LoPar
Design and Implementation

Helmut Schmid
Institute for Computational Linguistics
University of Stuttgart

January 18, 2000

Contents

Chapter 1

Introduction

LoPar is a parser for probabilistic context-free grammars (PCFGs) and head-lexicalised probabilistic context-free grammars (HPCFGs). Possible applications of this program are symbolic parsing with CFGs, training of (head-lexicalised) PCFGs, statistical parsing with (head-lexicalised) PCFGs, Viterbi parsing with (head-lexicalised) PCFGs, part-of-speech tagging and chunking.

This report is mainly a documentation of the parser implementation and the underlying theoretical concepts and not a manual for the LoPar program. For the latter, the reader is referred to the online manual pages.

1.1 Relationship to the Galacsy Tools

The functionality of LoPar is comparable to that of the Galacsy tools `supar`, `ultra`, `hypar`, `showWords` and `showRules`, which have been developed by Glenn Carroll. However, LoPar is faster and has additional functionality. The file formats of LoPar and the Galacsy tools differ and a conversion is only possible for unlexicalised grammar files.

Chapter 2

Probabilistic Context-Free Grammars

A probabilistic context-free grammar is a context-free grammar which assigns a probability $P(r)$ to each grammar rule `r`. In order for P to be a probability distribution, it is required that the probabilities of all rules with the same left hand side add up to 1.

The probability of a parse tree is defined as follows.

$$P(T) \quad = \quad \prod_{rule\ r} P(r)^{F(r)}$$

$F(r)$ is the number of times, rule `r` has been used to generate `T`.

In contrast to CFGs, a PCFG is able to rank different analyses of a sentence according to their probability. The ranking is used e.g. to reduce a large number of possible analyses to a small set of likely analyses for further processing. However, PCFGs fail to resolve many frequent syntactic ambiguities like PP attachment ambiguities and coordination ambiguities, because the same probability is assigned to two analyses which apply the same rules in different order. The phrase *the man on the hill with the telescope* where the prepositional phrase *with the telescope* could attach to either one of the preceding noun phrases is an example. Disambiguation of these ambiguities requires information about the lexical heads of the constituents, as Hindle and Rooth [Hindle and Rooth, 1993] show. Head-lexicalised probabilistic context-free grammars (HPCFGs) [Carroll, 1995, Carroll and Rooth, 1998] deal with this problem.

2.1 Head-Lexicalised Probabilistic Context-Free Grammars

Syntactically, a head-lexicalised probabilistic context-free grammar (HPCFG) is a probabilistic context-free grammar (PCFG) in which one of the categories on the right hand side of each grammar rule is marked as the head with an apostrophy (').

Example: NP → DT N'

Each constituent bears a lexical head, which is propagated from the head daughter. The lexical head of a terminal node is the respective word form.

HPCFGs assign the following probability to a parse tree T:

$$\begin{aligned} P(T) \quad = \quad & P_{start}(cat(root(T))) \; * \\ & P_{start}(head(root(T)) \,|\, cat(root(T))) \; * \\ & \prod_{nonterm\ n\ in\ T} P_{rule}(r \,|\, cat(n), head(n)) \; * \\ & \prod_{nonroot\ n\ in\ T} P_{choice}(head(n) \,|\, cat(n), cat(parent(n)), head(parent(n))) \; * \\ & \prod_{term\ n\ in\ T} P_{rule}(\langle term\rangle \,|\, cat(n), head(n)) \; * P_{lex}(word(n) \,|\, cat(n), head(n)) \end{aligned}$$

Five families of probability distributions are relevant here. $P_{start}(C)$ is the probability that C is the category of the root node of the parse tree. $P_{start}(h|C)$ is the probability that a root node of category C has lexical head h. $P_{rule}(r|C, h)$ is the probability that a node of category C with lexical head h is expanded with rule r. $P_{choice}(h|C, C_p, h_p)$ is the probability that a (non-head) node of category C bears the lexical head h given that the parent category is C_p and the parent head is h_p. $P_{rule}(\langle term\rangle|C, h)$ is the probability that a node of category C with lexical head h is a terminal node. $P_{lex}(w|C, h)$, finally, is the probability that a terminal node with category C and lexical head h expands to the word form w. If the lexical head of a terminal node is the word form, then $P_{lex}(w|C, h)$ is 1 if w and h are identical and 0 otherwise.

2.2 Relationship between HPCFGs and PCFGs

It is possible to convert a HPCFG into an equivalent PCFG by means of the following construction:

1. Start with an empty PCFG.
2. Add the start symbol TOP to the PCFG.
3. If C is a start symbol of the HPCFG, then add a rule TOP → ⟨C,h⟩ for each lexical head.
4. If $A \rightarrow B_1...B'_i...B_n$ is a rule of the HPCFG and h is a lexical head, then add a rule $\langle C, h\rangle \rightarrow \langle B_1, C, h\rangle...\langle B_i, h\rangle...\langle B_n, C, h\rangle$. (Note the difference between head and non-head items on the right hand side of the rule.)
5. For each pair of categories A and B such that the HPCFG contains a rule $A \rightarrow ...B...$, where B is not the head, add a rule $\langle B, A, h\rangle \rightarrow \langle B, h'\rangle$ to the PCFG for each combination h and h' of lexical heads.
6. For each lexical entry of the form $A \rightarrow w$ and each lexical head h, add a lexical entry $\langle A, h\rangle \rightarrow w$ if h is equal to w (or if h is a stem of w given category A in case of lemmatisation, see section 2.4).

The above construction will return a PCFG which is not minimal, i.e. which contains rules that can not be used in building a well-formed parse tree. A verb phrase with `probabilistic` as lexical head, e.g., will never be part of a parse tree. In order to minimise the size of the PCFG, it is necessary to restrict the set of lexical heads to those which can actually project to a node of the respective category.

2.3 Unknown Words

LoPar considers all unknown words as a single token `<unknown>` which propagates `<unknown>` as lexical head. The word form probability $P_{lex}(\langle unknown\rangle|C, \langle unknown\rangle)$ is 1. In order to get a true probability model for the actual input string – as required in language modelling – $P_{lex}(\langle unknown\rangle|C, \langle unknown\rangle)$ has to be replaced by a probability distribution $P_{lex}(w|C, \langle unknown\rangle)$ which assigns probabilities to an infinite number of possible unknown words `w`.

A probability distribution for unknown words could be built as follows.

1. Train an n-gram language model $L_n(C)$ of character sequencies on all words (types, not tokens) of category C.
2. Compute the sum S of the probabilities which $L_n(C)$ assigns to the words of category C in the lexicon.
3. Divide the probabilities $P_{L_n(C)}(w)$ which $L_n(C)$ assigns to words by $1 - S$ in order to get the probability distribution $P_{lex}(w|C, \langle unknown\rangle)$.

2.4 Lemmatisation

One of the major problems in training HPCFGs is the large number of parameters which have to be estimated from a limited amount of training data. The number of parameters is reduced considerably if stems are used as lexical heads rather than inflected word forms, increasing the reliability of the parameter estimates. This is in particular true for languages with a rich morphology like German.

When the lexical heads are stems, the word form probability distribution $P_{lex}(w|C, h)$ is not trivial anymore because several word forms could have the same stem and part of speech (just assume that all numbers have the same stem). The P_{lex} parameters therefore have to be estimated from training data like other parameters.

Chapter 3

Parsing

LoPar is an implementation of the well-known *left-corner parsing* algorithm for context-free grammars (see e.g. [Graham et al., 1980]). The chart is organised as a two-dimensional array of lists. Each list is a linked list of edges. Each edge contains information about its start position, the grammar rule and the "dot" position (the position after the last daughter constituent which is covered by the edge). The first dimension of the array of lists corresponds to the end position of the edges, the second dimension corresponds to the category of the next item after the dot.

The chart representation maximises the efficiency of the *complete* operation of the left-corner parser. Whenever a new constituent has been found, the *completer* is invoced with the list of edges ending at the start position of the new constituent and expecting the new constituent.

Before a new edge is inserted into the chart, the parser has to check whether the edge has been inserted before. In order to avoid a scan of the whole list into which the new edge is to be inserted (usually a costly operation), the parser stores the edges (or rather pointers to them) additionally in a hash table. By means of the hash table it is possible to perform the check for duplicates in approximately constant time[1].

The recognised constituents are stored in a hash table, as well. Whenever a new complete edge has been found, a new constituent is inserted into the hash table unless a constituent of the same category and with the same span has been inserted before. In order to build the parse forest, the parser also adds a link from the constituent to the new edge. Similar links are added to the edges.

3.1 Parse Forest Representation

The parse forest, a compact representation of the parsing result, is internally stored in the following way (cmp. fig. 3.1): Each constituent of the parse forest either dominates a word form (terminal node) or a set of passive edges (nonterminal node). Each edge dominates a

[1] Of course, the runtime will be worse if the hashing function is unable to distribute the hashed items evenly. So far, this problem has not been observed.

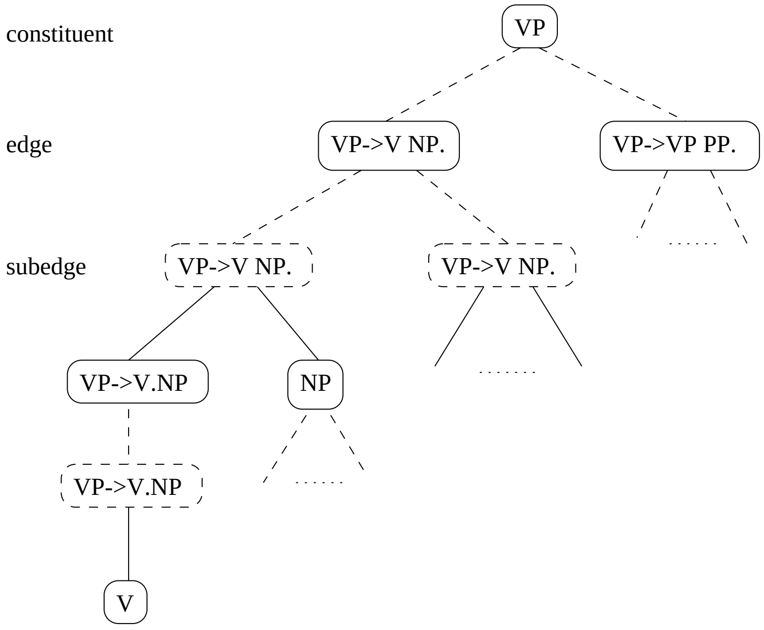

Figure 3.1: internal parse forest representation

list of *subedges*. Each subedge either dominates an (active) edge and a constituent or only a constituent if this constituent is the left-most daughter node.

3.2 Lexicalisation

In case of lexicalised parsing, the context-free parse forest has to be lexicalised after parsing. Ambiguous constituents may have more than one lexical head. Therefore the lexicalised parse forest will – in general – have a more complex structure than the unlexicalised parse forest.

LoPar computes the lexicalised parse forest in a single pass through the parse forest. Each node (constituent or edge) of the unlexicalised parse forest is linked to its lexicalised clones. Whenever a node of the unlexicalised parse forest is visited during lexicalisation, LoPar first lexicalises the child nodes recursively. Then it generates a new node for each combination of the lexicalised clones of its daughter nodes and inserts it into the new parse forest using the same hash table-based functions as for the generation of the unlexicalised parse forest. If the same node (same span, same category/dotted rule, same lexical head) has been inserted before, then, again, only a link to the new analysis is added. The unlexicalised parse forest is preserved during lexicalisation.

There is a problem, however: During the lexicalisation of subedges which do not cover the head of the rule, the lexical head is still unknown. The lexicalisation of these nodes therefore has to be delayed until the head information is available.

Chapter 4

Parameter Estimation

The parameters of lexicalised as well as unlexicalised probabilistic context-free grammars are iteratively estimated with the *Inside-Outside algorithm* [Lari and Young, 1990], which is an instance of the *Expectation-Maximisation* (EM) algorithm [Baum, 1972]. Each iteration of the Inside-Outside algorithm constists of two steps, namely frequency estimation and parameter estimation.

Lexicalised probability models are estimated with a bootstrapping approach. We first train the unlexicalised PCFG starting with a randomly initialised model. Then we generate lexicalised frequencies with the PCFG in order to obtain start values for the final lexicalised training.

4.1 Parameter Estimation for PCFGs

PCFGs encompass probability distributions for start events $P_{start}(C)$, rule events $P_{rule}(r)$ and lexical events $P_{lex}(w|C)$. The probabilities of these events are estimated based on estimated frequencies. The simplest estimation formula is the *Maximum-Likelihood-Estimation* (MLE) algorithm, which divides the frequency of an event by the sum of the frequencies of all competing events.

$$P(e) \quad = \quad \frac{F(e)}{\sum_e' F(e')}$$

MLE assigns zero probability to all unobserved events. A HPCFG has so many parameters, however, that many or even most of the corresponding events do not occur in the training corpus. These unobserved events should get a positive probability because a new corpus is likely to contain some of them. More sophisticated estimation techniques which "smooth" the probability distributions will be discussed in section 4.4. The estimation of the event frequencies is discussed in section 4.3.3.

4.1.1 Inside Probabilities

The inside probability of a *terminal constituent* `c` of category `C`, which expands to the word form `w` is defined as follows:

$$P_{inside}(c) \quad = \quad P_{rule}(\langle term \rangle | C) P_{lex}(w|C)$$

The inside probability of a *non-terminal constituent* of category `C`, is the sum over the inside probabilities of all (passive) edges which produce this constituent multiplied by the probabilities of the respective rules.

$$P_{inside}(c) \quad = \quad \sum_e P_{inside}(e) P(r|C)$$

The inside probability of an *edge* e is the sum over the inside probabilities of all its subedges s[1].

$$P_{inside}(e) \quad = \quad \sum_s P_{inside}(s)$$

The inside probability of a *subedge* s is the product of the inside probability of the daughter constituent c and the daughter edge e (if present, otherwise 1).

$$\begin{array}{lcll} P_{inside}(s) & = & P_{inside}(e)\ P_{inside}(c) & \text{if subedge e exists} \\ P_{inside}(s) & = & P_{inside}(c) & \text{otherwise} \end{array}$$

The *overall probability* of the parse forest is the sum over the inside probabilities of all root nodes `c` multiplied by the start symbol probabilities of the respective categories C:

$$P(T) \quad = \quad \sum_{root\ node\ c} P_{inside}(c)\ P_{start}(C)$$

4.1.2 Outside Probabilities

The outside probability of a *root node* of category `C`, is equal to the start symbol probability $P_{start}(C)$ of the respectice category `C`.

$$P_{outside}(c) \quad = \quad P_{start}(C)$$

The outside probabilities of the other events in the parse forest are initialised to zero and then computed while the parse forest is traversed in “prerequisite” order (see section 4.1.4).

For each *non-terminal constituent* `c` of category `C`, the parser runs through all passive (complete) edges `e` which produce this constituent and adds the product of the outside probability of `c` and the probability of the respective grammar rule `r` to the outside probability of `e`.

$$P_{outside}(e) \quad += \quad P_{outside}(c)\ P_{rule}(r|C)$$

[1] The parser does not actually store the subedge probabilities.

For each *edge* `e`, the parser runs through all subedges `s` and adds to the outside probability of the respective child constituent `c` the product of the outside probability of `e` times the inside probability of the child edge `e'` of subedge `s` (if present, otherwise 1).

$$\begin{array}{lcll} P_{outside}(c) & += & P_{outside}(e)\ P_{inside}(e') & \text{if e' exists} \\ P_{outside}(c) & += & P_{outside}(e) & \text{otherwise} \end{array}$$

Then we add the product of the outside probability of `e` times the inside probability of the child constituent `c` of `s` to the outside probability of the child edge *e'* (if present):

$$P_{outside}(e') \quad += \quad P_{outside}(e)\ P_{inside}(c)$$

4.1.3 Estimated Frequencies

The estimated frequency of a *start event* related to some root constituent `c` of category `C` is equal to the product of its inside probability and its outside probability divided by the overall probability $P(T)$. The start frequencies are accumulated in $F_{start}(C)$.

$$F_{start}(C) \quad += \quad P_{inside}(c)\ P_{outside}(c)\ /\ P(T)$$

The estimated frequency of a *rule event* `r` related to some passive edge `e` which is dominated by a constituent of category `C`, is equal to the product of its inside probability and its outside probability divided by the overall probability $P(T)$.

$$F_{rule}(r, C) \quad += \quad P_{inside}(e)\ P_{outside}(e)\ /\ P(T)$$

The estimated frequency of a *lexical event* related to some terminal constituent `c` of category `C` is equal to the product of its inside probability and its outside probability divided by the overall probability $P(T)$.

$$F_{lex}(w, C) \quad += \quad P_{inside}(c)\ P_{outside}(c)\ /\ P(T)$$

Finally, we derive the frequency that a constituent of category `C` is terminal.

$$F_{rule}(\langle term \rangle, C) \quad = \quad \sum_{w} F_{lex}(w, C)$$

4.1.4 Topological Sorting

In order to compute the outside probabilities, we have to traverse the parse forest in prerequisite order, i.e. before a node is processed, the processing of all parent nodes must have been completed. This ordering of the nodes is called a *topological ordering*.

The topological sorting algorithm presented in [Cormen et al., 1994] has been adapted for LoPar. It requires a counter for each node of the parse tree. The counters are initialised to 0. In a first pass, the parse forest is traversed top-down and whenever a node is visited, its counter is incremented. If the node is visited the first time (i.e. the counter value is 1), then its child nodes are visited, as well, from left to right.

After this first pass, the counters contain the number of parent nodes for each node. The outside probabilities are computed in a second top-down pass. Whenever a node is visited, its counter is decremented and – if the counter is 0 afterwards – the computations pertaining to this node are performed and the child nodes are visited.

4.2 Lexicalised Frequency Estimation with PCFGs

In the last iteration of unlexicalised training, the parser computes *lexicalised* frequencies in order to obtain start values for the subsequent training of the head-lexicalised PCFG. The computation parallels that for unlexicalised frequencies with the exception of the lexical choice frequencies $F_{choice}(h, C, C_p, h_p)$ which have no correspondence.

- lexicalised start event frequencies (`c` is a node with category `C` and lexical head `h`).

$$F_{start}(C,h) \quad += \quad P_{inside}(c)\ P_{outside}(c)\ /\ P(T)$$

- lexicalised rule event frequencies (`e` is a passive edge with category `C` on the left side of the rule and lexical head `h`).

$$F_{rule}(r,C,h) \quad += \quad P_{inside}(e)\ P_{outside}(e)\ /\ P(T)$$

- lexical choice event frequencies. These are related to subedges in the parse forest. C_p is the category on the left side of the corresponding rule, C is the category of the daughter constituent which must not be the head, and `e` is the left subedge (if present, otherwise $P_{inside}(e)$ is 1).

$$F_{choice}(h,C,C_p,h_p) \quad += \quad P_{outside}(s)\ P_{inside}(e)\ P_{inside}(c)\ /\ P(T)$$

- lexicalised word event frequencies

$$F_{lex}(w,C,h) \quad += \quad P_{inside}(c)\ P_{outside}(c)\ /\ P(T)$$

4.3 Frequency Estimation with HPCFGs

4.3.1 Inside Probabilities

The inside probabilities in a head-lexicalised parse forests are computed as follows:

- inside probabilities of terminal constituents:

$$P_{inside}(c) \quad = \quad P_{rule}(\langle term \rangle | C,h)\ P_{lex}(w|C,h)$$

- inside probability of *non-terminal constituents*:

$$P_{inside}(c) \quad = \quad \sum_e P_{inside}(e) P(r|C,h)$$

- inside probability of *edges*:

$$P_{inside}(e) \quad = \quad \sum_s P_{inside}(s)$$

- inside probability of *subedges* s where the daughter constituent c is the head:

$$\begin{array}{rcll} P_{inside}(s) & = & P_{inside}(e)\ P_{inside}(c) & \text{if subedge e exists} \\ P_{inside}(s) & = & P_{inside}(c) & \text{otherwise} \end{array}$$

- inside probability of *subedges* s where the daughter constituent c is *not* the head:

$$\begin{array}{rcll} P_{inside}(s) & = & P_{inside}(e)\ P_{choice}(h_d|C_d,C,h)\ P_{inside}(c) & \text{if subedge e exists} \\ P_{inside}(s) & = & P_{inside}(c)\ P_{choice}(h_d|C_d,C,h) & \text{otherwise} \end{array}$$

- overall probability of the parse forest:

$$P(T) \quad = \quad \sum_{root\ node\ c} P_{inside}(c)\ P_{start}(C)\ P_{start}(h|C)$$

4.3.2 Outside Probabilities

The outside probability of a *root node*:

$$P_{outside}(c) \quad = \quad P_{start}(C)\ P_{start}(h|C)$$

For each *non-terminal constituent* `c` of category `C`, the parser runs through all passive (complete) edges `e`:

$$P_{outside}(e) \quad += \quad P_{outside}(c)\ P_{rule}(r|C,h)$$

For each *edge* `e`, the parser runs through all subedges `s` and updates the outside probabilities of the non-head daughter constituents:

$$\begin{array}{rcll} P_{outside}(c) & += & P_{outside}(e)\ P_{inside}(e')\ P_{choice}(h_d|C_d,C,h) & \text{if e' exists} \\ P_{outside}(c) & += & P_{outside}(e)\ P_{choice}(h_d|C_d,C,h) & \text{otherwise} \end{array}$$

Then the parser updates the outside probabilities of the dominated edge e':

$$P_{outside}(e') \quad += \quad P_{outside}(e)\ P_{inside}(c)\ P_{choice}(h_d|C_d,C,h)$$

If the daughter constituent of e is the head then the parser adds

$$\begin{array}{rcll} P_{outside}(c) & += & P_{outside}(e)\ P_{inside}(e') & \text{if e' exists} \\ P_{outside}(c) & += & P_{outside}(e) & \text{otherwise} \end{array}$$

and

$$P_{outside}(e') \quad += \quad P_{outside}(e)\ P_{inside}(c)$$

4.3.3 Estimated Frequencies

The estimated frequency of a *lexicalised start event* related to some root constituent `c` of category `C` is computed as:

$$F_{start}(C,h) \quad += \quad P_{inside}(c)\ P_{outside}(c)\ /\ P(T)$$

The estimated frequency of a *lexicalised rule event* `r` related to some passive edge e:

$$F_{rule}(r,C,h) \quad += \quad P_{inside}(e)\ P_{outside}(e)\ /\ P(T)$$

The estimated frequency of a *lexical event* related to some constituent c:

$$F_{lex}(w,C,h) \quad += \quad P_{inside}(c)\ P_{outside}(c)\ /\ P(T)$$

The estimated frequency of a *lexical choice event* where c is the daughter constituent of some subedge s and not the head:

$$F_{choice}(h_d,C_d,C,h) \quad += \quad P_{outside}(e)\ P_{inside}(e')\ P_{inside}(c)\ P_{choice}(h_d|C_d,C,h)\ /\ P(T)$$

4.4 Parameter Smoothing

As mentioned earlier, it is necessary to assign a positive probability to events which are not observed in the training corpus. LoPar uses a variant of the absolute discounting method [Ney et al., 1994] for this purpose.

4.4.1 Absolute Discounting

The basic idea of absolute discounting is to subtract a small value (the discount) from all frequency counts and to redistribute the sum of these discounts over the events with zero frequency according to some *backoff* distribution. The absolute discounting method was adapted in order to be applicable to the real-valued counts generated by LoPar. The following method was implemented:

N_0 is the number of types whose frequency $F(t)$ is below 0.9.
N_1 is the number of types whose frequency $F(t)$ is between 0.9 and 1.9.
N_2 is the number of types whose frequency $F(t)$ is between 1.9 and 2.9.
N is the total frequency.

The *discount* d, which is subtracted from the positive counts, is computed according to the following formula presented in [Ney et al., 1994]:

$d := N_1/(N_1 + 2N_2)$

The *discounted frequencies* $F'(t)$ are obtained by subtracting the discount:

```
$D := 0$
for all types $t$ do
   if $F(t) > d$ then
      $F'(t) := F(t) - d$
      $D := D + d$
   else
      $F'(t) := 0$
      $D := D + F(t)$
```

The *probability* of a type t with a positive discounted frequency $F'(t)$ is then defined as:

$$P(t) := F'(t)/N$$

Finally, the *backoff factor* α is calculated.

$$\alpha = D/N/\sum_{t,F'(t)=0} P_{backoff}(t)$$

The probability of a type with a discounted frequency of 0 is then:

$$P(t) = \alpha\ P_{backoff}(t)$$

If the backoff probability distribution is uniform, then the probability of types with discounted frequency 0 is computed directly:

$$P(t) = D/N/N_0$$

If either N_1 or N_2 is 0 or if the resulting frequency for events with corpus frequency 0 is higher than that for events with corpus frequency 0.9, then a different smoothing method is used which replaces zero frequencies in the original counts by a small constant.

4.4.2 Backoff Distributions

LoPar implements the following backoff strategies.

- $P_{start}(h|C)$ is smoothed with a uniform backoff probability distribution $\hat{P}_{start}(h|C) = 1/hc(C)$, where $hc(C)$ is the number of possible heads of C.

- $P_{rule}(r|C,h)$ is smoothed with $P_{rule}(r|C)$ (the unlexicalised rule probability) as backoff distribution.

- $P_{choice}(h|C,C_p,h_p)$ is iteratively smoothed with $P_{choice}(h|C,C_p)$, $P_{choice}(h|C)$ and a uniform distribution $P_{choice}(h|C) = 1/hc(C)$ as backoff probabilities.

- $P_{lex}(w|C,h)$ is smoothed with a uniform backoff distribution $P_{lex}(w|C,h) = 1/wc(C,h)$, where $wc(C,H)$ is the number of lexical entries with category C and head (lemma) h.

4.4.3 Computation of the Number of Possible Heads

The number of possible lexical heads of the categories is computed with a bit-vector representation. The parser allocates a bit vector v_C with as many bits as there are lexical heads for each category C and initialises it to 0. Then it sets for each lexical entry with category C and lexical head h (either stem or word form) the corresponding bit $v_c[m]$ to 1. Then, it propagates the information by or-ing the bit vectors in bottom-up order. For each rule with C on the left side and C' as head category on the right side, the following operation is executed.

$$v_C := v_C \textbf{ or } v'_C$$

Once the bit vectors have been computed, the number of possible lexical heads is obtained by counting the number of bits in each vector.

4.5 Parameter Pooling

We have already discussed, how lemmatisation is used to reduce the number of parameters of a HPCFG. Another way to achieve a reduction is *parameter pooling.* Consider the rule

```
VP_fin_past -> VP_fin_past ADV
```

which adjoins an adverb to a verb phrase. The lexical choice distribution $P_{choice}(adv|ADV, VP\_fin\_past, verb)$ is unlikely to differ very much if we replace `VP_fin_past` by `VP_fin_pres` or `VP_inf_past`. Therefore, we would like to pool the corresponding distributions into one distribution $P_{choice}(adv|ADV, VP\_fin\_past|VP\_fin\_pres|..., verb)$ in order to get more reliable estimates.

But, what does the parameter pooling mean for the probability model? Remember that each HPCFG has an equivalent PCFG. The same is true for HPCFGs with parameter pooling. We just modify step 5 of the grammar transformation in section 2.2 in the following way:

5 For each pair of categories `A` and `B` such that the HPCFG contains a rule $A \rightarrow ...B...$, where `B` is not the head, add a rule $\langle B, A, h\rangle \rightarrow \langle B, |A|, h\rangle$ to the PCFG for each lexical head `h`, where $|A|$ is the pooling class of A.

Further add a rule $\langle B, |A|, h\rangle \rightarrow \langle B, h'\rangle$ for each lexical head `h'`.

In case of parameters pooling, the lexical choice probability $P_{choice}(h \,|\, C, C_p, h_p)$ is replaced by $P_{choice}(h \,|\, C_c, |C_p|, h_p)$, where $|C_p|$ is the pooling class of category C_p.

4.6 Logarithmic Computation

The computation of the inside and outside probabilities poses a numerical problem. For large sentences, the probability of the whole parse forest may drop below the smallest numerical value which the computer is able to represent. The usual double-precision floating point numbers cannot represent numbers smaller than about 10^{-308}. Lopar uses therefore a logarithmic representation of the probabilities.

For the Viterbi algorithm used to extract the most probable parse tree, this means that the product of probabilities has to be replaced by the sum of logarithmic probabilities.

The implementation of the inside-outside algorithm with logarithmic probabilities is more difficult because it also involves summation, which has no equivalent in the logarithmic domain. LoPar uses the following function to add logarithmic numbers:

```
add($x, y$)
/* x and y are logarithmic numbers */
if ($x < y - log(10^{-30})$)
   return $y$
if ($x - log(10^{-30}) > y$)
   return $x$
$base := min(x, y)$
return $base + log(exp(x - base) + exp(y - base))$
```

Chapter 5

Working with LoPar

5.1 Grammar Format

A grammar for LoPar consists of four files.

- the grammar file
- the lexicon file
- the start symbol file
- the open-class category file

5.1.1 The Grammar File

The grammar file contains – little surprise – the grammar. Each line starts with a number which is interpreted as the rule frequency. Then follows the category on the left side of the rule and the sequence of daughter categories. One daughter category has to be marked as the head with an apostrophy (').

Example:

```
11.3 S NP VP'
30.2 NP DT N'
...
```

The head marking is optional if neither option `-heads` nor option `-lexmodel` is specified. The frequencies in the grammar file are mandatory unless option `-symbolic` is chosen.

5.1.2 The Lexicon File

Each line of the lexicon file contains lexical entries for one word form. The line starts with the word form which may contain blanks. The word form is followed by a tab character and a sequence of category/frequency pairs. If the parser option -stems was specified, then the parser expects a stem after the frequency.

The specification of the frequency is mandatory unless option -symbolic is chosen.

Example (with stems):

```
because of      PREP 13.1 because_of
can     AUX 113 can NSG 7 can VFIN 0 can VINF 0 can
saw     NSG 9 saw VFIN 67 see VFIN 2 saw VINF 0 saw
...
```

The parser is able to parse without a lexicon if the parser input contains tags.

5.1.3 The Start Symbol File

The start symbol file contains the list of categories which are allowed at the root of a parse tree.

Example:

```
S  1000
NP 10
X  1
```

All categories are possible start symbols if no start symbol file is specified.

5.1.4 The Open-Class Categories

In order to parse unrestricted input, the parser must be able to deal with unknown words. The open-class category file contains the list of possible unknown word categories. It is then assumed that the lexicon is complete wrt. closed-class words like prepositions, conjunctions, determiners, etc.

Example:

```
N   100
V   1
ADJ 10
ADV 10
```

5.1.5 Input File

The parser input must have one-word-per-line format, i.e. each input line corresponds to one input token. The end of a sentence has to be marked with an empty line. The input may contain part-of-speech tags after the token and a tab character. If an input line contains tags, then these are the only tags which will be considered during parsing. Otherwise the set of possible tags of a token is obtained from the lexicon.

5.2 Symbolic Parsing

LoPar can be used for purely symbolic parsing as the following command illustrates.

```
lopar grammar text text.parsed
```

Because no lexicon was specified, the parts of speech of the words must be given in the input.

In the above example, the parser will print the parse forests for the sentences in `text` to the file `text.parsed`. Each parse forest contains one line for each constituent in the parse forest. The line starts with the category name followed by the start and end position and the list of analyses of the constituent. Each analysis either consists of the word form (in case of a terminal node) or of the rule number, the sequence of daughter node IDs and a percent sign. The daughter node ID is the number of the line in which the correponding constituent is printed (starting at 0). Each line ends with two percent signs and the parse forest ends with three percent signs.

Example:

```
NP 0 2  2 1 2 %%
DT 0 1  the %%
NBAR 1 2  4 3 %%
N 1 2  man %%%
```

It is possible to pipe input into the parser and out of the parser as the following command shows.

```
cat text | lopar grammar > text.parsed
```

5.3 Training

The training of a HPCFG proceeds in three steps.

- unlexicalised training (several iterations on text chunks of growing size)
- lexicalisation (one iteration on the whole training corpus)

- lexicalised training (several iterations on the whole corpus)

Two iterations of unlexicalised training on a few thousand sentences are usually sufficient to obtain a good basis for the lexicalisation of the grammar. The following commands might be used for this purpose:

```
head -10000 corpus | lopar -in param -t param1
head -10000 corpus | lopar -in param1 -t param1
```

Option `-in param` means that the grammar is to be read from `param.gram`, the lexicon from `param.lex`, the start categories from `param.start` and the open-class tags from `param.oc`.

The `-iter N` option allows to perform the unlexicalised training in one step. With this option, the parser first trains on `N` tokens, then on `2N` tokens, then on `4N` tokens and so on until the whole corpus has been parsed.

```
head -20000 corpus | lopar -in param -t param1 -iter 10000
```

After parsing a multiple of 10,000 tokens and after parsing the whole input corpus, the parser saves the parameters in files named `param1.gram`, `param1.lex`, `param1.start` and `param1.oc`.

Now, we can lexicalise the grammar with the command:

```
lopar -in param1 -t param2 -heads corpus
```

The lexicalised grammar comprises three additional files, namely `param2.lstart` with the lexicalised start frequencies, `param2.lgram` with the lexicalised grammar rule frequencies and `param2.lchoice` with the lexical choice frequencies.

After a few iterations of lexicalised training with the command

```
lopar -in param2 -t param2 -lexmodel corpus
```

the training is finished. Now, we are ready to parse a test corpus with the command:

```
lopar -in param2 -lexmodel corpus parse-forests
```

Viterbi parses are obtained with the command:

```
lopar -in param2 -lexmodel -viterbi corpus parse-trees
```

5.4 Tagging with LoPar

LoPar computes the most probable sequence of tags (terminal categories) for a sentence by parsing the sentence and then extracting the tags with the highest estimated frequency. The corresponding program option is `-tagging`.

5.5 Chunking with LoPar

Some applications (e.g. in information retrievel) need low-level constituents like NPs and PPs rather than full parses. If LoPar is called with the option `-chunking <file>`, then it will extract the most probable sequence of chunks. The set of chunk categories is listed in `<file>`.

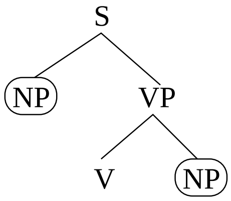

Figure 5.1: A partial parse tree. The chunks are encircled.

The most probable chunk sequence of a parse forest `P` is defined as the sequence of non-word leaf nodes of the partial parse tree `T` which satisfies the following conditions:

1. `T` is a subgraph of the parse forest `P`.
2. The root node of `T` is also a root node of `P`.
3. The leaf nodes of `T` are either words or nodes labelled with a chunk category.
4. `T` has a higher score $S(T)$ than any other partial parse tree satisfying 1.) and 2.).

The score $S(T)$ is defined as the sum of the probabilities of all parse trees which match `T`. It is efficiently computed as the (Viterbi) probability of the (partial parse tree) `T` times the product of the inside probabilities of all its terminal (chunk) nodes.

The score of the partial parse tree in fig. 5.1 is the probability of the partial tree itself times the inside probability of the sub-parse forest rooted in the first NP times the inside probability of the sub-parse forest rooted in the second NP.

Problem: The presented algorithm prefers recursive chunks. The reason is the following: As the size of the partial parse tree `T` grows, the number of matching complete parse trees decreases and hence the score of the partial parse tree decreases.

In order to get only non-recursive chunks, we replace the inside probability of a noun chunk `C` in the computation of the score by the sum of the probabilities of all subtrees rooted in `C` and *not* containing any chunk category labels.

Unsolved problem: The chunking algorithm does not sum over all analyses containing the chunk sequence. Consider e.g. the well-known sentence *I saw the man on the hill with the telescope.* It is possible to efficiently compute the probability of all parse trees exactly containing a particular chunk sequence, but there are exponentially many of them, so that enumerating them is too costly. A heuristic search strategy which examines only likely candidate sequences might help but has not been implemented so far.

Bibliography

[Baum, 1972] Baum, L. E. (1972). An inequality and association maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8.

[Carroll, 1995] Carroll, G. (1995). *Learning Probabilistic Grammars for Language Modelling*. PhD thesis, Department of Computer Science, Brown University.

[Carroll and Rooth, 1998] Carroll, G. and Rooth, M. (1998). Valence induction with a head-lexicalized PCFG. In *Proceedings of Third Conference on Empirical Methods in Natural Language Processing*, Granada, Spain. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9805001.

[Cormen et al., 1994] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1994). *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts.

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

[Hindle and Rooth, 1993] Hindle, D. M. and Rooth, M. (1993). Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1).

[Lari and Young, 1990] Lari, K. and Young, S. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computation Speech and Language Processing*, 4:35–56.

[Ney et al., 1994] Ney, H., Essen, U., and Kneser, R. (1994:). On structuring probabilistic dependencies in stochastic language modelling. *Computer Speech and Language*, 8:1–38.