ consists of finite sets of terminals T, nonterminals N, productions (rewriting rules) P and of a start symbol $S \in N$. Let $V = N \cup T$ denote the vocabulary of the grammar. Each production in P has the form $A \to \alpha$, where $A \in N$ and $\alpha \in V^*$. A is the left-hand side of the production and α its right-hand side.

In our work, we assumed that the grammar is proper (no useless rules or symbols, non-circular, epsilon-free). Grammars whose derivation trees describe secondary structures will always be ambiguous because a given RNA always has many different secondary structures.

CFGs allow us to give a synthetic description of a set of secondary structures, but they do not allow us to choose one structure among this set. S-attribute CFGs (S-ACFGs) [4] are an extension of CFGs 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 may be more complex. Our context of utilization of S-ACFGs has led us to the following definition for those grammars.

DEFINITION 2. An S-ACFG is denoted by $G = (T, N, P, S, \mathcal{A}, S_{\mathcal{A}}, F_P)$. This is an extension of the proper CFG $G = (T, N, P, S)$, 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 $f_{A\to\alpha}$ is in F_P iff $A \to \alpha$ is in P.

The attribute λ of a string α is the concatenation of the attributes of the symbols in α. When a function $f_{A\to\alpha}$ 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 $\alpha A\beta \to^* u$, where u must belong to T^* in order that the attribute of A may be computable.

3. Two known parsing algorithms

With our application of parsing to RNAs, we will have to find one derivation tree among a potentially exponential number of derivation trees. Hence tabular algorithms are a good way to deal with this parse forest since they output a compacted representation of the parse forest in polynomial time $O(n^3)$ and space $O(n^2)$.

The simplest algorithm is the one of Cocke-Younger-Kasami [1]:
Let G be a proper CFG in Chomsky normal form. The algorithm builds a table $(T_j)_{1\le j\le n}$ such that T_j contains the item $[A, i]$ iff $A \to a_{i+1} \dots a_j$.

For j between 1 and n, perform the following steps

(1) Add $[A, j-1]$ to T_j if $A \to a_j$;
(2) Add $[A, i]$ to T_j if $\exists k < j$ such that $[B, i] \in T_k$ and $[C, k] \in T_j$ and $A \to BC$;
(3) Repeat the previous step while there remains items to be added to T_j.

The string $a_1 \dots a_n$ belongs to $L(G)$ iff $[S, 0] \in T_n$.

CYK's algorithm needs grammars in Chomsky normal form, and it does not avoid many useless derivation subtrees. A much better algorithm is Earley's algorithm [1, 2]:
Let G be a proper CFG. Objects of the form $[A \to X_1 \dots X_k \cdot X_{k+1} \dots X_m, i]$, where $A \to X_1 \dots X_m$ and $0 \le i \le n$, are called items. The algorithm builds a table $(T_j)_{1\le j\le n}$ such that T_j contains an item $[A \to \alpha \cdot \beta, i]$ iff there exists γ such that

$$S \to^* a_1 \dots a_i A\gamma \to a_1 \dots a_i\alpha\beta\gamma \to^* a_1 \dots a_j\beta\gamma$$

At the beginning, $[S \to \cdot\alpha, 0] \in T_0$ for all $S \to \alpha$. Then, if $[A \to \cdot B\beta, 0] \in T_0$, add $[B \to \cdot\gamma, 0]$ to T_0 for all $B \to \gamma$. For j between 1 and n, perform the following steps

(1) For all $[B \to \alpha \cdot a\beta, i] \in T_{j-1}$ such that $a = a_j$, add $[B \to \alpha a \cdot \beta, i]$ to T_j;
(2) If $[A \to \gamma\cdot, i] \in T_j$, then for all $[B \to \alpha \cdot A\beta, k] \in T_i$, add $[B \to \alpha A \cdot \beta, k]$ to T_j;
(3) If $[A \to \alpha \cdot B\beta, i] \in T_j$, add $[B \to \cdot\gamma, j]$ to T_j for all $B \to \gamma$;
(4) Repeat the two previous steps while there remains items to be added to T_j.

The string $a_1 \dots a_n$ belongs to $L(G)$ iff $[S \to \alpha\cdot, 0] \in T_n$.

All items generated by Earley's algorithm are useful in the context of left to right parse of the input string.

4. Our parsing algorithm

Our parsing algorithm outputs items which are in fact a factorization of Earley's items sharing the same right part before the dot: it will replace a set of items $[A \to \alpha \cdot \beta, i]$ having the same string α by a single item $[\Delta \to \alpha, i]$ if $\alpha \neq \epsilon$, or by nothing if $\alpha = \epsilon$ [8, 5]. Δ is the set of non-terminals which were at the left-hand side of replaced items.

The algorithm replaces the search performed in step 4 of Earley's algorithm by an extraction in a priority queue Q holding pairs (X, i) of symbols and integers. We say that a pair (X, i) has a greater priority than a pair (Y, j) if $i > j$ or if $i = j$ and $Y \to^* X$. The function used to return and remove the set of maximum pairs of Q is denoted by $Extract$.

Let G be a proper CFG. Our algorithm builds a table $(T_j)_{1 \leq j \leq n}$ such that T_j contains an item $[\Delta \to \alpha, i]$ iff $\alpha \neq \epsilon$ and for all $A \in \Delta$ there exists β and γ such that

$$S \to^* a_1 \dots a_i A \gamma \to a_1 \dots a_i \alpha \beta \gamma \to^* a_1 \dots a_j \beta \gamma$$

Every time an item $[\Delta \to \alpha, i]$ is added to T_j, perform $Q := Q \cup \{(A, i) \mid A \in \Delta \wedge A \to \alpha\}$. At the beginning, all T_j are empty. Let $\Delta_0 = \{A \in N \mid \exists \beta, S \to A\beta\}$. For j between 1 and n perform the following steps

(1) $Q := \{(a_j, j-1)\}$;
(2) $(X, i) := Extract(Q)$;
(3) $T_j := T_j \cup \{[\Delta \to X, i] \mid \Delta = \{A \in \Delta_i \mid \exists \beta, A \to X\beta\} \neq \emptyset\}$;
(4) $T_j := T_j \cup \{[\Delta \to \alpha X, h] \mid \exists [\Delta' \to \alpha, h] \in T_i, \Delta = \{A \in \Delta' \mid \exists \beta, A \to \alpha X \beta\} \neq \emptyset\}$;
(5) Repeat steps 2 to 5 while Q is not empty;
(6) Compute $\Delta_j := \bigcup_{[\Delta \to \alpha, i] \in T_j} \{D \in N \mid \exists A \to \alpha B \beta, \exists \gamma, A \in \Delta \wedge B \to^* D\gamma\}$.

Then $a_1 \dots a_n \in L(G)$ iff there exists $[\Delta \to \alpha, 0] \in T_n$ such that $S \in \Delta$ and $S \to \alpha$.

In the general case, the complexity of our algorithm is $O(n^3)$ in time and $O(n^2)$ in space. As with Earley's algorithm, these orders might be improved for grammars having some special properties which are of no interest in our case (RNA folding).

Now that we have an algorithm which may use CFG, we may transform it to use S-ACFG:

- Items are $[\Delta \to \alpha, i, \lambda]$, where λ is the string of attributes attached to α;
- Pairs (X, i) added to Q are triplets (X, i, x), where x is the attribute attached to X;
- Functions $f_{A \to \alpha}$ are taken into account at the time of reduction of items;
- The combinatorial explosion of the number of items is avoided with constraints $\mathcal{C}_A$, associated with non-terminals A, which replace a set of triplets (A, i, x) with fixed A and i by a single triplet (A, i, y) whose attribute y is deduced from attributes in the replaced set.

Let G be a S-ACFG. Every time an item $[\Delta \to \alpha, i, \lambda]$ is added to T_j, perform $Q := Q \cup \{(A, i, f_{A \to \alpha}(\lambda)) \mid A \in \Delta \wedge A \to \alpha\}$. At the beginning, all T_j are empty. Let $\Delta_0 = \{A \in N \mid \exists \beta, S \to A\beta\}$. For $1 \leq j \leq n$ perform the following steps

(1) $Q := \{(a_j, j-1, S_{\mathcal{A}}(a_j))\}$;
(2) $(X, i, x) := \mathcal{C}_X(Extract(Q))$;
(3) $T_j := T_j \cup \{[\Delta \to X, i, x] \mid \Delta = \{A \in \Delta_i \mid \exists \beta, A \to X\beta\} \neq \emptyset\}$;
(4) $T_j := T_j \cup \{[\Delta \to \alpha X, h, \lambda x] \mid \exists [\Delta' \to \alpha, h, \lambda] \in T_i, \Delta = \{A \in \Delta' \mid \exists \beta, A \to \alpha X \beta\} \neq \emptyset\}$;
(5) Repeat steps 2 to 5 while Q is not empty;
(6) Compute $\Delta_j := \bigcup_{[\Delta \to \alpha, i, \lambda] \in T_j} \{D \in N \mid \exists A \to \alpha B \beta, \exists \gamma, A \in \Delta \wedge B \to^* D\gamma\}$.

Then $a_1 \dots a_n \in L(G)$ iff there exists $[\Delta \to \alpha, 0, \lambda] \in T_n$ such that $S \in \Delta$ and $S \to \alpha$.

Let $r \geq 1$ be the maximum number of nonterminals appearing at the right-hand side of any production of G. Then the space complexity is $O(n^r)$ and the time complexity is $O(n^{r+1})$. Grammars

used in practice usually verify $r = 2$ or may be turned into a grammar verifying $r = 2$. Hence our algorithm has the complexity of the dynamic programming algorithm used by Zuker [10] to find a secondary structure of minimal energy with the thermodynamic model.

The main advantage of our algorithm over a simpler Earley algorithm is that, with the factorization provided by our items, many different items may now be replaced by a single item. This feature is interesting with SCFGs we used with our algorithm.

5. Results

We have retrieved by ftp the *Vienna package*. This package is a set of C source files which implements the old style dynamic programming relations popularized by Zuker to find the minimum energy secondary structure of an RNA for the well known thermodynamic model. We then converted the thermodynamic model embedded in this package into a suitable S-ACFG, and then into a C source parser by a YACC-like tool which we wrote. Because they use the same model, our generated parser and the *Vienna package* will return the same secondary structure from the same input, thus we may compare dynamic programming and parsing. On 1667 bases of RNA on a DEC-server 2100-500MP, the Vienna package requires 350 s. and 19 Mbytes, while our program needs 266 s. and 50 Mbytes. Thus our parsing algorithm is faster than standard dynamic programming, and it uses less than three times as much memory. Yet, the description of the thermodynamic model with S-ACFG is much simpler and much more flexible in the expression of structural constraints than dynamic programming relations.

Our parsing algorithm may also be readily applied to SCFGs since probabilities of derivations trees may easily be interpreted as attributes of derivation trees. The first results are encouraging because we are 3 times faster on tRNAs than the CYK-like parser used by Haussler's team.

Bibliography

[1] Aho (Alfred V.) and Ullman (Jeffrey D.). – *The Theory of Parsing, Translation, and Compiling.* – Prentice-Hall, Inc., 1972, vol. 1.

[2] Earley (J.). – An efficient context-free parsing algorithm. *Communication of the ACM*, vol. 13, n° 2, February 1970, pp. 94–102.

[3] Eddy (Sean R.) and Durbin (Richard). – RNA sequence analysis using covariance models. *Nucleic Acids Research*, vol. 22, n° 11, 1994, pp. 2079–2088.

[4] Knuth (Donald E.). – Semantics of context-free languages. *Mathematical Systems Theory*, vol. 2, 1968, pp. 127–145. – Correction: *Mathematical Systems Theory* vol. 5, 1971, pp. 95-96.

[5] Nederhof (Mark-Jan). – A multidisciplinary view on PLR parsing. In *Twentieth Workshop on Language Technology 6.* – December 1993.

[6] Sakakibara (Yasubumi), Brown (Michael), Mian (I. Saira), Sjölander (Kimmen), Underwood (Rebecca C.), and Haussler (David). – *The application of stochastic context-free grammars to folding, aligning and modeling homologous RNA sequences.* – Technical Report n° UCSC-CRL-94-14, University of California, Santa Cruz, Santa Cruz CA 95064 USA, 1993.

[7] Searls (David B.). – The linguistics of DNA. *American Scientist*, vol. 80, 1992, pp. 579–591.

[8] Voisin (F.). – A botttom-up adaptation of Earley's parsing algorithm. In *Programming Languages Implementation and Logic Programming, International Workshop*, pp. 146–160. – Springer-Verlag, May 1988.

[9] Waterman (M. S.). – Secondary structure of single-stranded nucleic acids. *Studies in Foundations and Combinatorics, Advances in Mathematics Supplementary Studies*, vol. 1, 1978, p. 167.

[10] Zuker (Michael). – The use of dynamic programming algorithms in RNA secondary structure prediction. In Waterman (Michael S.) (editor), *Mathematical Methods for DNA Sequences*, Chapter 7, pp. 159–184. – CRC Press, 1989.