Efficacy of Beam Thresholding, Unification Filtering and Hybrid Parsing in Probabilistic HPSG Parsing

Takashi Ninomiya
CREST, JST
and
Department of Computer Science
The University of Tokyo
ninomi@is.s.u-tokyo.ac.jp

Yoshimasa Tsuruoka
CREST, JST
and
Department of Computer Science
The University of Tokyo
tsuruoka@is.s.u-tokyo.ac.jp

Yusuke Miyao
Department of Computer Science
The University of Tokyo
yusuke@is.s.u-tokyo.ac.jp

Jun'ichi Tsujii
Department of Computer Science
The University of Tokyo
and
School of Informatics
University of Manchester
and
CREST, JST
tsujii@is.s.u-tokyo.ac.jp

Abstract

We investigated the performance efficacy of beam search parsing and deep parsing techniques in probabilistic HPSG parsing using the Penn treebank. We first tested the beam thresholding and iterative parsing developed for PCFG parsing with an HPSG. Next, we tested three techniques originally developed for deep parsing: quick check, large constituent inhibition, and hybrid parsing with a CFG chunk parser. The contributions of the large constituent inhibition and global thresholding were not significant, while the quick check and chunk parser greatly contributed to total parsing performance. The precision, recall and average parsing time for the Penn treebank (Section 23) were 87.85%, 86.85%, and 360 ms, respectively.

1 Introduction

We investigated the performance efficacy of beam search parsing and deep parsing techniques in probabilistic head-driven phrase structure grammar (HPSG) parsing for the Penn treebank. We first applied beam thresholding techniques developed for CFG parsing to HPSG parsing, including local thresholding, global thresholding (Goodman, 1997), and iterative parsing (Tsuruoka and Tsujii, 2005b). Next, we applied parsing techniques developed for deep parsing, including quick check (Malouf et al., 2000), large constituent inhibition (Kaplan et al., 2004) and hybrid parsing with a CFG chunk parser (Daum et al., 2003; Frank et al., 2003; Frank, 2004). The experiments showed how each technique contributes to the final output of parsing in terms of precision, recall, and speed for the Penn treebank.

Unification-based grammars have been extensively studied in terms of linguistic formulation and computation efficiency. Although they provide precise linguistic structures of sentences, their processing is considered expensive because of the detailed descriptions. Since efficiency is of particular concern in practical applications, a number of studies have focused on improving the parsing efficiency of unification-based grammars (Oepen et al., 2002). Although significant improvements in efficiency have been made, parsing speed is still not high enough for practical applications.

The recent introduction of probabilistic models of wide-coverage unification-based grammars (Malouf and van Noord, 2004; Kaplan et al., 2004; Miyao and Tsujii, 2005) has opened up the novel possibility of increasing parsing speed by guiding the search path using probabilities. That is, since we often require only the most probable parse result, we can compute partial parse results that are likely to contribute to the final parse result. This approach has been extensively studied in the field of probabilistic

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 103–114,
Vancouver, October 2005.

CFG (PCFG) parsing, such as Viterbi parsing and beam thresholding.

While many methods of probabilistic parsing for unification-based grammars have been developed, their strategy is to first perform exhaustive parsing without using probabilities and then select the highest probability parse. The behavior of their algorithms is like that of the Viterbi algorithm for PCFG parsing, so the correct parse with the highest probability is guaranteed. The interesting point of this approach is that, once the exhaustive parsing is completed, the probabilities of non-local dependencies, which cannot be computed during parsing, are computed after making a packed parse forest. Probabilistic models where probabilities are assigned to the CFG backbone of the unification-based grammar have been developed (Kasper et al., 1996; Briscoe and Carroll, 1993; Kiefer et al., 2002), and the most probable parse is found by PCFG parsing. This model is based on PCFG and not probabilistic unification-based grammar parsing. Geman and Johnson (Geman and Johnson, 2002) proposed a dynamic programming algorithm for finding the most probable parse in a packed parse forest generated by unification-based grammars without expanding the forest. However, the efficiency of this algorithm is inherently limited by the inefficiency of exhaustive parsing.

In this paper we describe the performance of beam thresholding, including iterative parsing, in probabilistic HPSG parsing for a large-scale corpora, the Penn treebank. We show how techniques developed for efficient deep parsing can improve the efficiency of probabilistic parsing. These techniques were evaluated in experiments on the Penn Treebank (Marcus et al., 1994) with the wide-coverage HPSG parser developed by Miyao et al. (Miyao et al., 2005; Miyao and Tsujii, 2005).

2 HPSG and probabilistic models

HPSG (Pollard and Sag, 1994) is a syntactic theory based on lexicalized grammar formalism. In HPSG, a small number of schemata describe general construction rules, and a large number of lexical entries express word-specific characteristics. The structures of sentences are explained using combinations of schemata and lexical entries. Both schemata and lexical entries are represented by typed feature structures, and constraints represented by feature structures are checked with *unification*.

Figure 1 shows an example of HPSG parsing of the sentence "*Spring has come.*" First, each of the lexical entries for "*has*" and "*come*" is unified with a daughter feature structure of the Head-Complement

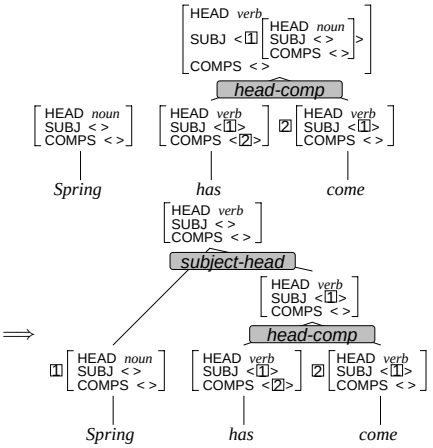

Figure 1: HPSG parsing

Schema. Unification provides the phrasal sign of the mother. The sign of the larger constituent is obtained by repeatedly applying schemata to lexical/phrasal signs. Finally, the parse result is output as a phrasal sign that dominates the sentence.

Given set $\mathcal{W}$ of words and set $\mathcal{F}$ of feature structures, an HPSG is formulated as a tuple, $G = \langle L, R \rangle$, where

$L = \{l = \langle w, F \rangle | w \in \mathcal{W}, F \in \mathcal{F}\}$ is a set of lexical entries, and
R is a set of schemata, i.e., $r \in R$ is a partial function: $\mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}$.

Given a sentence, an HPSG computes a set of phrasal signs, i.e., feature structures, as a result of parsing.

Previous studies (Abney, 1997; Johnson et al., 1999; Riezler et al., 2000; Miyao et al., 2003; Malouf and van Noord, 2004; Kaplan et al., 2004; Miyao and Tsujii, 2005) defined a probabilistic model of unification-based grammars as a *log-linear model* or *maximum entropy model* (Berger et al., 1996). The probability of parse result T assigned to given sentence $\mathbf{w} = \langle w_1, \ldots, w_n \rangle$ is

$$p(T|\mathbf{w}) = \frac{1}{Z_{\mathbf{w}}} \exp\left(\sum_i \lambda_i f_i(T)\right)$$

$$Z_{\mathbf{w}} = \sum_{T'} \exp\left(\sum_i \lambda_i f_i(T')\right),$$

where λ_i is a model parameter, and f_i is a feature function that represents a characteristic of parse tree T. Intuitively, the probability is defined as the normalized product of the weights $\exp(\lambda_i)$ when a characteristic corresponding to f_i appears in parse result T. Model parameters λ_i are estimated using numer-

ical optimization methods (Malouf, 2002) so as to maximize the log-likelihood of the training data.

However, the above model cannot be easily estimated because the estimation requires the computation of $p(T|\mathbf{w})$ for all parse candidates assigned to sentence $\mathbf{w}$. Because the number of parse candidates is exponentially related to the length of the sentence, the estimation is intractable for long sentences.

To make the model estimation tractable, Geman and Johnson (Geman and Johnson, 2002) and Miyao and Tsujii (Miyao and Tsujii, 2002) proposed a dynamic programming algorithm for estimating $p(T|\mathbf{w})$. They assumed that features are functions on nodes in a packed parse forest. That is, parse tree T is represented by a set of nodes, i.e., $T = \{c\}$, and the parse forest is represented by an and/or graph of the nodes. From this assumption, we can redefine the probability as

$$p(T|\mathbf{w}) = \frac{1}{Z_{\mathbf{w}}} \exp\left(\sum_{c \in T} \sum_{i} \lambda_i f_i(c)\right)$$

$$Z_{\mathbf{w}} = \sum_{T'} \exp\left(\sum_{c \in T'} \sum_{i} \lambda_i f_i(c)\right).$$

A packed parse forest has a structure similar to a chart of CFG parsing, and c corresponds to an edge in the chart. This assumption corresponds to the independence assumption in PCFG; that is, only a nonterminal symbol of a mother is considered in further processing by ignoring the structure of its daughters. With this assumption, we can compute the figures of merit (FOMs) of partial parse results.

This assumption restricts the possibility of feature functions that represent non-local dependencies expressed in a parse result. Since unification-based grammars can express semantic relations, such as predicate-argument relations, in their structure, the assumption unjustifiably restricts the flexibility of probabilistic modeling. However, previous research (Miyao et al., 2003; Clark and Curran, 2004; Kaplan et al., 2004) showed that predicate-argument relations can be represented under the assumption of feature locality. We thus assumed the locality of feature functions and exploited it for the efficient search of probable parse results.

3 Techniques for efficient deep parsing

Many of the techniques for improving the parsing efficiency of deep linguistic analysis have been developed in the framework of lexicalized grammars such as lexical functional grammar (LFG) (Bresnan, 1982), lexicalized tree adjoining grammar (LTAG) (Shabes et al., 1988), HPSG (Pollard and Sag, 1994) or combinatory categorial grammar (CCG) (Steedman, 2000). Most of them were developed for exhaustive parsing, i.e., producing all parse results that are given by the grammar (Matsumoto et al., 1983; Maxwell and Kaplan, 1993; van Noord, 1997; Kiefer et al., 1999; Malouf et al., 2000; Torisawa et al., 2000; Oepen et al., 2002; Penn and Munteanu, 2003). The strategy of exhaustive parsing has been widely used in grammar development and in parameter training for probabilistic models.

We tested three of these techniques.

Quick check Quick check filters out non-unifiable feature structures (Malouf et al., 2000). Suppose we have two non-unifiable feature structures. They are destructively unified by traversing and modifying them, and then finally they are found to be not unifiable in the middle of the unification process. Quick check quickly judges their unifiability by peeping the values of the given paths. If one of the path values is not unifiable, the two feature structures cannot be unified because of the necessary condition of unification. In our implementation of quick check, each edge had two types of arrays. One contained the path values of the edge's sign; we call this the sign array. The other contained the path values of the right daughter of a schema such that its left daughter is unified with the edge's sign; we call this a schema array. When we apply a schema to two edges, e_1 and e_2, the schema array of e_1 and the sign array of e_2 are *quickly* checked. If it fails, then quick check returns a unification failure. If it succeeds, the signs are unified with the schemata, and the result of unification is returned.

Large constituent inhibition (Kaplan et al., 2004) It is unlikely for a large medial edge to contribute to the final parsing result if it spans more than 20 words and is not adjacent to the beginning or ending of the sentence. Large constituent inhibition prevents the parser from generating medial edges that span more than some word length.

HPSG parsing with a CFG chunk parser A hybrid of deep parsing and shallow parsing was recently found to improve the efficiency of deep parsing (Daum et al., 2003; Frank et al., 2003; Frank, 2004). As a preprocessor, the shallow parsing must be very fast and achieve high precision but not high recall so that the

```
procedure Viterbi(⟨w_1, ..., w_n⟩, ⟨L', R⟩, κ, δ, θ)
  for i = 1 to n
    foreach F_u ∈ {F|⟨w_i, F⟩ ∈ L}
      α = Σ_i λ_i f_i(F_u)
      π[i−1, i] ← π[i−1, i] ∪ {F_u}
      if (α > ρ[i−1, i, F_u]) then
        ρ[i−1, i, F_u] ← α
  for d = 1 to n
    for i = 0 to n − d
      j = i + d
      for k = i + 1 to j − 1
        foreach F_s ∈ π[i, k], F_t ∈ π[k, j], r ∈ R
          if F = r(F_s, F_t) has succeeded
            α = ρ[i, k, F_s] + ρ[k, j, F_t] + Σ_i λ_i f_i(F)
            π[i, j] ← π[i, j] ∪ {F}
            if (α > ρ[i, j, F]) then
              ρ[i, j, F] ← α
```

Figure 2: Pseudo-code of Viterbi algorithms for probabilistic HPSG parsing

total parsing performance in terms of precision, recall and speed is not degraded. Because there is trade-off between speed and accuracy in this approach, the total parsing performance for large-scale corpora like the Penn treebank should be measured. We introduce a CFG chunk parser (Tsuruoka and Tsujii, 2005a) as a preprocessor of HPSG parsing. Chunk parsers meet the requirements for preprocessors; they are very fast and have high precision. The grammar for the chunk parser is automatically extracted from the CFG treebank translated from the HPSG treebank, which is generated during grammar extraction from the Penn treebank. The principal idea of using the chunk parser is to use the bracket information, i.e., parse trees without non-terminal symbols, and prevent the HPSG parser from generating edges that cross brackets.

4 Beam thresholding for HPSG parsing

4.1 Simple beam thresholding

Many algorithms for improving the efficiency of PCFG parsing have been extensively investigated. They include grammar compilation (Tomita, 1986; Nederhof, 2000), the Viterbi algorithm, controlling search strategies without FOM such as left-corner parsing (Rosenkrantz and Lewis II, 1970) or head-corner parsing (Kay, 1989; van Noord, 1997), and with FOM such as the beam search, the best-first search or A* search (Chitrao and Grishman, 1990; Caraballo and Charniak, 1998; Collins, 1999; Ratnaparkhi, 1999; Charniak, 2000; Roark, 2001; Klein and Manning, 2003). The beam search and best-first search algorithms significantly reduce the time required for finding the best parse at the cost of losing the guarantee of finding the correct parse.

The CYK algorithm, which is essentially a bottom-up parser, is a natural choice for non-probabilistic HPSG parsers. Many of the constraints are expressed as lexical entries in HPSG, and bottom-up parsers can use those constraints to reduce the search space in the early stages of parsing.

For PCFG, extending the CYK algorithm to output the Viterbi parse is straightforward (Ney, 1991; Jurafsky and Martin, 2000). The parser can efficiently calculate the Viterbi parse by taking the maximum of the probabilities of the same nonterminal symbol in each cell. With the probabilistic model defined in Section 2, we can also define the Viterbi search for unification-based grammars (Geman and Johnson, 2002). Figure 2 shows the pseudo-code of Viterbi algorithm. The $\pi[i, j]$ represents the set of partial parse results that cover words $w_{i+1}, \ldots, w_j$, and $\rho[i, j, F]$ stores the maximum FOM of partial parse result F at cell (i, j). Feature functions are defined over lexical entries and results of rule applications, which correspond to conjunctive nodes in a feature forest. The FOM of a newly created partial parse, F, is computed by summing the values of ρ of the daughters and an additional FOM of F.

The Viterbi algorithm enables various pruning techniques to be used for efficient parsing. Beam thresholding (Goodman, 1997) is a simple and effective technique for pruning edges during parsing. In each cell of the chart, the method keeps only a portion of the edges which have higher FOMs compared to the other edges in the same cell.

```
procedure BeamThresholding(⟨w_1, ..., w_n⟩, ⟨L', R⟩, κ, δ, θ)
    for i = 1 to n
        foreach F_u ∈ {F|⟨w_i, F⟩ ∈ L}
            α = Σ_i λ_i f_i(F_u)
            π[i − 1, i] ← π[i − 1, i] ∪ {F_u}
            if (α > ρ[i − 1, i, F_u]) then
                ρ[i − 1, i, F_u] ← α
    for d = 1 to n
        for i = 0 to n − d
            j = i + d
            for k = i + 1 to j − 1
                foreach F_s ∈ π[i, k], F_t ∈ π[k, j], r ∈ R
                    if F = r(F_s, F_t) has succeeded
                        α = ρ[i, k, F_s] + ρ[k, j, F_t] + Σ_i λ_i f_i(F)
                        π[i, j] ← π[i, j] ∪ {F}
                        if (α > ρ[i, j, F]) then
                            ρ[i, j, F] ← α
            LocalThresholding(κ, δ)
        GlobalThresholding(n, θ)
```

```
procedure LocalThresholding(κ, δ)
    sort π[i, j] according to ρ[i, j, F]
    π[i, j] ← {π[i, j]_1, ..., π[i, j]_κ}
    α_max = max_F ρ[i, j, F]
    foreach F ∈ π[i, j]
        if ρ[i, j, F] < α_max − δ
            π[i, j] ← π[i, j]\{F}
```

```
procedure GlobalThresholding(n, θ)
    f[0..n] ← {0, −∞ − ∞, ..., −∞}
    b[0..n] ← {−∞, −∞, ..., −∞, 0}
    #forward
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                f[j] ← max(f[j], f[i] + ρ[i, j, F])
    #backward
    for i = n − 1 to 0
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                b[i] ← max(b[i], b[j] + ρ[i, j, F])
    #global thresholding
    α_max = f[n]
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                if f[i] + ρ[i, j, F] + b[j] < α_max − θ then
                    π[i, j] ← π[i, j]\{F}
```

Figure 3: Pseudo-code of local beam search and global beam search algorithms for probabilistic HPSG parsing

```
procedure IterativeBeamThresholding(w, G, κ0, δ0, θ0, Δκ, Δδ, Δθ, κlast, δlast, θlast)
  κ ← κ0; δ ← δ0; θ ← θ0
  loop while κ ≤ κlast and δ ≤ δlast and θ ≤ θlast
    call BeamThresholding(w, G, κ, δ, θ)
    if π[1, n] ≠ ∅ then exit
    κ ← κ + Δκ; δ ← δ + Δδ; θ ← θ + Δθ
```

Figure 4: Pseudo-code of iterative beam thresholding

We tested three selection schemes for deciding which edges to keep in each cell.

Local thresholding by number of edges Each cell keeps the top κ edges based on their FOMs.

Local thresholding by beam width Each cell keeps the edges whose FOM is greater than $\alpha_{\max} - \delta$, where $\alpha_{\max}$ is the highest FOM among the edges in the cell.

Global thresholding by beam width Each cell keeps the edges whose *global FOM* is greater than $\alpha_{\max} - \theta$, where $\alpha_{\max}$ is the highest global FOM in the chart.

Figure 3 shows the pseudo-code of local beam search, and global beam search algorithms for probabilistic HPSG parsing. The code for local thresholding is inserted at the end of the computation for each cell. In Figure 3, $\pi[i,j]_k$ denotes the k-th element in sorted set $\pi[i,j]$. We first take the first κ elements that have higher FOMs and then remove the elements with FOMs lower than $\alpha_{\max} - \delta$.

Global thresholding is also used for pruning edges, and was originally proposed for CFG parsing (Goodman, 1997). It prunes edges based on their global FOM and the best global FOM in the chart. The global FOM of an edge is defined as its FOM plus its forward and backward FOMs, where the forward and backward FOMs are rough estimations of the outside FOM of the edge. The global thresholding is performed immediately after each line of the CYK chart is completed. The forward FOM is calculated first, and then the backward FOM is calculated. Finally, all edges with a global FOM lower than $\alpha_{\max} - \theta$ are pruned. Figure 3 gives further details of the algorithm.

4.2 Iterative beam thresholding

We tested the iterative beam thresholding proposed by Tsuruoka and Tsujii (2005b). We started the parsing with a narrow beam. If the parser output results, they were taken as the final parse results. If the parser did not output any results, we widened the beam, and reran the parsing. We continued widening the beam until the parser output results or the beam width reached some limit.

Table 1: Abbreviations used in experimental results

num	local beam thresholding by number
width	local beam thresholding by width
global	global beam thresholding by width
iterative	iterative parsing with local beam thresholding by number and width
chp	parsing with CFG chunk parser

The pseudo-code is presented in Figure 4. It calls the beam thresholding procedure shown in Figure 3 and increases parameters κ and δ until the parser outputs results, i.e., $\pi[1,n] \neq \emptyset$.

Preserved iterative parsing Our implemented CFG parser with iterative parsing cleared the chart and edges at every iteration although the parser regenerated the same edges using those generated in the previous iteration. This is because the computational cost of regenerating edges is smaller than that of reusing edges to which the rules have already been applied. For HPSG parsing, the regenerating cost is even greater than that for CFG parsing. In our implementation of HPSG parsing, the chart and edges were not cleared during the iterative parsing. Instead, the pruned edges were marked as thresholded ones. The parser counted the number of iterations, and when edges were generated, they were marked with the iteration number, which we call the *generation*. If edges were thresholded out, the generation was replaced with the current iteration number plus 1. Suppose we have two edges, e_1 and e_2. The grammar rules are applied iff both e_1 and e_2 are not thresholded out, and the generation of e_1 or e_2 is equal to the current iteration number. Figure 5 shows the pseudo-code of preserved iterative parsing.

```
procedure BeamThresholding(⟨w_1, ..., w_n⟩, ⟨L', R⟩, κ, δ, θ, iternum)
    for i = 1 to n
        foreach F_u ∈ {F|⟨w_i, F⟩ ∈ L}
            α = Σ_i λ_i f_i(F_u)
            π[i−1, i] ← π[i−1, i] ∪ {F_u}
            if (α > ρ[i−1, i, F_u]) then
                ρ[i−1, i, F_u] ← α
    for d = 1 to n
        for i = 0 to n − d
            j = i + d
            for k = i + 1 to j − 1
                foreach F_s ∈ φ[i, k], F_t ∈ φ[k, j], r ∈ R
                    if gen[i, k, F_s] = iternum ∨ gen[k, j, F_t] = iternum
                        if F = r(F_s, F_t) has succeeded
                            gen[i, j, F] ← iternum
                            α = ρ[i, k, F_s] + ρ[k, j, F_t] + Σ_i λ_i f_i(F)
                            π[i, j] ← π[i, j] ∪ {F}
                            if (α > ρ[i, j, F]) then
                                ρ[i, j, F] ← α
            LocalThresholding(κ, δ, iternum)
        GlobalThresholding(n, θ, iternum)
```

```
procedure LocalThresholding(κ, δ, iternum)
    sort π[i, j] according to ρ[i, j, F]
    φ[i, j] ← {π[i, j]_1, ..., π[i, j]_κ}
    α_max = max_F ρ[i, j, F]
    foreach F ∈ φ[i, j]
        if ρ[i, j, F] < α_max − δ
            φ[i, j] ← φ[i, j]\{F}
    foreach F ∈ (π[i, j] − φ[i, j])
        gen[i, j, F] ← iternum + 1
```

```
procedure GlobalThresholding(n, θ, iternum)
    f[0..n] ← {0, −∞ − ∞, ..., −∞}
    b[0..n] ← {−∞, −∞, ..., −∞, 0}
    #forward
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                f[j] ← max(f[j], f[i] + ρ[i, j, F])
    #backward
    for i = n − 1 to 0
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                b[i] ← max(b[i], b[j] + ρ[i, j, F])
    #global thresholding
    α_max = f[n]
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ φ[i, j]
                if f[i] + ρ[i, j, F] + b[j] < α_max − θ then
                    φ[i, j] ← φ[i, j]\{F}
            foreach F ∈ (π[i, j] − φ[i, j])
                gen[i, j, F] ← iternum + 1
```

```
procedure IterativeBeamThresholding(w, G, κ_0, δ_0, θ_0, Δκ, Δδ, Δθ, κ_last, δ_last, θ_last)
    κ ← κ_0; δ ← δ_0; θ ← θ_0; iternum = 0
    loop while κ ≤ κ_last and δ ≤ δ_last and θ ≤ θ_last
        call BeamThresholding(w, G, κ, δ, θ, iternum)
        if π[1, n] ≠ ∅ then exit
        κ ← κ + Δκ; δ ← δ + Δδ; θ ← θ + Δθ; iternum ← iternum + 1
```

Figure 5: Pseudo-code of preserved iterative parsing for HPSG

Table 2: Experimental results for development set (section 22) and test set (section 23)

	Precision	Recall	F-score	Avg. Time (ms)	No. of failed sentences
development set	88.21%	87.32%	87.76%	360	12
test set	87.85%	86.85%	87.35%	360	15

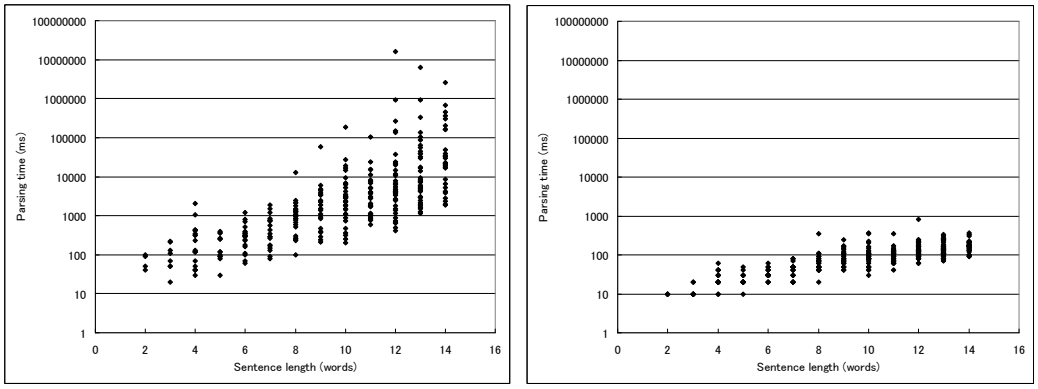

Figure 7: Parsing time for the sentences in Section 24 of less than 15 words of Viterbi parsing (none) (Left) and iterative parsing (iterative) (Right)

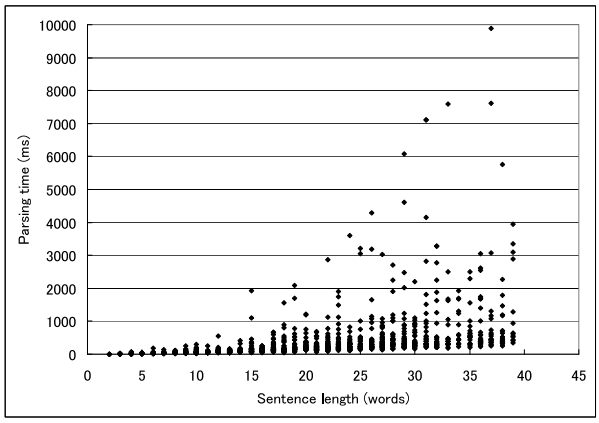

Figure 6: Parsing time versus sentence length for the sentences in Section 23 of less than 40 words

5 Evaluation

We evaluated the efficiency of the parsing techniques by using the HPSG for English developed by Miyao et al. (2005). The lexicon of the grammar was extracted from Sections 02-21 of the Penn Treebank (Marcus et al., 1994) (39,832 sentences). The grammar consisted of 2,284 lexical entry templates for 10,536 words[1]. The probabilistic disambiguation model of the grammar was trained using the same portion of the treebank (Miyao and Tsujii, 2005). The model included 529,856 features. The parameters for beam searching were determined manually by trial and error using Section 22; $\delta_0 = 12, \Delta\delta = 6, \delta_{\text{last}} = 30$, $\kappa_0 = 6.0, \Delta\kappa = 3.0, \kappa_{\text{last}} = 15.1$, $\theta_0 = 8.0, \Delta\theta = 4.0$, and $\theta_{\text{last}} = 20.1$. We used the chunk parser developed by Tsuruoka and Tsujii (2005a). Table 1 shows the abbreviations used in presenting the results.

We measured the accuracy of the predicate-argument relations output by the parser. A predicate-argument relation is defined as a tuple $\langle \sigma, w_h, a, w_a \rangle$, where σ is the predicate type (e.g., adjective, intransitive verb), w_h is the head word of the predicate, a is the argument label (MODARG, ARG1, ..., ARG4), and w_a is the head word of the argument. Precision/recall is the ratio of tuples correctly identified by the parser. This evaluation scheme was the same as used in previous evaluations of lexicalized grammars (Hockenmaier, 2003; Clark and Curran, 2004; Miyao and Tsujii, 2005). The experiments were conducted on an AMD Opteron server with a 2.4-GHz CPU. Section 22 of the Treebank was used as the development set, and performance was evaluated using sentences of less than 40 words in Section 23 (2,164 sentences, 20.3 words/sentence). The performance of each parsing technique was analyzed using the sentences in Section 24 of less than 15 words (305 sentences) and less than 40 words (1145 sentences).

Table 2 shows the parsing performance using all

[1]Lexical entry templates for POS are also developed. They are assigned to unknown words.

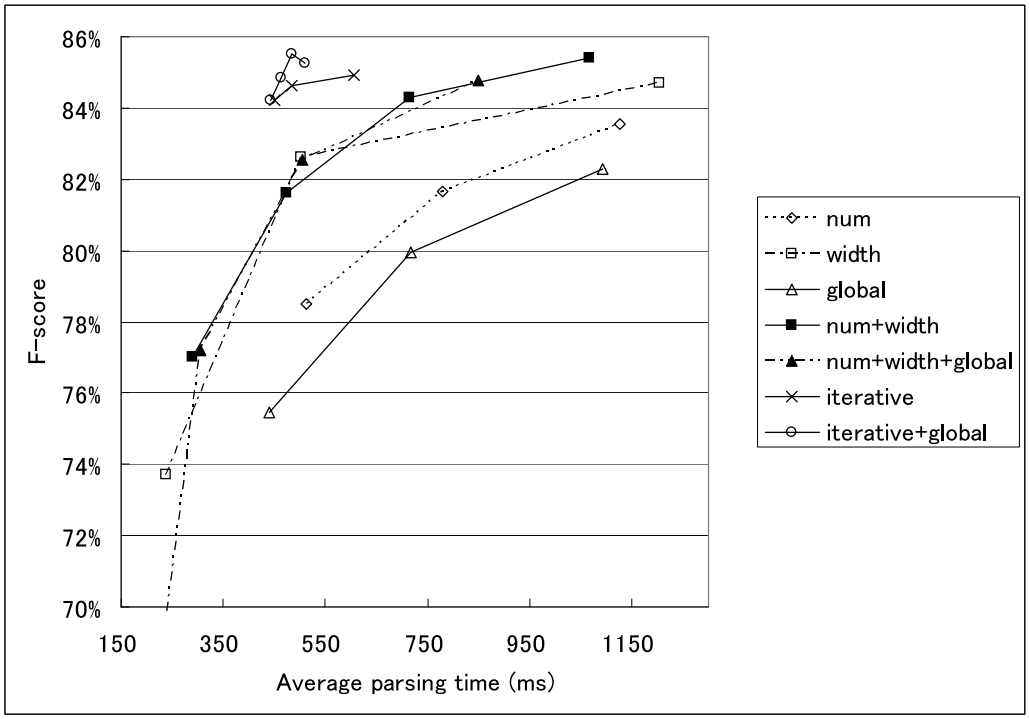

Figure 8: F-score versus average parsing time

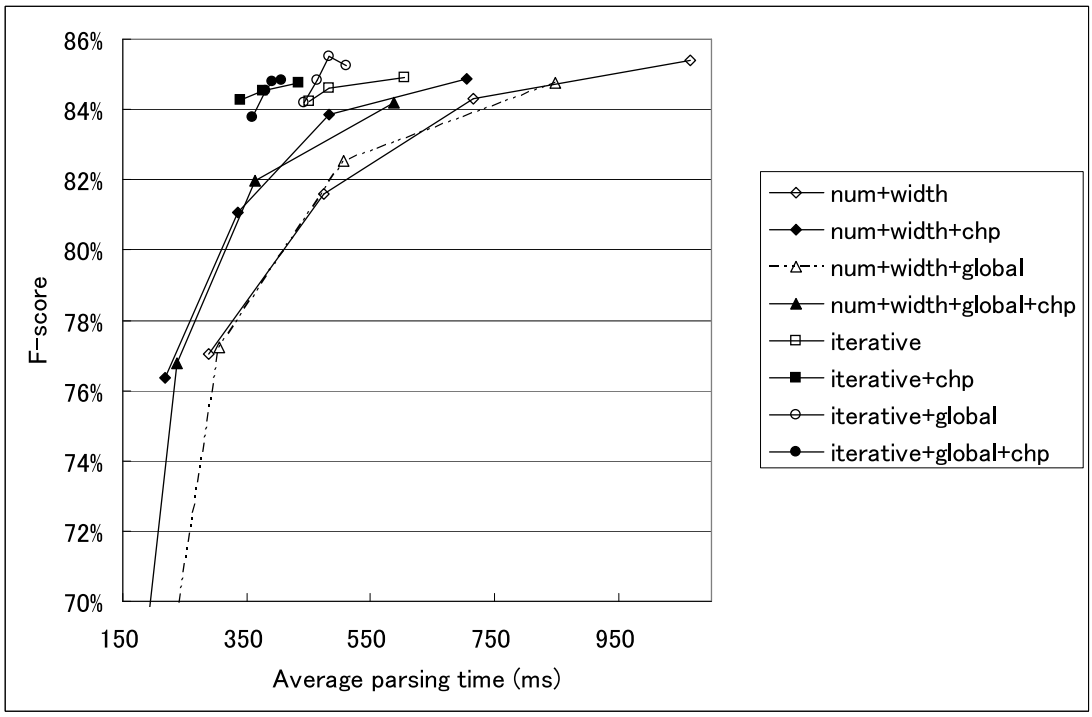

Figure 9: F-score versus average parsing time with/without chunk parser

Table 3: Viterbi parsing versus beam thresholding versus iterative parsing

	Precision	Recall	F-score	Avg. Time (ms)	No. of failed sentences
viterbi parsing (none)	88.22%	87.94%	88.08%	103923	2
beam search parsing (num+width)	88.96%	82.38%	85.54%	88	26
iterative parsing (iterative)	87.61%	87.24%	87.42%	99	2

Table 4: Contribution to performance of each implementation

	Precision	Recall	F-score	Avg. Time (ms)	diff(*)	No. of failed sentences
full	85.49%	84.21%	84.84%	407	0	13
full−piter	85.74%	84.70%	85.22%	631	224	10
full−qc	85.49%	84.21%	84.84%	562	155	13
full−chp	85.77%	84.76%	85.26%	510	103	10
full−global	85.23%	84.32%	84.78%	434	27	9
full−lci	85.68%	84.40%	85.03%	424	17	13
full−piter−qc−chp−global−lci	85.33%	84.71%	85.02%	1033	626	6

full ... iterative + global + chp
piter ... preserved iterative parsing
qc ... quick check
lci ... large constituent inhibition
diff(*) ... (Avg. Time of full) - (Avg. Time)

thresholding techniques and implementations described in Section 4 for the sentences in the development set (Section 22) and the test set (Section 23) of less than 40 words. In the table, precision, recall, average parsing time per sentence, and the number of sentences that the parser failed to parse are detailed. Figure 6 shows the distribution of parsing time for the sentence length.

Table 3 shows the performance of the Viterbi parsing, beam search parsing, and iterative parsing for the sentences in Section 24 of less than 15 words [2]. The parsing without beam searching took more than 1,000 times longer than with beam searching. However, the beam searching reduced the recall from 87.9% to 82.4%. The main reason for this reduction was parsing failure. That is, the parser could not output any results when the beam was too narrow instead of producing incorrect parse results. Although iterative parsing was originally developed for efficiency, the results revealed that it also increases the recall. This is because the parser continues trying until some results are output. Figure 7 shows the logarithmic graph of parsing time for the sentence length. The left side of the figure shows the parsing time of the Viterbi parsing and the right side shows the parsing time of the iterative parsing.

Figure 8 shows the performance of the parsing techniques for different parameters for the sentences in Section 24 of less than 40 words. The combinations of thresholding techniques achieved better results than the single techniques. Local thresholding using the width (width) performed better than that using the number (num). The combination of using width and number (num+width) performed better than single local and single global thresholding. The superiority of iterative parsing (iterative) was again demonstrated in this experiment. Although we did not observe significant improvement with global thresholding, the global plus iterative combination slightly improved performance.

Figure 9 shows the performance with and without the chunk parser. The lines with white symbols represent parsing without the chunk parser, and the lines with black symbols represent parsing with the chunk parser. The chunk parser improved the total parsing performance significantly. The improvements with global thresholding were less with the chunk parser.

Finally, Table 4 shows the contribution to performance of each implementation for the sentences in Section 24 of less than 40 words. The 'full' means the parser including all thresholding techniques and implementations described in Section 4. The 'full − x' means the full minus x. The preserved iterative parsing, the quick check, and the chunk parser greatly contributed to the final parsing speed, while the global thresholding and large constituent inhibition did not.

6 Conclusion

We have described the results of experiments with a number of existing techniques in head-driven phrase

[2]The sentence length was limited to 15 words because of inefficiency of Viterbi parsing

structure grammar (HPSG) parsing. Simple beam thresholding, similar to that for probabilistic CFG (PCFG) parsing, significantly increased the parsing speed over Viterbi algorithm, but reduced the recall because of parsing failure. Iterative parsing significantly increased the parsing speed without degrading precision or recall. We tested three techniques originally developed for deep parsing: quick check, large constituent inhibition, and HPSG parsing with a CFG chunk parser. The contributions of the large constituent inhibition and global thresholding were not significant, while the quick check and chunk parser greatly contributed to total parsing performance. The precision, recall and average parsing time for the Penn treebank (Section 23) were 87.85%, 86.85%, and 360 ms, respectively.

References

Steven P. Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618.

Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Joan Bresnan. 1982. *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

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

Sharon A. Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. of NAACL-2000*, pages 132–139.

Mahesh V. Chitrao and Ralph Grishman. 1990. Edge-based best-first chart parsing. In *Proc. of the DARPA Speech and Natural Language Workshop*, pages 263–266.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proc. of ACL'04*, pages 104–111.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, Univ. of Pennsylvania.

Michael Daum, Kilian A. Foth, and Wolfgang Menzel. 2003. Constraint based integration of deep and shallow parsing techniques. In *Proc. of EACL-2003*, pages 99–106.

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schaefer. 2003. Integrated shallow and deep parsing: TopP meets HPSG. In *Proc. of ACL'03*, pages 104–111.

Anette Frank. 2004. Constraint-based RMRS construction from shallow grammars. In *Proc. of COLING-2004*, pages 1269–1272.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proc. of ACL'02*, pages 279–286.

Joshua Goodman. 1997. Global thresholding and multiple pass parsing. In *Proc. of EMNLP-1997*, pages 11–25.

Julia Hockenmaier. 2003. Parsing with generative models of predicate-argument structure. In *Proc. of ACL'03*, pages 359–366.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proc. of ACL '99*, pages 535–541.

Dainiel Jurafsky and James H. Martin. 2000. *Speech and Language Processing*. Prentice Hall.

R. M. Kaplan, S. Riezler, T. H. King, J. T. Maxwell III, and A. Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc. of HLT/NAACL'04*.

Walter Kasper, Hans-Ulrich Krieger, Jörg Spilker, and Hans Weber. 1996. From word hypotheses to logical form: An efficient interleaved approach. In *Proceedings of Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, pages 77–88.

Martin Kay. 1989. Head driven parsing. In *Proc. of IWPT'89*, pages 52–62.

Bernd Kiefer, Hans-Ulrich Krieger, John Carroll, and Robert Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proc. of ACL'99*, pages 473–480, June.

Bernd Kiefer, Hans-Ulrich Krieger, and Detlef Prescher. 2002. A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. In *Proc. of COLING 2002*.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *Proc. of HLT-NAACL'03*.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proc. of IJCNLP-04 Workshop "Beyond Shallow Analyses"*.

Robert Malouf, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Journal of Natural Language Engineering*, 6(1):29–46.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proc. of CoNLL-2002*, pages 49–55.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Yuji Matsumoto, Hozumi Tanaka, Hideki Hirakawa, Hideo Miyoshi, and Hideki Yasukawa. 1983. BUP: A bottom up parser embedded in Prolog. *New Generation Computing*, 1(2):145–158.

John Maxwell and Ron Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proc. of HLT 2002*, pages 292–297.

Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proc. of ACL'05*, pages 83–90.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. 2003. Probabilistic modeling of argument structures including non-local dependencies. In *Proc. of RANLP '03*, pages 285–291.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii, 2005. *Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee and Oi Yee Kwong (Eds.), Natural Language Processing - IJCNLP 2004 LNAI 3248*, chapter Corpus-oriented Grammar Development for Acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank, pages 684–693. Springer-Verlag.

Mark-Jan Nederhof. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44.

H. Ney. 1991. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2):336–340.

Stephan Oepen, Dan Flickinger, Jun'ichi Tsujii, and Hans Uszkoreit, editors. 2002. *Collaborative Language Engineering: A Case Study in Efficient Grammar-Based Processing*. CSLI Publications.

Gerald Penn and Cosmin Munteanu. 2003. A tabulation-based parsing method that reduces copying. In *Proc. of ACL'03)*.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proc. of ACL'00*, pages 480–487.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Daniel J. Rosenkrantz and Philip M. Lewis II. 1970. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

Yves Shabes, Anne Abeillè, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proc. of COLING'88*, pages 578–583.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press.

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

Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. 2000. An HPSG parser with CFG filtering. *Journal of Natural Language Engineering*, 6(1):63–80.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005a. Chunk parsing revisited. In *Proc. of IWPT-2005*.

Yoshimasa Tsuruoka and Jun'ichi Tsujii, 2005b. *Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee and Oi Yee Kwong (Eds.), Natural Language Processing - IJCNLP 2004 LNAI 3248*, chapter Iterative CKY Parsing for Probabilistic Context-Free Grammars, pages 52–60. Springer-Verlag.

Gertjan van Noord. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3):425–456.