Constructing parse forests that include exactly the n-best PCFG trees

Pierre Boullier[1], **Alexis Nasr**[2] **and Benoît Sagot**[1]
1. Alpage, INRIA Paris-Rocquencourt & Université Paris 7
Domaine de Voluceau — Rocquencourt, BP 105 — 78153 Le Chesnay Cedex, France
{Pierre.Boullier,Benoit.Sagot}@inria.fr
2. LIF, Univ. de la Méditerrannée
163, avenue de Luminy - Case 901 — 13288 Marseille Cedex 9, France
Alexis.Nasr@lif.univ-mrs.fr

Abstract

This paper describes and compares two algorithms that take as input a shared PCFG parse forest and produce shared forests that contain exactly the n most likely trees of the initial forest. Such forests are suitable for subsequent processing, such as (some types of) reranking or LFG f-structure computation, that can be performed ontop of a shared forest, but that may have a high (e.g., exponential) complexity w.r.t. the number of trees contained in the forest. We evaluate the performances of both algorithms on real-scale NLP forests generated with a PCFG extracted from the Penn Treebank.

1 Introduction

The output of a CFG parser based on dynamic programming, such as an Earley parser (Earley, 1970), is a compact representation of all syntactic parses of the parsed sentence, called a *shared parse forest* (Lang, 1974; Lang, 1994). It can represent an exponential number of parses (with respect to the length of the sentence) in a cubic size structure. This forest can be used for further processing, as reranking (Huang, 2008) or machine translation (Mi et al., 2008).

When a CFG is associated with probabilistic information, as in a Probabilistic CFG (PCFG), it can be interesting to process only the n most likely trees of the forest. Standard state-of-the-art algorithms that extract the n best parses (Huang and Chiang, 2005) produce a collection of trees, losing the factorization that has been achieved by the parser, and reproduce some identical sub-trees in several parses.

This situation is not satisfactory since post-parsing processes, such as reranking algorithms or attribute computation, cannot take advantage of this lost factorization and may reproduce some identical work on common sub-trees, with a computational cost that can be exponentally high.

One way to solve the problem is to prune the forest by eliminating sub-forests that do not contribute to any of the n most likely trees. But this over-generates: the pruned forest contains more than the n most likely trees. This is particularly costly for post-parsing processes that may require in the worst cases an exponential execution time w.r.t. the number of trees in the forest, such as LFG f-structures construction or some advanced reranking techniques. The experiments detailed in the last part of this paper show that the over-generation factor of pruned sub-forest is more or less constant (see 6): after pruning the forest so as to keep the n best trees, the resulting forest contains approximately $10^3 n$ trees. At least for some post-parsing processes, this overhead is highly problematic. For example, although LFG parsing can be achieved by computing LFG f-structures on top of a c-structure parse forest with a reasonable efficiency (Boullier and Sagot, 2005), it is clear that a 10^3 factor drastically affects the overall speed of the LFG parser.

Therefore, simply pruning the forest is not an adequate solution. However, it will prove useful for comparison purposes.

The new direction that we explore in this paper is the production of shared forests that contain *exactly* the n most likely trees, avoiding both the explicit construction of n different trees and the over-generation of pruning techniques. This can be seen as a transduction which is applied on

a forest and produces another forest. The transduction applies some local transformations on the structure of the forest, developing some parts of the forest when necessary.

The structure of this paper is the following. Section 2 defines the basic objects we will be dealing with. Section 3 describes how to prune a shared forest, and introduces two approaches for building shared forests that contain exactly the n most likely parses. Section 4 describes experiments that were carried out on the Penn Treebank and section 5 concludes the paper.

2 Preliminaries

2.1 Instantiated grammars

Let $G = \langle \mathcal{N}, \mathcal{T}, \mathcal{P}, S \rangle$ be a context-free grammar (CFG), defined in the usual way (Aho and Ullman, 1972). Throughout this paper, we suppose that we manipulate only non-cyclic CFGs,[1] but they may (and usually do) include ε-productions. Given a production $p \in \mathcal{P}$, we note $\mathrm{lhs}(p)$ its left-hand side, $\mathrm{rhs}(p)$ its right-hand side and $|p|$ the length of $\mathrm{rhs}(p)$. Moreover, we note $\mathrm{rhs}_k(p)$, with $1 \leq k \leq |p|$, the k^{th} symbol of $\mathrm{rhs}(p)$. We call A-production any production $p \in \mathcal{P}$ of G such that $\mathrm{lhs}(p) = A$.

A complete derivation of a sentence $w = t_1 \dots t_{|w|}$ ($\forall i \leq |w|, t_i \in \mathcal{T}$) w.r.t. G is of the form $S \underset{G,w}{\overset{*}{\Rightarrow}} \alpha A \beta \underset{G,w}{\Rightarrow} \alpha X^1 X^2 \dots X^r \beta \underset{G,w}{\overset{*}{\Rightarrow}} w$. By definition, $A \rightarrow X^1 X^2 \dots X^r$ is a production of G. Each of A, X^1, X^2, …, X^r spans a unique occurrence of a substring $t_{i+1} \dots t_j$ of w, that can be identified by the corresponding *range*, noted $i..j$. A complete derivation represents a *parse tree* whose yield is w, in which each symbol X of range $i..j$ roots a subtree whose yield is $t_{i+1} \dots t_j$ (i.e., a derivation of the form $X \underset{G,w}{\overset{*}{\Rightarrow}} t_{i+1} \dots t_j$).

Let us define the w-*instantiation* operation (or *instantiation*). It can be applied to symbols and productions of G, and to G itself, w.r.t. a string w. It corresponds to the well-known intersection of G with the linear automaton that corresponds to the string w. We shall go into further detail for terminology, notation and illustration purposes.

[1] Actually, cyclic CFG can be treated as well, but not cyclic parse forests. Therefore, if using a cyclic CFG which, on a particular sentence, builds a cyclic parse forest, cycles have to be removed before the algorithms descibed in the next sections are applied. This is the case in the SYNTAX system (see below).

An *instantiated non terminal symbol* is a triple noted $A_{i..j}$ where $A \in \mathcal{N}$ and $0 \leq i \leq j \leq |w|$. Similarly, an *instantiated terminal symbol* is a triple noted $T_{i..j}$ where $T \in \mathcal{T}$ and $0 \leq i \leq j = i+1 \leq |w|$. An *instantiated symbol*, terminal or non terminal, is noted $X_{i..j}$. For any instantiated symbol $X_{i..j}$, i (resp. j) is called its *lower bound* (resp. *upper bound*), and can be extracted by the operator $\mathrm{lb}()$ (resp. $\mathrm{ub}()$).

An *instantiated production* (or *instantiated rule*) is a context-free production $A_{i..j} \rightarrow X^1_{i_1..j_1} X^2_{i_2..j_2} \dots X^r_{i_r..j_r}$ whose left-hand side is an instantiated non terminal symbol and whose right-hand side is a (possibly empty) sequence of instantiated (terminal or non terminal) symbols, provided the followings conditions hold:

1. the indexes involved are such that $i = i_1$, $j = j_r$, and $\forall l$ such that $1 \leq l < r$, $j_l = i_{l+1}$;
2. the corresponding non-instantiated production $A \rightarrow X^1 X^2 \dots X^r$ is a production of G.

If $\mathrm{lhs}(p) = A_{i..j}$, we set $\mathrm{lb}(p) = i$ and $\mathrm{ub}(p) = j$.

In a complete derivation $S \underset{G,w}{\overset{*}{\Rightarrow}} \alpha A \beta \underset{G,w}{\Rightarrow} \alpha X^1 X^2 \dots X^r \beta \underset{G,w}{\overset{*}{\Rightarrow}} w$, any symbol X that spans the range $i..j$ can be replaced by the instantiated symbols $X_{i..j}$. For example, the axiom S can be replaced by the instantiated axiom $S_{0..|w|}$ in the head of the derivation. If applied to the whole derivation, this operation creates an *instantiated derivation*, whose rewriting operations define a particular set of instantiated productions. Given G and w, the set of all instantiated productions involved in at least one complete derivation of w is unique, and noted $\mathcal{P}_w$. An instantiated derivation represents an *instantiated parse tree*, i.e., a parse tree whose node labels are instantiated symbols. In an instantiated parse tree, each node label is unique, and therefore we shall not distinguish between a node in an instantiated parse tree and its label (i.e., an instantiated symbol).

Then, the w-*instantiated grammar* G_w for G and w is a CFG $\langle \mathcal{N}_w, \mathcal{T}_w, \mathcal{P}_w, S_{0..|w|} \rangle$ such that:

1. $\mathcal{P}_w$ is defined as explained above;
2. $\mathcal{N}_w$ is a set of instantiated non terminal symbols;
3. $\mathcal{T}_w$ is a set of instantiated terminal symbols.

It follows from the definition of $\mathcal{P}_w$ that (instantiated) symbols of G_w have the following properties: $A_{i..j} \in \mathcal{N}_w \Leftrightarrow A \overset{*}{\underset{G,w}{\Rightarrow}} t_{i+1} \dots t_j$, and $T_{i..j} \in \mathcal{T}_w \Leftrightarrow T = t_j$.

The w-instantiated CFG G_w represents *all* parse trees for w in a shared (factorized) way. It is the grammar representation of the parse forest of w w.r.t. G.[2] In fact, $\mathcal{L}(G_w) = \{w\}$ and the set of parses of w with respect to G_w is isomorphic to the set of parses of w with respect to G, the isomorphism being the w-instantiation operation. The *size* of a forest is defined as the size of the grammar that represents it, i.e., as the number of symbol occurrences in this grammar, which is defined as the number of productions plus the sum of the lengths of all right-hand sides.

Example 1: First running example.

Let us illustrate these definitions by an example. Given the sentence $w =$ the boy saw a man with a telescope *and the grammar* G *(that the reader has in mind), the instantiated productions of* G_w *are:*

$Det_{0..1} \rightarrow the_{0..1}$	$N_{1..2} \rightarrow boy_{1..2}$
$NP_{0..2} \rightarrow Det_{0..1}\ N_{1..2}$	$V_{2..3} \rightarrow saw_{2..3}$
$Det_{3..4} \rightarrow a_{3..4}$	$N_{4..5} \rightarrow man_{4..5}$
$NP_{3..5} \rightarrow Det_{3..4}\ N_{4..5}$	$Prep_{5..6} \rightarrow with_{5..6}$
$Det_{6..7} \rightarrow a_{6..7}$	$N_{7..8} \rightarrow telescope_{7..8}$
$NP_{6..8} \rightarrow Det_{6..7}\ N_{7..8}$	$PP_{5..8} \rightarrow Prep_{5..6}\ NP_{6..8}$
$NP_{3..8} \rightarrow NP_{3..5}\ PP_{5..8}$	$VP_{2..8} \rightarrow V_{2..3}\ NP_{3..8}$
$VP_{2..5} \rightarrow V_{2..3}\ NP_{3..5}$	$VP_{2..8} \rightarrow VP_{2..5}\ PP_{5..8}$
$S_{0..8} \rightarrow NP_{0..2}\ VP_{2..8}$	

They represent the parse forest of w *according to* G*. This parse forest contains two trees, since there is one ambiguity:* $VP_{2..8}$ *can be rewritten in two different ways.*

The instantiated grammar G_w can be represented as an hypergraph (as in (Klein and Manning, 2001) or (Huang and Chiang, 2005)) where the instantiated symbols of G_w correspond to the vertices of the hypergraph and the instantiated productions to the hyperarcs.

We define the *extension* of an instantiated symbol $X_{i..j}$, noted $\mathcal{E}(X_{i..j})$, as the set of instantiated parse trees that have $X_{i..j}$ as a root. The set of all parse trees of w w.r.t. G is therefore $\mathcal{E}(S_{0..|w|})$. In the same way, we define the extension of an instantiated production $X_{i..j} \rightarrow \alpha$ to be the subset of $\mathcal{E}(X_{i..j})$ that corresponds to derivations of the form $X_{i..j} \underset{G,w}{\Rightarrow} \alpha \overset{*}{\underset{G,w}{\Rightarrow}} t_{i+1} \dots t_j$ (i.e., trees rooted in $X_{i..j}$ and where the daughters of the node $X_{i..j}$ are the symbols of α).

[2] In particular, if G is a binary grammar, its w-instantation (i.e., the parse forest of w) has a size $\mathcal{O}(|w|^3)$, whereas it represents a potentially exponential number of parse trees w.r.t $|w|$ since we manipulate only non-cyclic grammars.

2.2 Forest traversals

Let us suppose that we deal with non-cyclic forests, i.e., we only consider forests that are represented by a non-recursive instantiated CFG. In this case, we can define two different kinds of forest traversals.

A *bottom-up traversal* of a forest is a traversal with the following constraint: an $A_{i..j}$-production is visited if and only if all its instantiated right-hand side symbols have already been visited; the instantiated symbol $A_{i..j}$ is visited once all $A_{i..j}$-productions have been visited. The bottom-up visit starts by visiting all instantiated productions with right-hand sides that are empty or contain only (instantiated) terminal symbols.

A *top-down traversal* of a forest is a traversal with the following constraint: a node $A_{i..j}$ is visited if and only if all the instantiated productions in which it occurs in right-hand side have already been visited; once an instantiated production $A_{i..j}$ has been visited, all its $A_{i..j}$-productions are visited as well. Of course the top-down visit starts by the visit of the axiom $S_{0..|w|}$.

2.3 Ranked instantiated grammar

When an instantiated grammar $G_w = \langle \mathcal{N}_w, \mathcal{T}_w, \mathcal{P}_w, S_{0..|w|} \rangle$ is built on a PCFG, every parse tree in $\mathcal{E}(S_{0..|w|})$ has a probability that is computed in the usual way (Booth, 1969). We might be interested in extracting the k^{th} most likely tree of the forest represented by G_w,[3] without *unfolding* the forest, i.e., without enumerating trees. In order to do so, we need to add some extra structure to the instantiated grammar. The augmented instantiated grammar will be called a *ranked instantiated grammar*.

This extra structure takes the form of n*-best tables* that are associated with each instantiated non terminal symbol (Huang and Chiang, 2005), thus leading to *ranked instantiated non terminal symbols*, or simply *instantiated symbols* when the context is non ambiguous. A ranked instantiated non terminal symbol is written $\langle A_{i..j}, \mathcal{T}(A_{i..j}) \rangle$, where

[3] In this paper, we shall use the k^{th} *most likely tree* and *the tree of rank* k as synonyms.

$\mathcal{T}(A_{i..j})$ is the n-best table associated with the instantiated symbol $A_{i..j}$.

$\mathcal{T}(A_{i..j})$ is a table of at most n entries. The k-th entry of the table, noted e, describes how to build the k-th most likely tree of $\mathcal{E}(A_{i..j})$. This tree will be called the k-th extention of $A_{i..j}$, noted $\mathcal{E}_k(A_{i..j})$. More precisely, e indicates the instantiated $A_{i..j}$-production p such that $\mathcal{E}_k(A_{i..j}) \in \mathcal{E}(p)$. It indicates furthermore which trees of the extensions of p's right-hand side symbols must be combined together in order to build $\mathcal{E}_k(A_{i..j})$.

We also define the m,n-extension of $A_{i..j}$ as follows: $\mathcal{E}_{m,n}(A_{i..j}) = \cup_{m \leq k \leq n} \mathcal{E}_k(A_{i..j})$.

Example 2: n-best tables for the first running example.

Let us illustrate this idea on our first running example. Recall that in Example 1, the symbol $VP_{2..8}$ can be rewritten using the two following productions :

$$VP_{2..8} \rightarrow V_{2..3}\ NP_{3..8}$$
$$VP_{2..8} \rightarrow VP_{2..5}\ PP_{5..8}$$

$\mathcal{T}(VP_{2..8})$ has the following form:

1	P_1	$VP_{2..8} \rightarrow V_{2..3}\ NP_{3..8}$	$\langle 1,1 \rangle$	1
2	P_2	$VP_{2..8} \rightarrow VP_{2..5}\ PP_{5..8}$	$\langle 1,1 \rangle$	1

This table indicates that the most likely tree associated with $VP_{2..8}$ (line one) has probability P_1 and is built using the production $VP_{2..8} \rightarrow V_{2..3}\ NP_{3..8}$ by combining the most likely tree of $\mathcal{E}(V_{2..3})$ (indicated by the first 1 in $\langle 1,1 \rangle$) with the most likely tree of $\mathcal{E}(NP_{3..8})$ (indicated by the second 1 in $\langle 1,1 \rangle$). It also indicates that the most likely tree of $\mathcal{E}(VP_{2..8})$ is the most likely tree of $\mathcal{E}(VP_{2..8} \rightarrow V_{2..3}\ NP_{3..8})$ (indicated by the presence of 1 in the last column of entry 1) and the second most likely tree of $\mathcal{E}(VP_{2..8})$ is the most likely tree of $\mathcal{E}(VP_{2..8} \rightarrow VP_{2..5}\ PP_{5..8})$. This last integer is called the local rank of the entry.

More formally, the entry $\mathcal{T}(A_{i..j})[k]$ is defined as a 4-tuple $\langle P_k, p_k, \vec{v_k}, l_k \rangle$ where k is the rank of the entry, P_k is the probability of the tree $\mathcal{E}_k(A_{i..j})$, p_k is the instantiated production such that $\mathcal{E}_k(A_{i..j}) \in \mathcal{E}(p_k)$, $\vec{v_k}$ is a tuple of $|\mathrm{rhs}(p_k)|$ integers and l_k is the local rank.

The tree $\mathcal{E}_k(A_{i..j})$ is rooted by $A_{i..j}$, and its daughters root $N = |\mathrm{rhs}(p_k)|$ subtrees that are $\mathcal{E}_{\vec{v_k}[1]}(\mathrm{rhs}_1(p_k)), \ldots, \mathcal{E}_{\vec{v_k}[N]}(\mathrm{rhs}_N(p_k))$.

Given an instantiated symbol $A_{i..j}$ and an instantitated production $p \in P(A_{i..j})$, we define the n-best table of p to be the table composed of the entries $\langle P_k, p_k, \vec{v_k}, l_k \rangle$ of $\mathcal{T}(A_{i..j})$ such that $p_k = p$.

Example 3: Second running example.

The following is a standard PCFG (probabilities are shown next to the corresponding clauses).

$$S \rightarrow A\ B \quad 1$$
$$A \rightarrow A1 \quad 0.7 \qquad A1 \rightarrow a \quad 1$$
$$A \rightarrow A2 \quad 0.3 \qquad A2 \rightarrow a \quad 1$$
$$B \rightarrow B1 \quad 0.6 \qquad B1 \rightarrow b \quad 1$$
$$B \rightarrow B2 \quad 0.4 \qquad B2 \rightarrow b \quad 1$$

The instantiation of the underlying (non-probabilistic) CFG grammar by the input text $w = a\ b$ is the following.

$$S_{1..3} \rightarrow A_{1..2}\ B_{2..3}$$
$$A_{1..2} \rightarrow A1_{1..2} \qquad A1_{1..2} \rightarrow a_{1..2}$$
$$A_{1..2} \rightarrow A2_{1..2} \qquad A2_{1..2} \rightarrow a_{1..2}$$
$$B_{2..3} \rightarrow B1_{2..3} \qquad B1_{2..3} \rightarrow b_{2..3}$$
$$B_{2..3} \rightarrow B2_{2..3} \qquad B2_{2..3} \rightarrow b_{2..3}$$

This grammar represents a parse forest that contains four different trees, since on the one hand one can reach (parse) the instantiated terminal symbol $a_{1..2}$ through $A1$ or $A2$, and on the other hand one can reach (parse) the instantiated terminal symbol $b_{1..2}$ through $B1$ or $B2$. Therefore, when discussing this example in the remainder of the paper, each of these four trees will be named accordingly: the tree obtained by reaching a through Ai and b through Bj (i and j are 1 or 2) shall be called $T_{i,j}$.

The corresponding n-best tables are trivial (only one line) for all instantiated symbols but $A_{1..2}$, $B_{2..3}$ and $S_{1..3}$. That of $A_{1..2}$ is the following 2-line table.

1	0.7	$A \rightarrow A1$	$\langle 1 \rangle$	1
2	0.3	$A \rightarrow A2$	$\langle 1 \rangle$	1

The n-best table for $B_{2..3}$ is similar. The n-best table for $S_{1..3}$ is:

1	0.42	$S_{1..3} \rightarrow A_{1..2}\ B_{2..3}$	$\langle 1,1 \rangle$	1
2	0.28	$S_{1..3} \rightarrow A_{1..2}\ B_{2..3}$	$\langle 1,2 \rangle$	2
3	0.18	$S_{1..3} \rightarrow A_{1..2}\ B_{2..3}$	$\langle 2,1 \rangle$	3
4	0.12	$S_{1..3} \rightarrow A_{1..2}\ B_{2..3}$	$\langle 2,2 \rangle$	4

Thanks to the algorithm sketched in section 2.4, these tables allow to compute the following obvious result: the best tree is $T_{1,1}$, the second-best tree is $T_{1,2}$, the third-best tree is $T_{2,1}$ and the worst tree is $T_{2,2}$.

If $n = 3$, the pruned forest over-generates: all instantiated productions take part in at least one of the three best trees, and therefore the pruned

forest is the full forest itself, which contains four trees.

We shall use this example later on so as to illustrate both methods we introduce for building forests that contain exactly the n best trees, without overgenerating.

2.4 Extracting the k^{th}-best tree

An efficient algorithm for the extraction of the n-best trees is introduced in (Huang and Chiang, 2005), namely the authors' algorithm 3, which is a re-formulation of a procedure originally proposed by (Jiménez and Marzal, 2000). Contrarily to (Huang and Chiang, 2005), we shall sketch this algorithm with the terminology introduced above (whereas the authors use the notion of hypergraph). The algorithm relies on the n-best tables described above: extracting the k^{th}-best tree consists in extending the n-best tables as much as necessary by computing all lines in each n-best table up to those that concern the k^{th}-best tree.[4]

The algorithm can be divided in two sub-algorithms: (1) a bottom-up traversal of the forest for extracting the best tree; (2) a top-down traversal for extracting the k^{th}-best tree provided the $(k-1)^{\text{th}}$-best has been already extracted.

The extraction of the best tree can be seen as a bottom-up traversal that initializes the n-best tables: when visiting a node $A_{i..j}$, the best probability of each $A_{i..j}$-production is computed by using the tables associated with each of their right-hand side symbols. The best of these probabilities gives the first line of the n-best table for $A_{i..j}$ (the result for other productions are stored for possible later use). Once the traversal is completed (the instantiated axiom has been reached), the best tree can be easily output by following recursively where the first line of the axiom's n-best table leads to.

Let us now assume we have extracted all k'-best trees, $1 \leq k' < k$, for a given $k \leq n$. We want to extract the k^{th}-best tree. We achieve this recursively by a top-down traversal of the forest. In order to start the construction of the k^{th}-best tree, we need to know the following:

- which instantiated production p must be used for rewriting the instantiated axiom,
- for each of p's right-hand side symbols $A_{i..j}$, which subtree rooted in $A_{i..j}$ must be used; this subtree is identified by its *local rank* $k_{A_{i..j}}$, i.e., the rank of its probability among all subtrees rooted in $A_{i..j}$.

This information is given by the k^{th} line of the n-best table associated with the instantiated axiom. If this k^{th} line has not been filled yet, it is computed recursively.[5] Once the k^{th} line of the n-best table is known, i.e., p and all $k_{A_{i..j}}$'s are known, the rank k is added to p's so-called *rankset*, noted $\rho(p)$. Then, the top-down traversal extracts recursively for each $A_{i..j}$ the appropriate subtree as defined by $k_{A_{i..j}}$. After having extracted the n-th best tree, we know that a given production p is included in the k^{th}-best tree, $1 \leq k \leq n$, if and only if $k \in \rho(p)$.

3 Computing sub-forests that only contain the n best trees

Given a ranked instantiated grammar G_w, we are interested in building a new instantiated grammar which contains exactly the n most likely trees of $\mathcal{E}(G_w)$. In this section, we introduce two algorithms that compute such a grammar (or forest). Both methods rely on the construction of new symbols, obtained by decorating instantiated symbols of G_w.

An empirical comparison of the two methods is described in section 4. In order to evaluate the size of the new constructed grammars (forests), we consider as a lower bound the so-called *pruned forest*, which is the smallest sub-grammar of the initial instantiated grammar that includes the n best trees. It is built simply by pruning productions with an empty rankset: no new symbols are created, original instantiated symbols are kept. Therefore, it is a lower bound in terms of size. However, the pruned forest usually overgenerates, as illustrated by Example 3.

[4] In the remainder of this paper, we shall use "extracting the k^{th}-best tree" as a shortcut for "extending the n-best tables up to what is necessary to extract the k^{th}-best tree" (i.e., we do not necessarily really build or print the k^{th}-best tree).

[5] Because the $k-1^{\text{th}}$-best tree has been computed, this n-best table is filled exactly up to line $k-1$. The k^{th} line is then computed as follows: while constructing the k'^{th}-best trees for each k' between 1 and $k-1$, we have identified many possible rewritings of the instantiated axiom, i.e., many (production, right-hand side local ranks) pairs; we know the probability of all these rewritings, although only some of them consitute a line of the instantiated axiom's n-best table; we now identify new rewritings, starting from known rewritings and incrementing only one of their local ranks; we compute (recursively) the probability of these newly identified rewritings; the rewriting that has the best probability among all those that are not yet a line of the n-best table is then added: it is its k^{th} line.

3.1 The ranksets method

The algorithm described in this section builds an instantiated grammar G_w^n by decorating the symbols of G_w. The new (decorated) symbols have the form $A_{i..j}^{\rho}$ where ρ is a set of integers called a *rankset*. An integer r is a *rank* iff we have $1 \leq r \leq n$.

The starting point of this algorithm is set of n-best tables, built as explained in section 2.4, without explicitely unfolding the forest.

A preliminary top-down step uses these n-best tables for building a parse forest whose non-terminal symbols (apart from the axiom) have the form $A_{i..j}^{\rho}$ where ρ is a singleton $\{r\}$: the sub-forest rooted in $A_{i..j}^{\{r\}}$ contains only one tree, that of local rank r. Only the axiom is not decorated, and remains unique. Terminal symbols are not affected either.

At this point, the purpose of the algorithm is to merge productions with identical right-hand sides, whenever possible. This is achieved in a bottom-up fashion as follows. Consider two symbols $A_{i..j}^{\rho_1}$ and $A_{i..j}^{\rho_2}$, which differ only by their underlying ranksets. These symbols correspond to two different production sets, namely the set of all $A_{i..j}^{\rho_1}$-productions (resp. $A_{i..j}^{\rho_2}$-productions). Each of these production sets define a set of right-hand sides. If these two right-hand side sets are identical we say that $A_{i..j}^{\rho_1}$ and $A_{i..j}^{\rho_2}$ are *equivalent*. In that case introduce the rankset $\rho = \rho_1 \cup \rho_2$ and create a new non-terminal symbol $A_{i..j}^{\rho}$. We now simply replace all occurrences of $A_{i..j}^{\rho_1}$ and $A_{i..j}^{\rho_2}$ in left- and right-hand sides by $A_{i..j}^{\rho}$. Of course (newly) identical productions are erased. After such a transformation, the newly created symbol may appear in the right-hand side of productions that now only differ by their left-hand sides; the factorization spreads to this symbol in a bottom-up way. Therefore, we perform this transformation until no new pair of equivalent symbols is found, starting from terminal leaves and percolating bottom-up as far as possible.

Example 4: Applying the ranksets method to the second running example.

Let us come back to the grammar of Example 3, and the same input text $w = a\,b$ as before. As in Example 3, we consider the case when we are interested in the $n = 3$ best trees.

Starting from the instantiated grammar and the n-best tables given in Example 3, the preliminary top-down step builds the following forest (for clarity, ranksets have not been shown on symbols that root sub-forests containing only one tree):

$$S_{1..3} \rightarrow A_{1..2}^{\{1\}}\, B_{2..3}^{\{1\}}$$
$$S_{1..3} \rightarrow A_{1..2}^{\{1\}}\, B_{2..3}^{\{2\}}$$
$$S_{1..3} \rightarrow A_{1..2}^{\{2\}}\, B_{2..3}^{\{1\}}$$
$$A_{1..2}^{\{1\}} \rightarrow A1_{1..2} \qquad A1_{1..2} \rightarrow a_{1..2}$$
$$A_{1..2}^{\{2\}} \rightarrow A2_{1..2} \qquad A2_{1..2} \rightarrow a_{1..2}$$
$$B_{2..3}^{\{1\}} \rightarrow B1_{2..3} \qquad B1_{2..3} \rightarrow b_{2..3}$$
$$B_{2..3}^{\{2\}} \rightarrow B2_{2..3} \qquad B2_{2..3} \rightarrow b_{2..3}$$

In this example, the bottom-up step doesn't factorize out any other symbols, and this is therefore the final output of the ranksets method. It contains 2 more productions and 3 more symbols than the pruned forest (which is the same as the original forest), but it contains exactly the 3 best trees, contrarily to the pruned forest.

3.2 The rectangles method

In this section only, we assume that the grammar G is binary (and therefore the forest, i.e., the grammar G_w, is binary). Standard binarization algorithms can be found in the litterature (Aho and Ullman, 1972).

The algorithm described in this section performs, as the preceding one, a decoration of the symbols of G_w. The new (decorated) symbols have the form $A_{i..j}^{x,y}$, where x and y denote ranks such that $1 \leq x \leq y \leq n$. The semantics of the decoration is closely related to the x, y extention of $A_{i..j}$, introduced in 2.3:

$$\mathcal{E}(A_{i..j}^{x,y}) = \mathcal{E}_{x,y}(A_{i..j})$$

It corresponds to ranksets (in the sense of the previous section) that are intervals: $A_{i..j}^{x,y}$ is equivalent to the previous section's $A_{i..j}^{\{x,x+1,\ldots,y-1,y\}}$. In other words, the sub-forest rooted with $A_{i..j}^{x,y}$ contains exactly the trees of the initial forest, rooted with $A_{i..j}$, which rank range from x to y.

The algorithm performs a top-down traversal of the initial instantiated grammar G_w. This traversal also takes as input two parameters x and y. It starts with the symbol $S_{0..|w|}$ and parameters 1 and n. At the end of the traversal, a new decorated forest is built which contains exactly n most likely the parses. During the traversal, every instantiated symbol $A_{i..j}$ will give birth to decorated instantiated symbols of the form $A_{i..j}^{x,y}$ where x and y are

determined during the traversal. Two different actions are performed depending on whether we are visiting an instantiated symbol or an instantiated production.

3.2.1 Visiting an instantiated symbol

When visiting an instantiated symbol $A_{i..j}$ with parameters x and y, a new decorated instantiated symbol $A_{i,j}^{x,y}$ is created and the traversal continues on the instantiated productions of $P(A_{i..j})$ with parameters that have to be computed. These parameters depend on how the elements of $\mathcal{E}_{x,y}(A_{i..j})$ are "distributed" among the sets $\mathcal{E}(p)$ with $p \in P(A_{i..j})$. In other words, we need to determine x_k's and y_k's such that:

$$\mathcal{E}_{x,y}(A_{i..j}) = \bigcup_{p_k \in P(A_{i..j})} \mathcal{E}_{x_k,y_k}(p_k)$$

The idea can be easily illustrated on an example. Suppose we are visiting the instantiated symbol $A_{i..j}$ with parameters 5 and 10. Suppose also that $A_{i..j}$ can be rewritten using the two instantiated productions p_1 and p_2. Suppose finally that the 5 to 10 entries of $\mathcal{T}(A_{i..j})$ are as follows[6]:

5		p_1		4
6		p_2		2
7		p_2		3
8		p_1		5
9		p_2		4
10		p_1		6

This table says that $\mathcal{E}_5(A_{i..j}) = \mathcal{E}_4(p_1)$ i.e. the 5^{th} most likely analysis of $\mathcal{E}(A_{i..j})$ is the 4^{th} most likely analysis of $\mathcal{E}(p_1)$ and $\mathcal{E}_6(A_{i..j}) = \mathcal{E}_2(p_2)$ and so on. From this table we can deduce that:

$$\mathcal{E}_{5,10}(A_{i..j}) = \mathcal{E}_{4,6}(p_1) \cup \mathcal{E}_{2,4}(p_2)$$

The traversal therefore continues on p_1 and p_2 with parameters $4, 6$ and $2, 4$.

3.2.2 Visiting an instantiated production

When visiting an instantiated production p of the form $A_{i..j} \rightarrow B_{i..l}\ C_{l..j}$ with parameters x and y, a collection of q instantiated productions p_r of the form $A_{i..j}^{x,y} \rightarrow B_{i..l}^{x_r^1,x_r^2}\ C_{l..j}^{y_r^1,y_r^2}$, with $1 \leq r \leq q$, are built, where the parameters $x_r^1, x_r^2, y_r^1, y_r^2$ and q have to be computed.

Once the parameters q and $x_r^1, x_r^2, y_r^1, y_r^2$ with $1 \leq r \leq q$, have been computed, the traversal continues independently on $B_{i..l}$ with parameters x_r^1 and x_r^2 and on $C_{l..j}$ with parameters y_r^1 and y_r^2.

The computation of the four parameters x_r^1, x_r^2, y_r^1 and y_r^2 for $1 \leq r \leq q$, is the most complex part of the algorithm, it relies on the three notions of *rectangles*, *q-partitions* and *n-best matrices*, which are defined below.

Given a 4-tuple of parameters $x_r^1, x_r^2, y_r^1, y_r^2$, a rectangle is simply a pairing of the form $\langle\langle x_r^1, x_r^2\rangle, \langle y_r^1, y_r^2\rangle\rangle$. A rectangle can be interpreted as a couple of rank ranges : $\langle x_r^1, y_r^1\rangle$ and $\langle x_r^2, y_r^2\rangle$. It denotes the cartesian product $[x_r^1, x_r^2] \times [y_r^1, y_r^2]$.

Let $\langle\langle x_1^1, x_1^2\rangle, \langle y_1^1, y_1^2\rangle\rangle, \ldots, \langle\langle x_q^1, x_q^2\rangle, \langle y_q^1, y_q^2\rangle\rangle$ be a collection of q rectangles. It will be called a *q-partition* of the instantiated production p iff the following is true:

$$\mathcal{E}_{x,y}(p) = \bigcup_{1 \leq r \leq q} \mathcal{E}(A_{i..j}^{x,y} \rightarrow B_{i..l}^{x_r^1,x_r^2}\ C_{l..j}^{y_r^1,y_r^2})$$

To put it differently, this definition means that $\langle\langle x_1^1, x_1^2\rangle, \langle y_1^1, y_1^2\rangle\rangle, \ldots, \langle\langle x_q^1, x_q^2\rangle, \langle y_q^1, y_q^2\rangle\rangle$ is a q partition of p if any tree of $\mathcal{E}(B_{i..l}^{x_r^1,x_r^2})$ combined with any tree of $\mathcal{E}(C_{l..j}^{y_r^1,y_r^2})$ is a tree of $\mathcal{E}_{x,y}(p)$ and, conversely, any tree of $\mathcal{E}_{x,y}(p)$ is the combination of a tree of $\mathcal{E}(B_{i..l}^{x_r^1,x_r^2})$ and a tree of $\mathcal{E}(C_{l..j}^{y_r^1,y_r^2})$.

The *n-best matrix* associated with an instantiated production p, introduced in (Huang and Chiang, 2005), is merely a two dimensional representation of the n-best table of p. Such a matrix, represents how the n most likely trees of $\mathcal{E}(p)$ are built. An example of an n-best matrix is represented in figure 1. This matrix says that the first most likely tree of p is built by combining the tree $\mathcal{E}_1(B_{i..l})$ with the tree $\mathcal{E}_1(C_{l..j})$ (there is a 1 in the cell of coordinate $\langle 1, 1\rangle$). The second most likely tree is built by combining the tree $\mathcal{E}_1(B_{i..l})$ and $\mathcal{E}_2(C_{l..j})$ (there is a 2 in the cell of coordinate $\langle 1, 2\rangle$) and so on.

An n-best matrix M has, by construction, the remarkable following properties:

$$M(i, y) < M(x, y)\ \forall i\ 1 \leq i < x$$
$$M(x, j) < M(x, y)\ \forall j\ 1 \leq j < y$$

Given an n-best matrix M of dimensions $d = X \cdot Y$ and two integers x and y such that $1 \leq x < y \leq d$, M can be decomposed into three regions:

- the *lower region*, composed of the cells which contain ranks i with $1 \leq i < x$
- the *intermediate region*, composed of the cells which contain ranks i with $x \leq i \leq y$

[6] Only the relevant part of the table have been kept in the figure.

$B_{i..l}$ \ $C_{l..j}$	1	2	3	4	5	6
1	1	2	6	8	14	15
2	3	5	11	13	18	29
3	4	9	12	17	24	30
4	7	10	20	21	26	33
5	16	19	22	25	27	35
6	23	28	31	32	34	36

Figure 1: n-best matrix

- the *upper region*, composed of the cells which contain ranks i such that $y < i \leq d$.

The three regions of the matrix of figure 1, for $x = 4$ and $y = 27$ have been delimited with bold lines in figure 2.

$B_{i..l}$ \ $C_{l..j}$	1	2	3	4	5	6
1	1	2	6	8	14	15
2	3	5	11	13	18	29
3	4	9	12	17	24	30
4	7	10	20	21	26	33
5	16	19	22	25	27	35
6	23	28	31	32	34	36

Figure 2: Decomposition of an n-best matrix into a lower, an intermediate and an upper region with parameters 4 and 27.

It can be seen that a rectangle, as introduced earlier, defines a *sub-matrix* of the n-best matrix. For example the rectangle $\langle\langle 2, 5\rangle, \langle 2, 5\rangle\rangle$ defines the sub-matrix which north west corner is $M(2, 2)$ and south east corner is $M(5, 5)$, as represented in figure 3.

When visiting an instantiated production p, having M as an n-best matrix, with the two parameters x and y, the intermediate region of M, with respect to x and y, contains, by definition, all the ranks that we are interested in (the ranks ranging from x to y). This region can be partitioned into a collection of disjoint rectangular regions. Each such partition therefore defines a collection of rectangles or a q-partition.

The computation of the four parameters x_r^1, y_r^1, x_r^2 and y_r^2 for an instantiated production p

$B_{i..l}$ \ $C_{l..j}$	2	3	4	5
2	5	11	13	18
3	9	12	17	24
4	10	20	21	26
5	19	22	25	27

Figure 3: The sub-matrix corresponding to the rectangle $\langle\langle 2, 5\rangle, \langle 2, 5\rangle\rangle$

therefore boils down to the computation of a partition of the intermediate region of the n-best matrix of p.

We have represented schematically, in figure 4, two 4-partitions and a 3-partition of the intermediate region of the matrix of figure 2. The leftmost (resp. rightmost) partition will be called the vertical (resp. horizontal) partition. The middle partition will be called an optimal partition, it decomposes the intermediate region into a minimal number of sub-matrices.

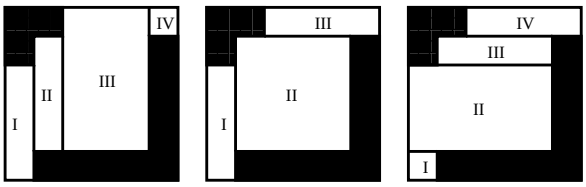


Figure 4: Three partitions of an n-best matrix

The three partitions of figure 4 will give birth to the following instantiated productions:

- Vertical partition

$$A_{i..j}^{4,27} \rightarrow B_{i..l}^{3,6}\ C_{l..j}^{1,1} \quad A_{i..j}^{4,27} \rightarrow B_{i..l}^{2,5}\ C_{l..j}^{2,2}$$
$$A_{i..j}^{4,27} \rightarrow B_{i..l}^{1,5}\ C_{l..j}^{3,5} \quad A_{i..j}^{4,27} \rightarrow B_{i..l}^{1,1}\ C_{l..j}^{6,6}$$

- Optimal partition

$$A_{i..j}^{4,27} \rightarrow B_{i..l}^{1,1}\ C_{l..j}^{3,6} \quad A_{i..j}^{4,27} \rightarrow B_{i..l}^{2,5}\ C_{l..j}^{2,5}$$
$$A_{i..j}^{4,27} \rightarrow B_{i..l}^{3,6}\ C_{l..j}^{1,1}$$

- Horizontal partition

$$A_{i..j}^{4,27} \rightarrow B_{i..l}^{1,1}\ C_{l..j}^{3,6} \quad A_{i..j}^{4,27} \rightarrow B_{i..l}^{2,2}\ C_{l..j}^{2,5}$$
$$A_{i..j}^{4,27} \rightarrow B_{i..l}^{3,5}\ C_{l..j}^{1,5} \quad A_{i..j}^{4,27} \rightarrow B_{i..l}^{6,6}\ C_{l..j}^{1,1}$$

Vertical and horizontal partition of the intermediate region of a n-best matrix can easily be computed. We are not aware of an efficient method that computes an optimal partition. In the implementation used for experiments described in section 4, a simple heuristic has been used which computes horizontal and vertical partitions and keeps the partition with the lower number of parts.

The size of the new forest is clearly linked to the partitions that are computed: a partition with a lower number of parts will give birth to a lower number of decorated instantiated productions and therefore a smaller forest. But this optimization is local, it does not take into account the fact that an instantiated symbol may be shared in the initial forest. During the computation of the new forest, an instantiated production p can therefore be visited several times, with different parameters, and several partitions of p be computed. If a rectangle is shared by several partitions, this will tend to decrease the size of the new forest. The global optimal must therefore take into account all the partitions of an instantiated production that are computed during the construction of the new forest.

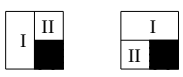
Example 5: Applying the rectangles method to the second running example.

We now illustrate more concretely the rectangles method on our second running example introduced in Example 3. Let us recall that we are interested in the $n = 3$ best trees, the original forest containing 4 trees.

As said above, this method starts on the instantiated axiom $S_{1..3}$. Since it is the left-hand side of only one production, this production is visited with parameters $1, 3$. Moreover, its n-best table is the same as that of $S_{1..3}$, given in Example 3. We show here the corresponding n-best matrix, with the empty lower region, the intermediate region (cells corresponding to ranks 1 to 3) and the upper region:

		$B_{2..3}$	
		1	2
$A_{1..2}$	1	1	2
	2	3	4

As can be seen on that matrix, there are two optimal 2-partitions, namely the horizontal and the vertical partitions, illustrated as follows:



Let us arbitrarily chose the vertical partition. It gives birth to two $S_{1..3}$-productions, namely:

$$S^{1,3}_{1..3} \rightarrow A^{1,2}_{1..2}\ B^{1,1}_{2..3}$$
$$S^{1,3}_{1..3} \rightarrow A^{1,1}_{1..2}\ B^{2,2}_{2..3}$$

Since this is the only non-trivial step while applying the rectangles algorithm to this example, we can now give its final result, in which the axiom's (unnecessary) decorations have been removed:

$$S_{1..3} \rightarrow A^{1,2}_{1..2}\ B^{\{1,1\}}_{2..3}$$
$$S_{1..3} \rightarrow A^{1,1}_{1..2}\ B^{\{2,2\}}_{2..3}$$
$$A^{1,2}_{1..2} \rightarrow A1_{1..2} \qquad A1_{1..2} \rightarrow a_{1..2}$$
$$A^{1,2}_{1..2} \rightarrow A2_{1..2} \qquad A2_{1..2} \rightarrow a_{1..2}$$
$$B^{1,2}_{2..3} \rightarrow B1_{2..3} \qquad B1_{2..3} \rightarrow b_{2..3}$$
$$B^{2,2}_{2..3} \rightarrow B2_{2..3} \qquad B2_{2..3} \rightarrow b_{2..3}$$

Compared to the forest built by the ranksets algorithm, this forest has one less production and one less non-terminal symbol. It has only one more production than the over-generating pruned forest.

4 Experiments on the Penn Treebank

The methods described in section 3 have been tested on a PCFG G extracted from the Penn Treebank (Marcus et al., 1993). G has been extracted naively: the trees have been decomposed into binary context free rules, and the probability of every rule has been estimated by its relative frequency (number of occurrences of the rule divided by the number of occurrences of its left hand side). Rules occurring less than 3 times and rules with probabilities lower than 3×10^{-4} have been eliminated. The grammar produced contains 932 non terminals and $3,439$ rules.[7]

The parsing has been realized using the SYNTAX system which implements, and optimizes, the Earley algorithm (Boullier, 2003).

The evaluation has been conducted on the $1,845$ sentences of section 1, which constitute our test

[7] We used this test set only to generate practical NLP forests, with a real NLP grammar, and evaluate the performances of our algorithms for constucting sub-forests that contain only the n-best trees, both in terms of compression rate and execution time. Therefore, the evaluation carried out here has nothing to do with the usual evaluation of the precision and recall of parsers based on the Penn Treebank. In particular, we are not interested here in the accuracy of such a grammar, its only purpose is to generate parse forests from which n-best sub-forests will be built.

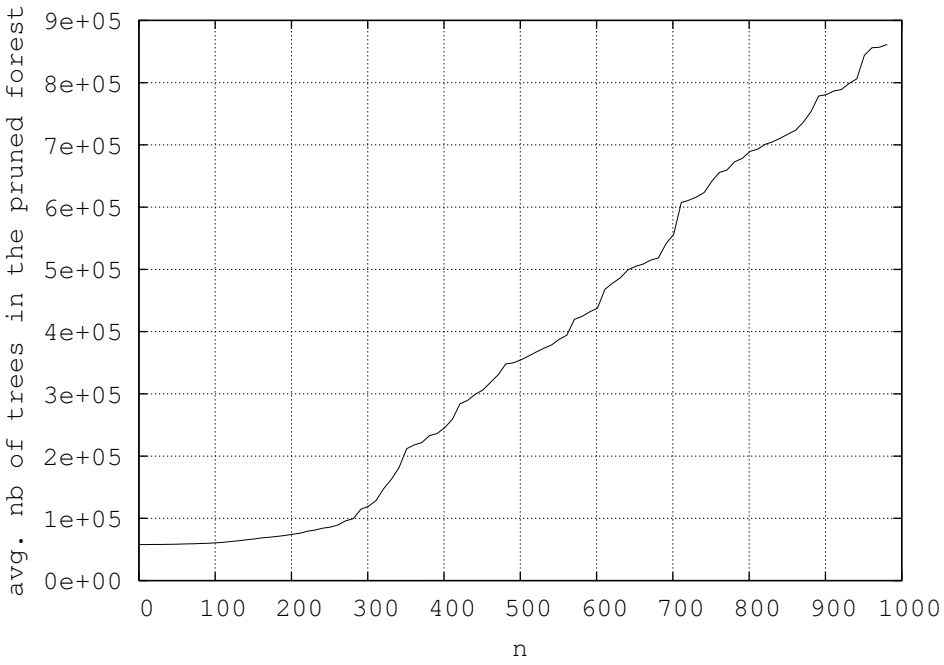


Figure 5: Overgeneration of the pruned n-best forest

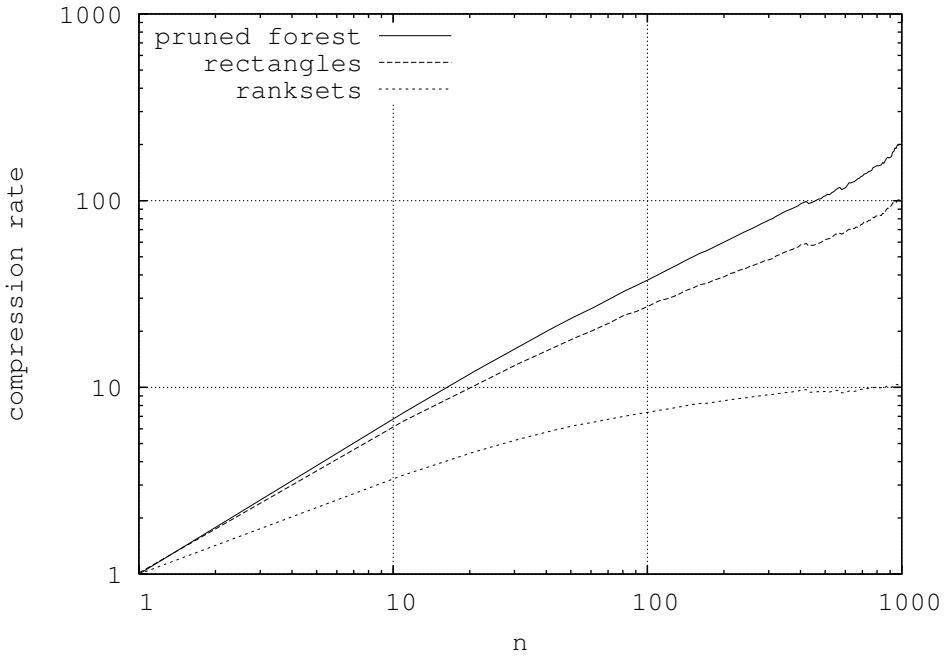


Figure 6: Average compression rates

set. For every sentence and for increasing values of n, an n-best sub-forest has been built using the rankset and the rectangles method.

The performances of the algorithms have been measured by the average *compression rate* they achieve for different values of n. The compression rate is obtained by dividing the size of the n-best sub-forest of a sentence, as defined in section 2, by the size of the (unfolded) n-best forest. The latter is the sum of the sizes of all trees in the forest, where every tree is seen as an instantiated grammar, its size is therefore the size of the corresponding instantiated grammar.

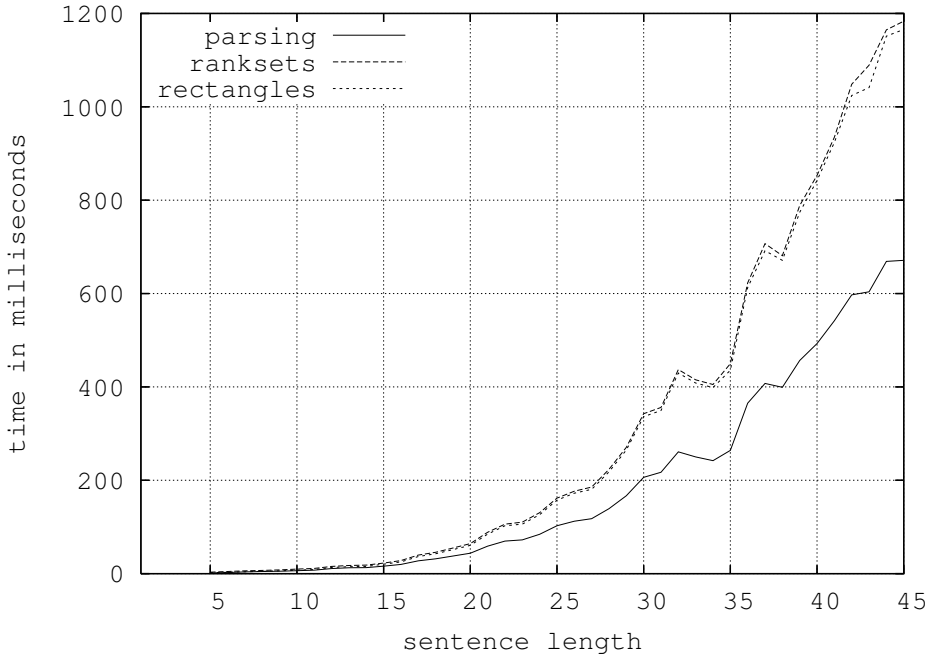


Figure 7: Processing time

The size of the n-best forest constitutes a natural upper bound for the representation of the n-best trees. Unfortunately, we have no natural lower bound for the size of such an object. Nevertheless, we have computed the compression rates of the pruned n-best forest and used it as an imperfect lower bound. As already mentioned, its imperfection comes from the fact that a pruned n-best forest contains more trees than the n best ones. This overgeneration appears clearly in Figure 5 which shows, for increasing values of n, the average number of trees in the n-best pruned forest for all sentences in our test set.

Figure 6 shows the average compression rates achieved by the three methods (forest pruning, rectangles and ranksets) on the test set for increasing values of n. As predicted, the performances lie between 1 (no compression) and the compression of the n-best pruned forest. The rectangle method outperforms the ranksets algorithm for every value of n.

The time needed to build an 100-best forest with the rectangle and the ranksets algorithms is shown in Figure 7. This figure shows the average parsing time for sentences of a given length, as well as the average time necessary for building the 100-best forest using the two aforementioned algorithms. This time includes the parsing time i.e. it is the time necessary for parsing a sentence and building the 100-best forest. As shown by the figure, the time complexities of the two methods are very close.

5 Conclusion and perspectives

This work presented two methods to build n-best sub-forests. The so called rectangle methods showed to be the most promising, for it allows to build efficient sub-forests with little time overhead. Future work will focus on computing optimized partitions of the n-best matrices, a crucial part of the rectangle method, and adapting the method to arbitrary (non binary) CFG. Another line of research will concentrate on performing re-ranking of the n-best trees directly on the sub-forest.

Acknowledgments

This research is supported by the French National Research Agency (ANR) in the context of the SEQUOIA project (ANR-08-EMER-013).

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

Taylor L. Booth. 1969. Probabilistic representation of formal languages. In *Tenth Annual Symposium on Switching and Automata Theory*, pages 74–81.

Pierre Boullier and Philippe Deschamp. 1988. Le système SYNTAX^{TM} - manuel d'utilisation. http://syntax.gforge.inria.fr/syntax3.8-manual.pdf.

Pierre Boullier and Benot Sagot. 2005. Efficient and robust LFG parsing: SXLFG. In *Proceedings of IWPT'05*, Vancouver, Canada.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of IWPT'03*, pages 43–54.

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

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of IWPT'05*, pages 53–64.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL'08*, pages 586–594.

Víctor M. Jiménez and Andrés Marzal. 2000. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 183–192, London, United Kingdom. Springer-Verlag.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of IWPT'01*.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In J. Loeckx, editor, *Proceedings of the Second Colloquium on Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269. Springer-Verlag.

Bernard Lang. 1994. Recognition can be harder then parsing. *Computational Intelligence*, 10:486–494.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, June.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proceedings of ACL-08: HLT*, pages 192–199.