An alternative method of training probabilistic LR parsers

Mark-Jan Nederhof
Faculty of Arts
University of Groningen
P.O. Box 716
NL-9700 AS Groningen
The Netherlands
markjan@let.rug.nl

Giorgio Satta
Dept. of Information Engineering
University of Padua
via Gradenigo, 6/A
I-35131 Padova
Italy
satta@dei.unipd.it

Abstract

We discuss existing approaches to train LR parsers, which have been used for statistical resolution of structural ambiguity. These approaches are non-optimal, in the sense that a collection of probability distributions cannot be obtained. In particular, some probability distributions expressible in terms of a context-free grammar cannot be expressed in terms of the LR parser constructed from that grammar, under the restrictions of the existing approaches to training of LR parsers. We present an alternative way of training that is provably optimal, and that allows all probability distributions expressible in the context-free grammar to be carried over to the LR parser. We also demonstrate empirically that this kind of training can be effectively applied on a large treebank.

1 Introduction

The LR parsing strategy was originally devised for programming languages (Sippu and Soisalon-Soininen, 1990), but has been used in a wide range of other areas as well, such as for natural language processing (Lavie and Tomita, 1993; Briscoe and Carroll, 1993; Ruland, 2000). The main difference between the application to programming languages and the application to natural languages is that in the latter case the parsers should be nondeterministic, in order to deal with ambiguous context-free grammars (CFGs). Nondeterminism can be handled in a number of ways, but the most efficient is tabulation, which allows processing in polynomial time. Tabular LR parsing is known from the work by (Tomita, 1986), but can also be achieved by the generic tabulation technique due to (Lang, 1974; Billot and Lang, 1989), which assumes an input pushdown transducer (PDT). In this context, the LR parsing strategy can be seen as a particular mapping from context-free grammars to PDTs.

The acronym 'LR' stands for 'Left-to-right processing of the input, producing a Right-most derivation (in reverse)'. When we construct a PDT $\mathcal{A}$ from a CFG $\mathcal{G}$ by the LR parsing strategy and apply it on an input sentence, then the set of output strings of $\mathcal{A}$ represents the set of all right-most derivations that $\mathcal{G}$ allows for that sentence. Such an output string enumerates the rules (or labels that identify the rules uniquely) that occur in the corresponding right-most derivation, in reversed order.

If LR parsers do not use lookahead to decide between alternative transitions, they are called LR(0) parsers. More generally, if LR parsers look ahead k symbols, they are called LR(k) parsers; some simplified LR parsing models that use lookahead are called SLR(k) and LALR(k) parsing (Sippu and Soisalon-Soininen, 1990). In order to simplify the discussion, we abstain from using lookahead in this article, and 'LR parsing' can further be read as 'LR(0) parsing'. We would like to point out however that our observations carry over to LR parsing *with* lookahead.

The theory of probabilistic pushdown automata (Santos, 1972) can be easily applied to LR parsing. A probability is then assigned to each transition, by a function that we will call the *probability function* $p_{\mathcal{A}}$, and the probability of an accepting computation of $\mathcal{A}$ is the product of the probabilities of the applied transitions. As each accepting computation produces a right-most derivation as output string, a probabilistic LR parser defines a probability distribution on the set of parses, and thereby also a probability distribution on the set of sentences generated by grammar $\mathcal{G}$. Disambiguation of an ambiguous sentence can be achieved on the basis of a comparison between the probabilities assigned to the respective parses by the probabilistic LR model.

The probability function can be obtained on the basis of a treebank, as proposed by (Briscoe and Carroll, 1993) (see also (Su et al., 1991)). The model by (Briscoe and Carroll, 1993) however incorporated a mistake involving lookahead, which was corrected by (Inui et al., 2000). As we will not discuss lookahead here, this matter does not play a significant role in the current study. Noteworthy is that (Sornlertlamvanich et al., 1999) showed empirically that an LR parser may be more accurate than the original CFG, if both are trained on the basis of the same treebank. In other words, the resulting probability function $p_{\mathcal{A}}$ on transitions of the PDT allows better disambiguation than the corresponding function $p_{\mathcal{G}}$ on rules of the original grammar.

A plausible explanation of this is that stack symbols of an LR parser encode some amount of left context, i.e. information on rules applied earlier, so that the probability function on transitions may encode dependencies between rules that cannot be encoded in terms of the original CFG extended with rule probabilities. The explicit use of left context in probabilistic context-free models was investigated by e.g. (Chitrao and Grishman, 1990; Johnson, 1998), who also demonstrated that this may significantly improve accuracy. Note that the probability distributions of language may be beyond the reach of a given context-free grammar, as pointed out by e.g. (Collins, 2001). Therefore, the use of left context, and the resulting increase in the number of parameters of the model, may narrow the gap between the given grammar and ill-understood mechanisms underlying actual language.

One important assumption that is made by (Briscoe and Carroll, 1993) and (Inui et al., 2000) is that trained probabilistic LR parsers should be *proper*, i.e. if several transitions are applicable for a given stack, then the sum of probabilities assigned to those transitions by probability function $p_{\mathcal{A}}$ should be 1. This assumption may be motivated by pragmatic considerations, as such a proper model is easy to train by *relative frequency estimation*: count the number of times a transition is applied with respect to a treebank, and divide it by the number of times the relevant stack symbol (or pair of stack symbols) occurs at the top of the stack. Let us call the resulting probability function p_{rfe}. This function is provably optimal in the sense that the likelihood it assigns to the training corpus is maximal among all probability functions $p_{\mathcal{A}}$ that are proper in the above sense.

However, properness restricts the space of probability distributions that a PDT allows. This means that a (consistent) probability function $p_{\mathcal{A}}$ may exist that is not proper and that assigns a higher likelihood to the training corpus than p_{rfe} does. (By 'consistent' we mean that the probabilities of all strings that are accepted sum to 1.) It may even be the case that a (proper and consistent) probability function $p_{\mathcal{G}}$ on the rules of the input grammar $\mathcal{G}$ exists that assigns a higher likelihood to the corpus than p_{rfe}, and therefore it is not guaranteed that LR parsers allow better probability estimates than the CFGs from which they were constructed, if we constrain probability functions $p_{\mathcal{A}}$ to be proper. In this respect, LR parsing differs from at least one other well-known parsing strategy, viz. left-corner parsing. See (Nederhof and Satta, 2004) for a discussion of a property that is shared by left-corner parsing but not by LR parsing, and which explains the above difference.

As main contribution of this paper we establish that this restriction on expressible probability distributions can be dispensed with, without losing the ability to perform training by relative frequency estimation. What comes in place of properness is *reverse-properness*, which can be seen as properness of the reversed pushdown automaton that processes input from right to left instead of from left to right, interpreting the transitions of $\mathcal{A}$ backwards. As we will show, reverse-properness does not restrict the space of probability distributions expressible by an LR automaton. More precisely, assume some probability distribution on the set of derivations is specified by a probability function $p_{\mathcal{A}}$ on transitions of PDT $\mathcal{A}$ that realizes the LR strategy for a given grammar $\mathcal{G}$. Then the same probability distribution can be specified by an alternative such function $p'_{\mathcal{A}}$ that is reverse-proper. In addition, for each probability distribution on derivations expressible by a probability function $p_{\mathcal{G}}$ for $\mathcal{G}$, there is a reverse-proper probability function $p_{\mathcal{A}}$ for $\mathcal{A}$ that expresses the same probability distribution. Thereby we ensure that LR parsers become at least as powerful as the original CFGs in terms of allowable probability distributions.

This article is organized as follows. In Section 2 we outline our formalization of LR parsing as a construction of PDTs from CFGs, making some superficial changes with respect to standard formulations. Properness and reverse-properness are discussed in Section 3, where we will show that reverse-properness does not restrict the space of probability distributions. Section 4 reports on experiments, and Section 5 concludes this article.

2 LR parsing

As LR parsing has been extensively treated in existing literature, we merely recapitulate the main definitions here. For more explanation, the reader is referred to standard literature such as (Harrison, 1978; Sippu and Soisalon-Soininen, 1990).

An LR parser is constructed on the basis of a CFG that is augmented with an additional rule $S^{\dagger} \rightarrow \vdash S$, where S is the former start symbol, and the new nonterminal $S^{\dagger}$ becomes the start symbol of the augmented grammar. The new terminal $\vdash$ acts as an imaginary start-of-sentence marker. We denote the set of terminals by Σ and the set of nonterminals by N. We assume each rule has a unique label r.

As explained before, we construct LR parsers as pushdown transducers. The main stack symbols of these automata are sets of *dotted rules*, which consist of rules from the augmented grammar with a distinguished position in the right-hand side indicated by a dot '$\bullet$'. The initial stack symbol is $p_{init} = \{S^\dagger \rightarrow \vdash \bullet S\}$.

We define the closure of a set p of dotted rules as the smallest set $closure(p)$ such that:

1. $p \subseteq closure(p)$; and
2. for $(B \rightarrow \alpha \bullet A\beta) \in closure(p)$ and $A \rightarrow \gamma$ a rule in the grammar, also $(A \rightarrow \bullet\ \gamma) \in closure(p)$.

We define the operation *goto* on a set p of dotted rules and a grammar symbol $X \in \Sigma \cup N$ as:

$$\begin{aligned} goto(p, X) \;=\; & \{A \rightarrow \alpha X \bullet \beta \mid \\ & (A \rightarrow \alpha \bullet X\beta) \in closure(p)\} \end{aligned}$$

The set of *LR states* is the smallest set such that:

1. p_{init} is an LR state; and
2. if p is an LR state and $goto(p, X) = q \neq \emptyset$, for some $X \in \Sigma \cup N$, then q is an LR state.

We will assume that PDTs consist of three types of transitions, of the form $P \overset{a,b}{\mapsto} P\ Q$ (a push transition), of the form $P \overset{a,b}{\mapsto} Q$ (a swap transition), and of the form $P\ Q \overset{a,b}{\mapsto} R$ (a pop transition). Here P, Q and R are stack symbols, a is one input terminal or is the empty string ε, and b is one output terminal or is the empty string ε. In our notation, stacks grow from left to right, so that $P \overset{a,b}{\mapsto} P\ Q$ means that Q is pushed on top of P. We do not have internal states next to stack symbols.

For the PDT that implements the LR strategy, the stack symbols are the LR states, plus symbols of the form $[p; X]$, where p is an LR state and X is a grammar symbol, and symbols of the form (p, A, m), where p is an LR state, A is the left-hand side of some rule, and m is the length of some prefix of the right-hand side of that rule. More explanation on these additional stack symbols will be given below.

The stack symbols and transitions are simultaneously defined in Figure 1. The final stack symbol is $p_{final} = (p_{init}, S^\dagger, 0)$. This means that an input $a_1 \cdots a_n$ is accepted if and only if it is entirely read by a sequence of transitions that take the stack consisting only of p_{init} to the stack consisting only of p_{final}. The computed output consists of the string of terminals $b_1 \cdots b_{n'}$ from the output components of the applied transitions. For the PDTs that we will use, this output string will consist of a sequence of rule labels expressing a right-most derivation of the input. On the basis of the original grammar, the corresponding parse tree can be constructed from such an output string.

There are a few superficial differences with LR parsing as it is commonly found in the literature. The most obvious difference is that we divide reductions into 'binary' steps. The main reason is that this allows tabular interpretation with a time complexity cubic in the length of the input. Otherwise, the time complexity would be $\mathcal{O}(n^{m+1})$, where m is the length of the longest right-hand side of a rule in the CFG. This observation was made before by (Kipps, 1991), who proposed a solution similar to ours, albeit formulated differently. See also a related formulation of tabular LR parsing by (Nederhof and Satta, 1996).

To be more specific, instead of one step of the PDT taking stack:
$\sigma p_0 p_1 \cdots p_m$
immediately to stack:
$\sigma p_0 q$
where $(A \rightarrow X_1 \cdots X_m \bullet) \in p_m$, σ is a string of stack symbols and $goto(p_0, A) = q$, we have a number of smaller steps leading to a series of stacks:

$$\begin{array}{l} \sigma p_0 p_1 \cdots p_{m-1} p_m \\ \sigma p_0 p_1 \cdots p_{m-1} (A, m{-}1) \\ \sigma p_0 p_1 \cdots (A, m{-}2) \\ \vdots \\ \sigma p_0 (A, 0) \\ \sigma p_0 q \end{array}$$

There are two additional differences. First, we want to avoid steps of the form:

$$\begin{array}{l} \sigma p_0 (A, 0) \\ \sigma p_0 q \end{array}$$

by transitions $p_0\ (A, 0) \overset{\varepsilon,\varepsilon}{\mapsto} p_0\ q$, as such transitions complicate the generic definition of 'properness' for PDTs, to be discussed in the following section. For this reason, we use stack symbols of the form $[p; X]$ next to p, and split up $p_0\ (A, 0) \overset{\varepsilon,\varepsilon}{\mapsto} p_0\ q$ into pop $[p_0; X_0]\ (A, 0) \overset{\varepsilon,\varepsilon}{\mapsto} [p_0; A]$ and push $[p_0; A] \overset{\varepsilon,\varepsilon}{\mapsto} [p_0; A]\ q$. This is a harmless modification, which increases the number of steps in any computation by at most a factor 2.

Secondly, we use stack symbols of the form (p, A, m) instead of (A, m). This concerns the conditions of reverse-properness to be discussed in the

- For LR state p and $a \in \Sigma$ such that *goto*$(p, a) \neq \emptyset$:

$$p \overset{a,\varepsilon}{\mapsto} [p; a] \tag{1}$$

- For LR state p and $(A \rightarrow \bullet) \in p$, where $A \rightarrow \varepsilon$ has label r:

$$p \overset{\varepsilon,r}{\mapsto} [p; A] \tag{2}$$

- For LR state p and $(A \rightarrow \alpha \bullet) \in p$, where $|\alpha| = m > 0$ and $A \rightarrow \alpha$ has label r:

$$p \overset{\varepsilon,r}{\mapsto} (p, A, m-1) \tag{3}$$

- For LR state p and $(A \rightarrow \alpha \bullet X\beta) \in p$, where $|\alpha| = m > 0$, such that *goto*$(p, X) = q \neq \emptyset$:

$$[p; X]\ (q, A, m) \overset{\varepsilon,\varepsilon}{\mapsto} (p, A, m-1) \tag{4}$$

- For LR state p and $(A \rightarrow \bullet\ X\beta) \in p$, such that *goto*$(p, X) = q \neq \emptyset$:

$$[p; X]\ (q, A, 0) \overset{\varepsilon,\varepsilon}{\mapsto} [p; A] \tag{5}$$

- For LR state p and $X \in \Sigma \cup N$ such that *goto*$(p, X) = q \neq \emptyset$:

$$[p; X] \overset{\varepsilon,\varepsilon}{\mapsto} [p; X]\ q \tag{6}$$

Figure 1: The transitions of a PDT implementing LR(0) parsing.

following section. By this condition, we consider LR parsing as being performed from right to left, so backwards with regard to the normal processing order. If we were to omit the first components p from stack symbols (p, A, m), we may obtain 'dead ends' in the computation. We know that such dead ends make a (reverse-)proper PDT inconsistent, as probability mass lost in dead ends causes the sum of probabilities of all computations to be strictly smaller than 1. (See also (Nederhof and Satta, 2004).) It is interesting to note that the addition of the components p to stack symbols (p, A, m) does *not* increase the number of transitions, and the nature of LR parsing in the normal processing order from left to right is preserved.

With all these changes together, reductions are implemented by transitions resulting in the following sequence of stacks:

$$\begin{array}{l}
\sigma'[p_0; X_0][p_1; X_1] \cdots [p_{m-1}; X_{m-1}]p_m \\
\sigma'[p_0; X_0][p_1; X_1] \cdots [p_{m-1}; X_{m-1}](p_m, A, m-1) \\
\sigma'[p_0; X_0][p_1; X_1] \cdots (p_{m-1}, A, m-2) \\
\vdots \\
\sigma'[p_0; X_0](p_1, A, 0) \\
\sigma'[p_0; A] \\
\sigma'[p_0; A]q
\end{array}$$

Please note that transitions of the form $[p; X]\ (q, A, m) \overset{\varepsilon,\varepsilon}{\mapsto} (p, A, m-1)$ may correspond to several dotted rules $(A \rightarrow \alpha \bullet X\beta) \in p$, with different α of length m and different β. If we were to multiply such transitions for different α and β, the PDT would become prohibitively large.

3 Properness and reverse-properness

If a PDT is regarded to process input from left to right, starting with a stack consisting only of p_{init}, and ending in a stack consisting only of p_{final}, then it seems reasonable to cast this process into a probabilistic framework in such a way that the sum of probabilities of all choices that are possible at any given moment is 1. This is similar to how the notion of 'properness' is defined for probabilistic context-free grammars (PCFGs); we say a PCFG is proper if for each nonterminal A, the probabilities of all rules with left-hand side A sum to 1.

Properness for PCFGs does not restrict the space of probability distributions on the set of parse trees. In other words, if a probability distribution can be defined by attaching probabilities to rules, then we may reassign the probabilities such that that PCFG becomes proper, while preserving the probability distribution. This even holds if the input grammar is non-tight, meaning that probability mass is lost in 'infinite derivations' (Sánchez and Benedí, 1997; Chi and Geman, 1998; Chi, 1999; Nederhof and Satta, 2003).

Although CFGs and PDTs are weakly equivalent, they behave very differently when they are extended with probabilities. In particular, there seems to be no notion similar to PCFG properness that can be imposed on all types of PDTs without losing generality. Below we will discuss two constraints, which we will call properness and reverse-properness. Neither of these is suitable for all types of PDTs, but as we will show, the second is more suitable for probabilistic LR parsing than the first. This is surprising, as only properness has been described in existing literature on probabilistic PDTs (PPDTs). In particular, all existing approaches to probabilistic LR parsing have assumed properness rather than anything related to reverse-properness.

For properness we have to assume that for each stack symbol P, we either have one or more transitions of the form $P \overset{a,b}{\mapsto} P\ Q$ or $P \overset{a,b}{\mapsto} Q$, or one or more transitions of the form $Q\ P \overset{a,b}{\mapsto} R$, but no combination thereof. In the first case, properness demands that the sum of probabilities of all transitions $P \overset{a,b}{\mapsto} P\ Q$ and $P \overset{a,b}{\mapsto} Q$ is 1, and in the second case properness demands that the sum of probabilities of all transitions $Q\ P \overset{a,b}{\mapsto} R$ is 1 for each Q.

Note that our assumption above is without loss of generality, as we may introduce swap transitions $P \overset{\varepsilon,\varepsilon}{\mapsto} P_1$ and $P \overset{\varepsilon,\varepsilon}{\mapsto} P_2$, where P_1 and P_2 are new stack symbols, and replace transitions $P \overset{a,b}{\mapsto} P\ Q$ and $P \overset{a,b}{\mapsto} Q$ by $P_1 \overset{a,b}{\mapsto} P_1\ Q$ and $P_1 \overset{a,b}{\mapsto} Q$, and replace transitions $Q\ P \overset{a,b}{\mapsto} R$ by $Q\ P_2 \overset{a,b}{\mapsto} R$.

The notion of properness underlies the normal training process for PDTs, as follows. We assume a corpus of PDT computations. In these computations, we count the number of occurrences for each transition. For each P we sum the total number of all occurrences of transitions $P \overset{a,b}{\mapsto} P\ Q$ or $P \overset{a,b}{\mapsto} Q$. The probability of, say, a transition $P \overset{a,b}{\mapsto} P\ Q$ is now estimated by dividing the number of occurrences thereof in the corpus by the above total number of occurrences of transitions with P in the left-hand side. Similarly, for each pair (Q, P) we sum the total number of occurrences of all transitions of the form $Q\ P \overset{a,b}{\mapsto} R$, and thereby estimate the probability of a particular transition $Q\ P \overset{a,b}{\mapsto} R$ by relative frequency estimation. The resulting PPDT is proper.

It has been shown that imposing properness is without loss of generality in the case of PDTs constructed by a wide range of parsing strategies, among which are top-down parsing and left-corner parsing. This does not hold for PDTs constructed by the LR parsing strategy however, and in fact, properness for such automata may reduce the expressive power in terms of available probability distributions to strictly less than that offered by the original CFG. This was formally proven by (Nederhof and Satta, 2004), after (Ng and Tomita, 1991) and (Wright and Wrigley, 1991) had already suggested that creating a probabilistic LR parser that is equivalent to an input PCFG is difficult in general. The same difficulty for ELR parsing was suggested by (Tendeau, 1997).

For this reason, we investigate a practical alternative, viz. reverse-properness. Now we have to assume that for each stack symbol R, we either have one or more transitions of the form $P \overset{a,b}{\mapsto} R$ or $Q\ P \overset{a,b}{\mapsto} R$, or one or more transitions of the form $P \overset{a,b}{\mapsto} P\ R$, but no combination thereof. In the first case, reverse-properness demands that the sum of probabilities of all transitions $P \overset{a,b}{\mapsto} R$ or $Q\ P \overset{a,b}{\mapsto} R$ is 1, and in the second case reverse-properness demands that the sum of probabilities of transitions $P \overset{a,b}{\mapsto} P\ R$ is 1 for each P. Again, our assumption above is without loss of generality.

In order to apply relative frequency estimation, we now sum the total number of occurrences of transitions $P \overset{a,b}{\mapsto} R$ or $Q\ P \overset{a,b}{\mapsto} R$ for each R, and we sum the total number of occurrences of transitions $P \overset{a,b}{\mapsto} P\ R$ for each pair (P, R).

We now prove that reverse-properness does not restrict the space of probability distributions, by means of the construction of a 'cover' grammar from an input CFG, as reported in Figure 2. This cover CFG has almost the same structure as the PDT resulting from Figure 1. Rules and transitions almost stand in a one-to-one relation. The only noteworthy difference is between transitions of type (6) and rules of type (12). The right-hand sides of those rules can be ε because the corresponding transitions are deterministic if seen from right to left. Now it becomes clear why we needed the components p in stack symbols of the form (p, A, m). Without it, one could obtain an LR state q that does not match the underlying $[p; X]$ in a reversed computation.

We may assume without loss of generality that rules of type (12) are assigned probability 1, as a probability other than 1 could be moved to corresponding rules of types (10) or (11) where state q was introduced. In the same way, we may assume that transitions of type (6) are assigned probability 1. After making these assumptions, we obtain a bijection between probability functions $p_{\mathcal{A}}$ for the PDT and probability functions $p_{\mathcal{G}}$ for the cover CFG. As was shown by e.g. (Chi, 1999) and (Nederhof and Satta, 2003), properness for CFGs does not restrict the space of probability distributions, and thereby the same holds for reverse-properness for PDTs that implement the LR parsing strategy.

It is now also clear that a reverse-proper LR parser can describe any probability distribution that the original CFG can. The proof is as follows. Given a probability function $p_{\mathcal{G}}$ for the input CFG, we define a probability function $p_{\mathcal{A}}$ for the LR parser, by letting transitions of types (2) and (3)

- For LR state p and $a \in \Sigma$ such that *goto*$(p, a) \neq \emptyset$:

$$[p; a] \rightarrow p \tag{7}$$

- For LR state p and $(A \rightarrow \bullet) \in p$, where $A \rightarrow \varepsilon$ has label r:

$$[p; A] \rightarrow p\ r \tag{8}$$

- For LR state p and $(A \rightarrow \alpha\ \bullet) \in p$, where $|\alpha| = m > 0$ and $A \rightarrow \alpha$ has label r:

$$(p, A, m-1) \rightarrow p\ r \tag{9}$$

- For LR state p and $(A \rightarrow \alpha \bullet X\beta) \in p$, where $|\alpha| = m > 0$, such that *goto*$(p, X) = q \neq \emptyset$:

$$(p, A, m-1) \rightarrow [p; X]\ (q, A, m) \tag{10}$$

- For LR state p and $(A \rightarrow \bullet\ X\beta) \in p$, such that *goto*$(p, X) = q \neq \emptyset$:

$$[p; A] \rightarrow [p; X]\ (q, A, 0) \tag{11}$$

- For LR state q:

$$q \rightarrow \varepsilon \tag{12}$$

Figure 2: A grammar that describes the set of computations of the LR(0) parser. Start symbol is $p_{\mathit{final}} = (p_{\mathit{init}}, S^{\dagger}, 0)$. Terminals are rule labels. Generated language consists of right-most derivations in reverse.

have probability $p_{\mathcal{G}}(r)$, and letting all other transitions have probability 1. This gives us the required probability distribution in terms of a PPDT that is not reverse-proper in general. This PPDT can now be recast into reverse-proper form, as proven by the above.

4 Experiments

We have implemented both the traditional training method for LR parsing and the novel one, and have compared their performance, with two concrete objectives:

1. We show that the number of free parameters is significantly larger with the new training method. (The number of free parameters is the number of probabilities of transitions that can be freely chosen within the constraints of properness or reverse-properness.)
2. The larger number of free parameters does not make the problem of sparse data any worse, and precision and recall are at least comparable to, if not better than, what we would obtain with the established method.

The experiments were performed on the Wall Street Journal (WSJ) corpus, from the Penn Treebank, version II. Training was done on sections 02-21, i.e., first a context-free grammar was derived from the 'stubs' of the combined trees, taking parts of speech as leaves of the trees, omitting all affixes from the nonterminal names, and removing ε-generating subtrees. Such preprocessing of the WSJ corpus is consistent with earlier attempts to derive CFGs from that corpus, as e.g. by (Johnson, 1998). The obtained CFG has 10,035 rules. The dimensions of the LR parser constructed from this grammar are given in Table 1.

The PDT was then trained on the trees from the same sections 02-21, to determine the number of times that transitions are used. At first sight it is not clear how to determine this on the basis of the treebank, as the structure of LR parsers is very different from the structure of the grammars from which they are constructed. The solution is to construct a second PDT from the PDT to be trained, replacing each transition $\alpha \overset{a,b}{\mapsto} \beta$ with label r by transition $\alpha \overset{b,r}{\mapsto} \beta$. By this second PDT we parse the treebank, encoded as a series of right-most derivations in reverse.[1] For each input string, there is exactly one parse, of which the output is the list of used transitions. The same method can be used for other parsing strategies as well, such as left-corner parsing, replacing right-most derivations by a suitable alternative representation of parse trees.

By the counts of occurrences of transitions, we may then perform maximum likelihood estimation to obtain probabilities for transitions. This can be done under the constraints of properness or of reverse-properness, as explained in the previous section. We have not applied any form of smooth-

[1]We have observed an enormous gain in computational efficiency when we also incorporate the 'shifts' next to 'reductions' in these right-most derivations, as this eliminates a considerable amount of nondeterminism.

total # transitions	8,340,315
# push transitions	753,224
# swap transitions	589,811
# pop transitions	6,997,280

Table 1: Dimensions of PDT implementing LR strategy for CFG derived from WSJ, sect. 02-21.

	proper	rev.-prop.
# free parameters	577,650	6,589,716
# non-zero probabilities	137,134	137,134
labelled precision	0.772	0.777
labelled recall	0.747	0.749

Table 2: The two methods of training, based on properness and reverse-properness.

ing or back-off, as this could obscure properties inherent in the difference between the two discussed training methods. (Back-off for probabilistic LR parsing has been proposed by (Ruland, 2000).) All transitions that were not seen during training were given probability 0.

The results are outlined in Table 2. Note that the number of free parameters in the case of reverse-properness is much larger than in the case of normal properness. Despite of this, the number of transitions that actually receive non-zero probabilities is (predictably) identical in both cases, viz. 137,134. However, the potential for fine-grained probability estimates and for smoothing and parameter-tying techniques is clearly greater in the case of reverse-properness.

That in both cases the number of non-zero probabilities is lower than the total number of parameters can be explained as follows. First, the treebank contains many rules that occur a small number of times. Secondly, the LR automaton is much larger than the CFG; in general, the size of an LR automaton is bounded by a function that is exponential in the size of the input CFG. Therefore, if we use the same treebank to estimate the probability function, then many transitions are never visited and obtain a zero probability.

We have applied the two trained LR automata on section 22 of the WSJ corpus, measuring labelled precision and recall, as done by e.g. (Johnson, 1998).[2] We observe that in the case of reverse-properness, precision and recall are slightly better. The most important conclusion that can be drawn from this is that the substantially larger space of obtainable probability distributions offered by the reverse-properness method does not come at the expense of a degradation of accuracy for large grammars such as those derived from the WSJ. For comparison, with a standard PCFG we obtain labelled precision and recall of 0.725 and 0.670, respectively.[3]

We would like to stress that our experiments did not have as main objective the improvement of state-of-the-art parsers, which can certainly not be done without much additional fine-tuning and the incorporation of some form of lexicalization. Our main objectives concerned the relation between our newly proposed training method for LR parsers and the traditional one.

5 Conclusions

We have presented a novel way of assigning probabilities to transitions of an LR automaton. Theoretical analysis and empirical data reveal the following.

- The efficiency of LR parsing remains unaffected. Although a right-to-left order of reading input underlies the novel training method, we may continue to apply the parser from left to right, and benefit from the favourable computational properties of LR parsing.
- The available space of probability distributions is significantly larger than in the case of the methods published before. In terms of the number of free parameters, the difference that we found empirically exceeds one order of magnitude. By the same criteria, we can now guarantee that LR parsers are at least as powerful as the CFGs from which they are constructed.
- Despite the larger number of free parameters, no increase of sparse data problems was observed, and in fact there was a small increase in accuracy.

Acknowledgements

Helpful comments from John Carroll and anonymous reviewers are gratefully acknowledged. The first author is supported by the PIONIER Project *Algorithms for Linguistic Processing*, funded by NWO (Dutch Organization for Scientific Research). The second author is partially supported by MIUR under project PRIN No. 2003091149_005.

[2]We excluded all sentences with more than 30 words however, as some required prohibitive amounts of memory. Only one of the remaining 1441 sentences was not accepted by the parser.

[3]In this case, all 1441 sentences were accepted.

References

S. Billot and B. Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June.

T. Briscoe and J. Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Z. Chi and S. Geman. 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Z. Chi. 1999. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.

M.V. Chitrao and R. Grishman. 1990. Statistical parsing of messages. In *Speech and Natural Language, Proceedings*, pages 263–266, Hidden Valley, Pennsylvania, June.

M. Collins. 2001. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, Beijing, China, October.

M.A. Harrison. 1978. *Introduction to Formal Language Theory*. Addison-Wesley.

K. Inui, V. Sornlertlamvanich, H. Tanaka, and T. Tokunaga. 2000. Probabilistic GLR parsing. In H. Bunt and A. Nijholt, editors, *Advances in Probabilistic and other Parsing Technologies*, chapter 5, pages 85–104. Kluwer Academic Publishers.

M. Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

J.R. Kipps. 1991. GLR parsing in time $\mathcal{O}(n^3)$. In M. Tomita, editor, *Generalized LR Parsing*, chapter 4, pages 43–59. Kluwer Academic Publishers.

B. Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming, 2nd Colloquium*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269, Saarbrücken. Springer-Verlag.

A. Lavie and M. Tomita. 1993. GLR* – an efficient noise-skipping parsing algorithm for context free grammars. In *Third International Workshop on Parsing Technologies*, pages 123–134, Tilburg (The Netherlands) and Durbuy (Belgium), August.

M.-J. Nederhof and G. Satta. 1996. Efficient tabular LR parsing. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 239–246, Santa Cruz, California, USA, June.

M.-J. Nederhof and G. Satta. 2003. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, LORIA, Nancy, France, April.

M.-J. Nederhof and G. Satta. 2004. Probabilistic parsing strategies. In *42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain, July.

S.-K. Ng and M. Tomita. 1991. Probabilistic LR parsing for general context-free grammars. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 154–163, Cancun, Mexico, February.

T. Ruland. 2000. A context-sensitive model for probabilistic LR parsing of spoken language with transformation-based postprocessing. In *The 18th International Conference on Computational Linguistics*, volume 2, pages 677–683, Saarbrücken, Germany, July–August.

J.-A. Sánchez and J.-M. Benedí. 1997. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055, September.

E.S. Santos. 1972. Probabilistic grammars and automata. *Information and Control*, 21:27–47.

S. Sippu and E. Soisalon-Soininen. 1990. *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

V. Sornlertlamvanich, K. Inui, H. Tanaka, T. Tokunaga, and T. Takezawa. 1999. Empirical support for new probabilistic generalized LR parsing. *Journal of Natural Language Processing*, 6(3):3–22.

K.-Y. Su, J.-N. Wang, M.-H. Su, and J.-S. Chang. 1991. GLR parsing with scoring. In M. Tomita, editor, *Generalized LR Parsing*, chapter 7, pages 93–112. Kluwer Academic Publishers.

F. Tendeau. 1997. *Analyse syntaxique et sémantique avec évaluation d'attributs dans un demi-anneau*. Ph.D. thesis, University of Orléans.

M. Tomita. 1986. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers.

J.H. Wright and E.N. Wrigley. 1991. GLR parsing with probability. In M. Tomita, editor, *Generalized LR Parsing*, chapter 8, pages 113–128. Kluwer Academic Publishers.