Better Binarization for the CKY Parsing

Xinying Song† * **Shilin Ding**§ * **Chin-Yew Lin**‡
†MOE-MS Key Laboratory of NLP and Speech, Harbin Institute of Technology, Harbin, China
§Department of Statistics, University of Wisconsin-Madison, Madison, USA
‡Microsoft Research Asia, Beijing, China
xysong@mtlab.hit.edu.cn dingsl@gmail.com cyl@microsoft.com

Abstract

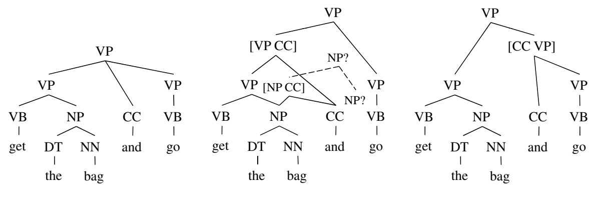
We present a study on how grammar binarization empirically affects the efficiency of the CKY parsing. We argue that binarizations affect parsing efficiency primarily by affecting the number of incomplete constituents generated, and the effectiveness of binarization also depends on the nature of the input. We propose a novel binarization method utilizing rich information learnt from training corpus. Experimental results not only show that different binarizations have great impacts on parsing efficiency, but also confirm that our learnt binarization outperforms other existing methods. Furthermore we show that it is feasible to combine existing parsing speed-up techniques with our binarization to achieve even better performance.

1 Introduction

Binarization, which transforms an n-ary grammar into an equivalent binary grammar, is essential for achieving an $O(n^3)$ time complexity in the context-free grammar parsing. $O(n^3)$ tabular parsing algorithms, such as the CKY algorithm (Kasami, 1965; Younger, 1967), the GHR parser (Graham et al., 1980), the Earley algorithm (Earley, 1970) and the chart parsing algorithm (Kay, 1980; Klein and Manning, 2001) all convert their grammars into binary branching forms, either explicitly or implicitly (Charniak et al., 1998).

In fact, the number of all possible binarizations of a production with $n + 1$ symbols on its right hand side is known to be the nth *Catalan Number* $C_n = \frac{1}{n+1}\binom{2n}{n}$. All binarizations lead to the same parsing accuracy, but maybe different parsing efficiency, i.e. parsing speed. We are interested in investigating whether and how binarizations will affect the efficiency of the CKY parsing.

Do different binarizations lead to different parsing efficiency? Figure 1 gives an example to help answer this question. Figure 1(a) illustrates the correct parse of the phrase "*get the bag and go*". We assume that $NP \rightarrow NP\ CC\ NP$ is in the original grammar. The symbols enclosed in square brackets in the figure are intermediate symbols.



(a) final parse (b) with left (c) with right

Figure 1: Parsing with left and right binarization.

If a left binarized grammar is used, see Figure 1(b), an extra constituent $[NP\,CC]$ spanning "*the bag and*" will be produced. Because rule $[NP\,CC] \rightarrow NP\,CC$ is in the left binarized grammar and there is an NP over "*the bag*" and a CC over the right adjacent "*and*". Having this constituent is unnecessary, because it lacks an NP to the right to complete the production. However, if a right binarization is used, as shown in Figure 1(c), such unnecessary constituent can be avoided.

One observation from this example is that different binarizations affect constituent generation, thus affect parsing efficiency. Another observation is that

*This work was done when Xinying Song and Shilin Ding were visiting students at Microsoft Research Asia.

for rules like $X \rightarrow Y\,C\,C\,Y$, it is more suitable to binarize them in a right branching way. This can be seen as a linguistic nature: for "*and*", usually the right neighbouring word can indicate the correct parse. A good binarization should reflect such liguistic nature.

In this paper, we aim to study the effect of binarization on the efficiency of the CKY parsing. To our knowledge, this is the first work on this problem.

We propose the problem to find the optimal binarization in terms of parsing efficiency (Section 3). We argue that binarizations affect parsing efficiency primarily by affecting the number of incomplete constituents generated, and the effectiveness of binarization also depends on the nature of the input (Section 4). Therefore we propose a novel binarization method utilizing rich information learnt from training corpus (Section 5). Experimental results show that our binarization outperforms other existing methods (Section 7.2).

Since binarization is usually a preprocessing step before parsing, we argue that better performance can be achieved by combining other parsing speed-up techniques with our binarization (Section 6). We conduct experiments to confirm this (Section 7.3).

2 Binarization

In this paper we assume that the *original* grammar, perhaps after preprocessing, contains no ϵ-productions or useless symbols. However, we allow the existence of unary productions, since we adopt an extended version of the CKY algorithm which can handle the unary productions. Moreover we do not distinguish nonterminals and terminals explicitly. We treat them as symbols. What we focus on is the procedure of binarization.

Definition 1. A *binarization* is a function π, mapping an n-ary grammar G to an equivalent binary grammar G'. We say that G' is a binarized grammar of G, denoted as $\pi(G)$.

Two grammars are *equivalent* if they define the same probability distribution over strings (Charniak et al., 1998).

We use the most widely used *left binarization* (Aho and Ullman, 1972) to show the procedure of binarization, as illustrated in Table 1, where p and q are the probabilities of the productions.

Original grammar	Left binarized grammar
$Y \rightarrow A\,B\,C : p$	$[A\,B] \rightarrow A\,B : 1.0$
$Z \rightarrow A\,B\,D : q$	$Y \rightarrow [A\,B]\,C : p$
	$Z \rightarrow [A\,B]\,D : q$

Table 1: Left binarization

In the binarized grammar, symbols of form $[A\,B]$ are *new* (also called *intermediate*) nonterminals. Left binarization always selects the left most pair of symbols and combines them to form an intermediate nonterminal. This procedure is repeated until all productions are binary.

In this paper, we assume that all binarizations follow the fashion above, except that the choice of pair of symbols for combination can be arbitrary. Next we show three other known binarizations.

Right binarization is almost the same with left binarization, except that it always selects the right most pair, instead of left, to combine.

Head binarization always binarizes from the head outward (Klein and Manning, 2003b). Please refer to Charniak et al. (2006) for more details.

Compact binarization (Schmid, 2004) tries to minimize the size of the binarized grammar. It leads to a *compact* grammar. We therefore call it compact binarization. It is done via a greedy approach: it always selects the pair that occurs most on the right hand sides of rules to combine.

3 The optimal binarization

The optimal binarization should help CKY parsing to achieve its best efficiency. We formalize the idea as follows:

Definition 2. The *optimal binarization* is π^*, for a given n-ary grammar G and a test corpus C:

$$\pi^* = \arg\min_{\pi} T(\pi(G), C) \qquad (1)$$

where $T(\pi(G), C)$ is the running time for CKY to parse corpus C, using the binarized grammar $\pi(G)$.

It is hard to find the optimal binarization directly from Definition 2. We next give an empirical analysis of the running time of the CKY algorithm and simplify the problem by introducing assumptions.

3.1 Analysis of CKY parsing efficiency

It is known that the complexity of the CKY algorithm is $O(n^3 L)$. The constant L depends on the binarized grammar in use. Therefore binarization will affect L. Our goal is to find a good binarization that makes parsing more efficient.

It is also known that in the inner most loop of CKY as shown in Algorithm 1, the for-statement in Line 1 can be implemented in several different methods. The choice will affect the efficiency of CKY. We present here four possible methods:

M1 Enumerate all rules $X \rightarrow Y\,Z$, and check if Y is in left span and Z in right span.

M2 For each Y in left span, enumerate all rules $X \rightarrow Y\,Z$, and check if Z is in right span.

M3 For each Z in right span, enumerate all rules $X \rightarrow Y\,Z$, and check if Y is in left span.

M4 Enumerate each Y in left span and Z in right span[1], check if there are any rules $X \rightarrow Y\,Z$.

Algorithm 1 The inner most loop of CKY

```
1: for $X \rightarrow YZ$, $Y$ in left span and $Z$ in right span
2:     Add $X$ to parent span
```

3.2 Model assumption

We have shown that both binarization and the for-statement implementation in the inner most loop of CKY will affect the parsing speed.

About the for-statement implementations, no previous study has addressed which one is superior. The actual choice may affect our study on binarization. If using M1, since it enumerates all rules in the grammar, the optimal binarization will be the one with minimal number of rules, i.e. minimal binarized grammar size. However, M1 is usually not preferred in practice (Goodman, 1997). For other methods, it is hard to tell which binarization is optimal theoretically. In this paper, for simplicity reasons we do not consider the effect of for-statement implementations on the optimal binarization.

On the other hand, it is well known that reducing the number of constituents produced in parsing can greatly improve CKY parsing efficiency. That is how most thresholding systems (Goodman, 1997; Tsuruoka and Tsujii, 2004; Charniak et al., 2006) speed up CKY parsing. Apparently, the number of constituents produced in parsing is not affected by for-statement implementations.

Therefore we assume that the running time of CKY is primarily determined by the number of constituents generated in parsing. We simplify the optimal binarization to be:

$$\pi^* \approx \arg\min_{\pi} E(\pi(G), C) \qquad (2)$$

where $E(\pi(G), C)$ is the number of constituents generated when CKY parsing C with $\pi(G)$.

We next discuss how binarizations affect the number of constituents generated in parsing, and present our algorithm for finding a good binarization.

4 How binarizations affect constituents

Throughout this section and the next, we will use an example to help illustrate the idea. The grammar is:

$$\begin{aligned} X &\rightarrow A\ B\ C\ D \\ Y &\rightarrow A\ B\ C \\ C &\rightarrow C\ D \\ Z &\rightarrow A\ B\ C\ E \\ W &\rightarrow F\ C\ D\ E \end{aligned}$$

The input sentence is ${}_0A_1B_2C_3D_4E_5$, where the subscripts are used to indicate the positions of spans. For example, $[1,3]$ stands for $B\,C$. The final parse[2] is shown in Figure 2. Symbols surrounded by dashed circles are fictitious, which do not actually exist in the parse.

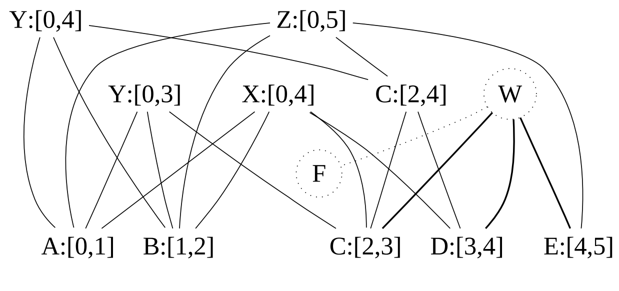


Figure 2: Parse of the sentence $A\,B\,C\,D\,E$

4.1 Complete and incomplete constituents

In the procedure of CKY parsing, there are two kinds of constituents generated: *complete* and *incomplete*.

Complete constituents (henceforth CCs) are those composed by the original grammar symbols and

[1]Note that we should skip Y (Z) if it never appears as the first (second) symbol on the right hand side of any rule.

[2]More precisely, it is more than a parse tree for it contains all symbols recognized in parsing.

spans. For example in Figure 2, $X\!:\![0,4]$, $Y\!:\![0,3]$ and Y:$[0,4]$ are all CCs.

Incomplete constituents (henceforth ICs) are those labeled by intermediate symbols. Figure 2 does not show them directly, but we can still read the possible ones. For example, if the binarized grammar in use contains an intermediate symbol $[A\,B\,C]$, then there will be two related ICs $[A\,B\,C]$:$[0,3]$ and $[A\,B\,C]$:$[0,4]$ (the latter is due to C:$[2,4]$) produced in parsing. ICs represent the intermediate steps to recognize and complete CCs.

4.2 Impact on complete constituents

Binarizations do not affect whether a CC will be produced. If there is a CC in the parse, whatever binarization we use, it will be produced. The difference merely lies on what intermediate ICs are used. Therefore given a grammar and an input sentence, no matter what binarization is used, the CKY parsing will generate the same set of CCs.

For example in Figure 2 there is a CC $X\!:\![0,4]$, which is associated with rule $X \rightarrow A\,B\,C\,D$. No matter what binarization we use, this CC will be recognized eventually. For example if using left binarization, we will get $[A\,B]\!:\![0,2]$, $[A\,B\,C]\!:\![0,3]$ and finally $X\!:\![0,4]$; if using right binarization, we will get $[C\,D]$:$[2,4]$, $[B\,C\,D]$:$[1,4]$ and again X:$[0,4]$.

4.3 Impact on incomplete constituents

Binarizations do affect the generation of ICs, because they generate different intermediate symbols. We discuss the impact on two aspects:

Shared IC. Some ICs can be used to generate multiple CCs in parsing. We call them *shared*. If a binarization can lead to more shared ICs, then overall there will be fewer ICs needed in parsing.

For example, in Figure 2, if we use left binarization, then $[A\,B]$:$[0,2]$ can be shared to generate both $X\!:\![0,4]$ and $Y\!:\![0,3]$, in which we can save one IC overall. However, if right binarization is used, there will be no common ICs to share in the generation steps of $X\!:\![0,4]$ and $Y\!:\![0,3]$, and overall there are one more IC generated.

Failed IC. For a CC, if it can be recognized eventually by applying an original rule of length k, whatever binarization to use, we will have to generate the same number of $k-2$ ICs before we can complete the CC. However, if the CC cannot be fully recognized but only partially recognized, then the number of ICs needed will be quite different.

For example, in Figure 2, the rule $W \rightarrow F\,C\,D\,E$ can be only partially recognized over $[2,5]$, so it cannot generate the corresponding CC. Right binarization needs two ICs ($[D\,E]\!:\![3,5]$ and $[C\,D\,E]\!:\![2,5]$) to find that the CC cannot be recognized, while left binarization needs none.

As mentioned earlier, ICs are auxiliary means to generate CCs. If an IC cannot help generate any CCs, it is totally useless and even harmful. We call such an IC *failed*, otherwise it is *successful*. Therefore, if a binarization can help generate fewer failed ICs then parsing would be more efficient.

4.4 Binarization and the nature of the input

Now we show that the impact of binarization also depends on the actual input. When the input changes, the impact may also change.

For example, in the previous example about the rule $W \rightarrow F\,C\,D\,E$ in Figure 2, we believe that left binarization is better based on the observation that there are more snippets of $[C\,D\,E]$ in the input which lack for F to the left. If there are more snippets of $[F\,C\,D]$ in the input lacking for E to the right, then right binarization would be better.

The discussion above confirms such a view: the effect of binarization depends on the nature of the input language, and a good binarization should reflect this nature. This accords with our intuition. So we use training corpus to learn a good binarization. And we verify the effectiveness of the learnt binarization using a test corpus with the same nature.

In summary, binarizations affect the efficiency of parsing primarily by affecting the number of ICs generated, where more shared and fewer failed ICs will help lead to higher efficiency. Meanwhile, the effectiveness of binarization also depends on the nature of its input language.

5 Towards a good binarization

Based on the analysis in the previous section, we employ a greedy approach to find a good binarization. We use training corpus to compute metrics for every possible intermediate symbol. We use this information to greedily select the best pair to combine.

5.1 Algorithm

Given the original grammar G and training corpus C, for every sentence in C, we firstly obtain the final parse (like Figure 2). For every possible intermediate symbol, i.e. every ngram of the original symbols, denoted by w, we compute the following two metrics:

1. How many ICs labeled by w can be generated in the final parse, denoted by $num(w)$ (number of related ICs).

2. How many CCs can be generated via ICs labeled by w, denoted by $ctr(w)$ (contribution of related ICs).

For example in Figure 2, for a possible intermediate symbol $[A\,B\,C]$, there are two related ICs ($[A\,B\,C] : [0, 3]$ and $[A\,B\,C] : [0, 4]$) in the parse, so we have $num([A\,B\,C]) = 2$. Meanwhile, four CCs ($Y:[0, 3]$, $X:[0, 4]$, $Y:[0, 4]$ and $Z:[0, 5]$) can be generated from the two related ICs. Therefore $ctr([A\,B\,C]) = 4$. We list the two metrics for every ngram in Figure 2 in Table 2. We will discuss how to compute these two metrics in Section 5.2.

w	num	ctr	w	num	ctr
$[A\,B]$	1	4	$[B\,C\,E]$	1	1
$[A\,B\,C]$	2	4	$[C\,D]$	1	2
$[A\,B\,C\,D]$	1	1	$[C\,D\,E]$	1	0
$[A\,B\,C\,E]$	1	1	$[C\,E]$	1	1
$[B\,C]$	2	4	$[D\,E]$	1	0
$[B\,C\,D]$	1	1			

Table 2: Metrics of every ngram

The two metrics indicate the goodness of a possible intermediate symbol w: $num(w)$ indicates how many ICs labeled by w are likely to be generated in parsing; while $ctr(w)$ represents how much w can *contribute* to the generation of CCs. If $ctr(w)$ is larger, the corresponding ICs are more likely to be shared. If ctr is zero, those ICs are surely failed. Therefore the smaller $num(w)$ is and the larger $ctr(w)$ is, the better w would be.

Combining num and ctr, we define a *utility* function for each ngram w in the original grammar:

$$utility(w) = f(num(w), ctr(w)) \quad (3)$$

where f is a ranking function, satisfying that $f(x, y)$ is larger when x is smaller and y is larger. We will discuss more details about it in Section 5.3.

Using *utility* as the ranking function, we sort all pairs of symbols and choose the best to combine. The formal algorithm is as follows:

S1 For every symbol pair of $\langle v_1, v_2\rangle$ (where v_1 and v_2 can be original symbols or intermediate symbols generated in previous rounds), let w_1 and w_2 be the ngrams of original symbols represented by v_1 and v_2, respectively. Let $w = w_1w_2$ be the ngram represented by the symbol pair. Compute $utility(w)$.

S2 Select the ngram w with the highest $utility(w)$, let it be w^* (in case of a tie, select the one with a smaller num). Let the corresponding symbol pair be $\langle v_1^*, v_2^*\rangle$.

S3 Add a new intermediate symbol v^*, and replace all the occurrences of $\langle v_1^*, v_2^*\rangle$ on the right hand sides of rules with v^*.

S4 Add a new rule $v^* \rightarrow v_1^*v_2^* : 1.0$.

S5 Repeat S1 $\sim$ S4, until there are no rules with more than two symbols on the right hand side.

5.2 Metrics computing

In this section, we discuss how to compute num and ctr in details.

Computing ctr is straightforward. First we get final parses like in Figure 2 for training sentences. From a final parse, we traverse along every parent node and enumerate every subsequence of its child nodes. For example in Figure 2, from the parent node of $X : [0, 4]$, we can enumerate the following: $[A\,B]:[0, 2]$, $[A\,B\,C]:[0, 3]$, $[A\,B\,C\,D]:[0, 4]$, $[B\,C]$:$[1, 3]$, $[B\,C\,D]$:$[1, 4]$, $[C\,D]$:$[2, 4]$. We add 1 to all the ctr of these ngrams, respectively.

To compute num, we resort to the same idea of dynamic programming as in CKY. We perform a normal left binarization except that we add all ngrams in the original grammar G as intermediate symbols into the binarized grammar G'. For example, for the rule of $S \rightarrow A\,B\,C : p$, the constructed grammar is as follows:

$$\begin{aligned} [A\,B] &\rightarrow A\,B : 1.0 \\ S &\rightarrow [A\,B]\,C : p \\ [B\,C] &\rightarrow B\,C : 1.0 \end{aligned}$$

Using the constructed G', we employ a normal CKY parsing on the training corpus and compute how many constituents are produced for each ngram. The result is num. Suppose the length of the training sentence is n, the original grammar G has N symbols, and the maximum length of rules is k, then the complexity of this method can be written as $O(N^k n^3)$.

5.3 Ranking function

We discuss the details of the ranking function f used to compute the $utility$ of each ngram w. We come up with two forms for f: linear and log-linear

1. linear: $f(x, y) = -\lambda_1 x + \lambda_2 y$
2. log-linear[3]: $f(x, y) = -\lambda_1 \log(x) + \lambda_2 \log(y)$

where λ_1 and λ_2 are non-negative weights subject to $\lambda_1 + \lambda_2 = 1$[4].

We will use development set to determine which form is better and to learn the best weight settings.

6 Combination with other techniques

Binarization usually plays a role of preprocessing in the procedure of parsing. Grammars are binarized before they are fed into the stage of parsing. There are many known works on speeding up the CKY parsing. So we can expect that if we replace the part of binarization by a better one while keeping the subsequent parsing unchanged, the parsing will be more efficient. We will conduct experiment to confirm this idea in the next section.

We would like to make more discussions before we advance to the experiments. The first is about parsing accuracy in combining binarization with other parsing speed-up techniques. Binarization itself does not affect parsing accuracy. When combined with exact inference algorithms, like the iterative CKY (Tsuruoka and Tsujii, 2004), the accuracy will be the same. However, if combined with other inexact pruning techniques like beam-pruning (Goodman, 1997) or coarse-to-fine parsing (Charniak et al., 2006), binarization may interact with those pruning methods in a complicated way to affect parsing accuracy. This is due to different binarizations generate different sets of intermediate symbols. With the same complete constituents, one binarization might derive incomplete constitutes that could be pruned while another binarization may not. This would affect the accuracy. We do not address this interaction on in this paper, but leave it to the future work. In Section 7.3 we will use the iterative CKY for testing.

In addition, we believe there exist some speed-up techniques which are incompatible with our binarization. One such example may be the top-down left-corner filtering (Graham et al., 1980; Moore, 2000), which seems to be only applicable to the process of left binarization. A detailed investigation on this problem will be left to the future work.

The last issue is how our binarization performs on a lexicalized parser, like Collins (1997). Our intuition is that we cannot apply our binarization to Collins (1997). The key fact in lexicalized parsers is that we cannot explicitly write down all rules and compute their probabilities precisely, due to the great number of rules and the severe data sparsity problem. Therefore in Collins (1997) grammar rules are already factorized into a set of probabilities. In order to capture the dependency relationship between lexcial heads Collins (1997) breaks down the rules from head outwards, which prevents us from factorizing them in other ways. Therefore our binarization cannot apply to the lexicalized parser. However, there are state-of-the-art unlexicalized parsers (Klein and Manning, 2003b; Petrov et al., 2006), to which we believe our binarization can be applied.

7 Experiments

We conducted two experiments on Penn Treebank II corpus (Marcus et al., 1994). The first is to compare the effects of different binarizations on parsing and the second is to test the feasibility to combine our work with iterative CKY parsing (Tsuruoka and Tsujii, 2004) to achieve even better efficiency.

7.1 Experimental setup

Following conventions, we learnt the grammar from Wall Street Journal (WSJ) section 2 to 21 and modified it by discarding all functional tags and empty nodes. The parser obtained this way is a pure unlexicalized context-free parser with the raw treebank grammar. Its accuracy turns out to be 72.46% in

[3]For log-linear form, if $num(w) = 0$ (and consequently $ctr(w) = 0$), we set $f(num(w), ctr(w)) = 0$; if $num(w) > 0$ but $ctr(w) = 0$, we set $f(num(w), ctr(w)) = -\infty$.

[4]Since f is used for ranking, the magnitude is not important.

terms of F1 measure, quite the same as 72.62% as stated in Klein and Manning (2003b). We adopt this parser in our experiment not only because of simplicity but also because we focus on parsing efficiency.

For all sentences with no more than 40 words in section 22, we use the first 10% as the development set, and the last 90% as the test set. There are 158 and 1,420 sentences in development set and test set, respectively. We use the whole 2,416 sentences in section 23 as the training set.

We use the development set to determine the better form of the ranking function f as well as to tune its weights. Both metrics of num and ctr are normalized before use. Since there is only one free variable in λ_1 and λ_2, we can just enumerate $0 \leq \lambda_1 \leq 1$, and set $\lambda_2 = 1 - \lambda_1$. The increasing step is firstly set to 0.05 for the approximate location of the optimal weight, then set to 0.001 to learn more precisely around the optimal.

We find that the optimal is 5,773,088 (constituents produced in parsing development set) with $\lambda_1 = 0.014$ for linear form, while for log-linear form the optimal is 5,905,292 with $\lambda_1 = 0.691$. Therefore we determine that the better form for the ranking function is linear with $\lambda_1 = 0.014$ and $\lambda_2 = 0.986$.

The size of each binarized grammar used in the experiment is shown in Table 3. "Original" refers to the raw treebank grammar. "Ours" refers to the learnt binarized grammar by our approach. For the rest please refer to Section 2.

	# of Symbols	# of Rules
Original	72	14,971
Right	10,654	25,553
Left	12,944	27,843
Head	11,798	26,697
Compact	3,644	18,543
Ours	8,407	23,306

Table 3: Grammar size of different binarizations

We also tested whether the size of the training set would have significant effect. We use the first 10%, 20%, $\cdots$, up to 100% of section 23 as the training set, respectively, and parse the development set. We find that all sizes examined have a similar impact, since the numbers of constituents produced are all around 5,780,000. It means the training corpus does not have to be very large.

The entire experiments are conducted on a server with an Intel Xeon 2.33 GHz processor and 8 GB memory.

7.2 Experiment 1: compare among binarizations

In this part, we use CKY to parse the entire test set and evaluate the efficiency of different binarizations.

The for-statement implementation of the inner most loop of CKY will affect the parsing time though it won't affect the number of constituents produced as discussed in Section 3.2. The best implementations may be different for different binarized grammars. We examine M1$\sim$M4, testing their parsing time on the development set. Results show that for right binarization the best method is M3, while for the rest the best is M2. We use the best method for each binarized grammar when comparing the parsing time in Experiment 1.

Table 4 reports the total number of constituents and total time required for parsing the entire test set. It shows that different binarizations have great impacts on the efficiency of CKY. With our binarization, the number of constituents produced is nearly 20% of that required by right binarization and nearly 25% of that by the widely-used left binarization. As for the parsing time, CKY with our binarization is about 2.5 times as fast as with right binarization and about 1.75 times as fast as with left binarization. This illustrates that our binarization can significantly improve the efficiency of the CKY parsing.

Binarization	Constituents	Time (s)
Right	241,924,229	5,747
Left	193,238,759	3,474
Head	166,425,179	3,837
Compact	94,257,478	2,302
Ours	**52,206,466**	**2,182**

Table 4: Performance on test set

Figure 3 reports the detailed number of complete constituents, successful incomplete constituents and failed incomplete constituents produced in parsing. The result proves that our binarization can significantly reduce the number of failed incomplete constituents, by a factor of 10 in contrast with left binarization. Meanwhile, the number of successful incomplete constituents is also reduced by a factor of 2 compared to left binarization.

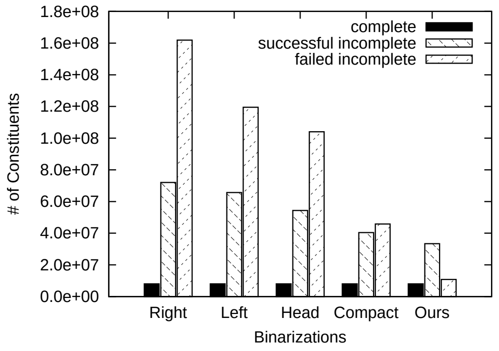


Figure 3: Comparison on various constituents

Another interesting observation is that parsing with a smaller grammar does not always yield a higher efficiency. Our binarized grammar is more than twice the size of compact binarization, but ours is more efficient. It proves that parsing efficiency is related to both the size of grammar in use as well as the number of constituents produced.

In Section 1, we used an example of "*get the bag and go*" to illustrate that for rules like $X \rightarrow Y\,CC\,Y$, right binarization is more suitable. We also investigated the corresponding linguistic nature that the word to the right of "*and*" is more likely to indicate the true relationship represented by "*and*". We argued that a better binarization can reflect such linguistic nature of the input language. To our surprise, our learnt binarization indeed captures this linguistic insight, by binarizing $NP \rightarrow NP\,CC\,NP$ from right to left.

Finally, we would like to acknowledge the limitation of our assumption made in Section 3.2. Table 4 shows that the parsing time of CKY is not always monotonic increasing with the number of constituents produced. Head binarization produces fewer constituents than left binarization but consumes more parsing time.

7.3 Experiment 2: combine with iterative CKY

In this part, we test the performance of combining our binarization with the iterative CKY (Tsuruoka and Tsujii, 2004) (henceforth T&T) algorithm.

Iterative CKY is a procedure of multiple passes of normal CKY: in each pass, it uses a threshold to prune bad constituents; if it cannot find a successful parse in one pass, it will relax the threshold and start another; this procedure is repeated until a successful parse is returned. T&T used left binarization. We re-implement their experiments and combine iterative CKY with our binarization. Note that iterative CKY is an exact inference algorithm that guarantees to return the optimal parse. As discussed in Section 6, the parsing accuracy is not changed in this experiment.

T&T used a held-out set to learn the best step of threshold decrease. They reported that the best step was 11 (in log-probability). We found that the best step was indeed 11 for left binarization; for our binarizaiton, the best step was 17. T&T used M4 as the for-statement implementation of CKY. In this part, we follow the same method.

The result is shown in Table 5. We can see that iterative CKY can achieve better performance by using a better binarization. We also see that the reduction by binarization with pruning is less significant than without pruning. It seems that the pruning itself in iterative CKY can counteract the reduction effect of binarization to some extent. Still the best performance is archieved by combining iterative CKY with a better binarization.

CKY + Binarization	Constituents	Time (s)
Tsuruoka and Tsujii (2004)		
CKY + Left	45,406,084	1,164
Iterative CKY + Left	17,520,427	613
Reimplement		
CKY + Left	52,128,941	932
CKY + Ours	14,892,203	571
Iterative CKY + Left	23,267,594	377
Iterative CKY + Ours	**10,966,272**	**314**

Table 5: Combining with iterative CKY parsing

8 Related work

Almost all work on parsing starts from a binarized grammar. Usually binarization plays a role of preprocessing. Left binarization is widely used (Aho and Ullman, 1972; Charniak et al., 1998; Tsuruoka and Tsujii, 2004) while right binarization is rarely used in the literature. Compact binarization was introduced in Schmid (2004), based on the intuition that a more compact grammar will help acheive a highly efficient CKY parser, though from our experiment it is not always true.

We define the fashion of binarizations in Section 2, where we encode an intermediate symbol using the ngrams of original symbols (content) it derives. This encoding is known as the Inside-Trie (I-Trie) in Klein and Manning (2003a), in which they also mentioned another encoding called Outside-Trie (O-Trie). O-Trie encodes an intermediate symbol using the its parent and the symbols surrounding it in the original rule (context). Klein and Manning (2003a) claimed that O-Trie is superior for calculating estimates for A* parsing. We plan to investigate binarization defined by O-Trie in the future.

Both I-Trie and O-Trie are equivalent encodings, resulting in equivalent grammars, because they both encode using the complete content or context information of an intermediate symbol. If we use part of the information to encode, for example just parent in O-Trie case, the encoding will be non-equivalent.

Proper non-equivalent encodings are used to generalize the grammar and prevent the binarized grammar becoming too specific (Charniak et al., 2006). It is equipped with head binarization to help improve parsing accuracy, following the traditional linguistic insight that phrases are organized around the head (Collins, 1997; Klein and Manning, 2003b). In contrast, we focus our attention on parsing efficiency not accuracy in this paper.

Binarization also attracts attention in the syntax-based models for machine translation, where translation can be modeled as a parsing problem and binarization is essential for efficient parsing (Zhang et al., 2006; Huang, 2007).

Wang et al. (2007) employs binarization to decompose syntax trees to acquire more re-usable translation rules in order to improve translation accuracy. Their binarization is restricted to be a mixture of left and right binarization. This constraint may decrease the power of binarization when applied to speeding up parsing in our problem.

9 Conclusions and future work

We have studied the impact of grammar binarization on parsing efficiency and presented a novel binarization which utilizes rich information learnt from training corpus. Experiments not only showed that our learnt binarization outperforms other existing ones in terms of parsing efficiency, but also demonstrated the feasibility to combine our binarization with known parsing speed-up techniques to achieve even better performance.

An advantage of our approach to finding a good binarization would be that the training corpus does not need to be parsed sentences. Only POS tagged sentences will suffice for training. This will save the effort to adapt the model to a new domain.

Our approach is based on the assumption that the efficiency of CKY parsing is primarily determined by the number of constituents produced. This is a fairly sound one, but not always true, as shown in Section 7.2. One future work will be relaxing the assumption and finding a better appraoch.

Another future work will be to apply our work to chart parsing. It is known that binarization is also essential for an $O(n^3)$ complexity of chart parsing, where *dotted rules* are used to binarize the grammar implicitly from left. As shown in Charniak et al. (1998), we can binarize explicitly and use intermediate symbols to replace dotted rules in chart parsing. Therefore chart parsing can use multiple binarizations. We expect that a better binarization will also help improve the efficiency of chart parsing.

Acknowledgements

We thank the anonymous reviwers for their pertinent comments, Yoshimasa Tsuruoka for the detailed explanations on his referred paper, Yunbo Cao, Shujian Huang, Zhenxing Wang , John Blitzer and Liang Huang for their valuable suggestions in preparing the paper.

References

Aho, A. V. and Ullman, J. D. (1972). *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Charniak, E., Goldwater, S., and Johnson, M. (1998). Edge-based best-first chart parsing. In *Proceedings of the Six Workshop on Very Large Corpora*, pages 127–133.

Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., and Vu, T. (2006). Multi-level coarse-to-fine pcfg parsing. In *HLT-NAACL*.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *ACL*.

Earley, J. (1970). An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.

Goodman, J. (1997). Global thresholding and multiple-pass parsing. In *EMNLP*.

Graham, S. L., Harrison, M. A., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462.

Huang, L. (2007). Binarization, synchronous binarization, and target-side binarization. In *Proceedings of SSST, NAACL-HLT 2007 / AMTA Workshop on Syntax and Structure in Statistical Translation*, pages 33–40, Rochester, New York. Association for Computational Linguistics.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.

Kay, M. (1980). Algorithm schemata and data structures in syntactic processing. Technical Report CSL80-12, Xerox PARC, Palo Alto, CA.

Klein, D. and Manning, C. D. (2001). Parsing and hypergraphs. In *IWPT*.

Klein, D. and Manning, C. D. (2003a). A* parsing: Fast exact viterbi parse selection. In *HLT-NAACL*.

Klein, D. and Manning, C. D. (2003b). Accurate unlexicalized parsing. In *ACL*.

Marcus, M. P., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The penn treebank: Annotating predicate argument structure. In *HLT-NAACL*.

Moore, R. C. (2000). Improved left-corner chart parsing for large context-free grammars. In *IWPT*.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *ACL*.

Schmid, H. (2004). Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *COLING*.

Tsuruoka, Y. and Tsujii, J. (2004). Iterative cky parsing for probabilistic context-free grammars. In *IJCNLP*.

Wang, W., Knight, K., and Marcu, D. (2007). Binarizing syntax trees to improve syntax-based machine translation accuracy. In *EMNLP-CoNLL*.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Zhang, H., Huang, L., Gildea, D., and Knight, K. (2006). Synchronous binarization for machine translation. In *HLT-NAACL*.