A Comparison of Pruning Methods for CYK-based Decoding in Machine Translation

YuZe Gao
Natural Language Processing Lab
Northeastern University
yuze.gao@outlook.com

Tong Xiao
Natural Language Processing Lab
Northeastern University
xiaotong@mail.neu.edu.cn

Abstract

We present some popular pruning methods for CYK-based decoding in machine translation, and describe the implementation of them. Then, we provide the experimental results of these methods and the comparison of these results. In addition, we analyze each method in terms of decoding speed and translation accuracy, based on which some possible optimizations for each method are given. Lastly, we propose some novel pruning methods for CYK-based decoding.

1 Introduction

In recent years, statistical machine translation (SMT) has been extensively investigated, showing state-of-the-art performance in many translation tasks. In current SMT paradigm, a core step is to search for the "best" target string for the given source string, namely decoding. Several methods are available to implement SMT decoders. For instance, we can incrementally add target words in a left-to-right fashion [Ortiz, 2003; Yang, 2010], or build translation hypotheses in a bottom-up fashion [Young, 1996]. One popular method is CYK-based decoding that originates from monolingual parsing [Cheppalier, 1998].

In CYK decoders, a partial hypothesis can be produced by the use of hypotheses generated on smaller segments/spans (Fig. 1). The algorithm starts with the smallest spans, and proceeds once it generates all possible hypotheses for a span. The final translation can be accessed when we finish the computation on the entire span. The brilliance of CYK-based decoding comes from its simplicity and from the natural manner in which one can build derivations using linguistically-motivated grammars or formally syntactic rules. Therefore, it is widely used in hierarchical machine translation (MT) systems [Vilar, 2012; V. F. López, 2010; Chiang David, 2007].

One bottleneck of SMT decoding is its speed. In CYK-based decoding, there are two factors that can slow down the system.

1) ***Large Search Space***. Given a source string, the number of possible translations is huge. Even for a word-based model, decoding is a NP-complete problem [Knight, 1999]. The situation is worse for modern hierarchical MT models because more ambiguities are introduced by the underlying derivations of rules.

2) ***Cubic Time Complexity of CYK***. The time complexity of CYK algorithm is $O(n^3)$, i.e., the decoding time is cubic to the length of the input sentence. Decoding long sentences is a big problem.

Obviously, pruning is of great importance to the speed-up of CYK decoders. The simplest of these is beam pruning, which keeps the most promising candidates in a certain distance from the top-1 candidate, and discards the rest [Koehn, 2004; Robert C 2007]. The decoder can run faster using cube pruning/growing, which is also popular in MT systems [Gesmundo et al, 2010]. Cube pruning is particularly powerful if one can organize the decoding problem into a search problem in hypergraphs.

In this paper, we empirically compare three pruning methods for CYK-based decoding that try to address or relieve its cubic time complexity (Fig. 1). In particular, we divide the CYK algorithm into two parts – a dual loop on spans ($O(n^2)$) and a loop on segmentation that chops a given span ($O(n)$). We use the parser tree and punctuation information to prune unlikely spans (the two outermost loops), and use phrase

Proceedings of the 11th China Workshop on Machine Translation, pages 65 – 73, Sept. 23-25, 2015 HeFei, AnHui, China

boundary tags to prune unlikely segmentations (the innermost loop).

We implement these methods using the NiuTrans toolkit, and compare the experimental results of them with that of the baseline, which applied beam pruning and cube pruning. We found that the pruning quality is in proportion to the retentivity and osculation of the expression meaning. For example, the punctuation inserted by people is not only good for pruning, but also not harmful to the BLEU (translation quality index) due to its retentivity of expression intention and its meaning entirety. We also found that the pruning strategies we obtained from our analysis are advantageous for decoding speed-up but have some impair on the BLUE. Furthermore, because parser trees are not the perfect decoding paths that the machine translation systems can recognize, they may skip some key paths with potential global optimal hypotheses. In addition, the methods that used information we got from word alignment showed favorable maintenance on the BLUE.

Lastly, we propose some novel ideas that can bring further decoding speed-up and BLUE improvements.

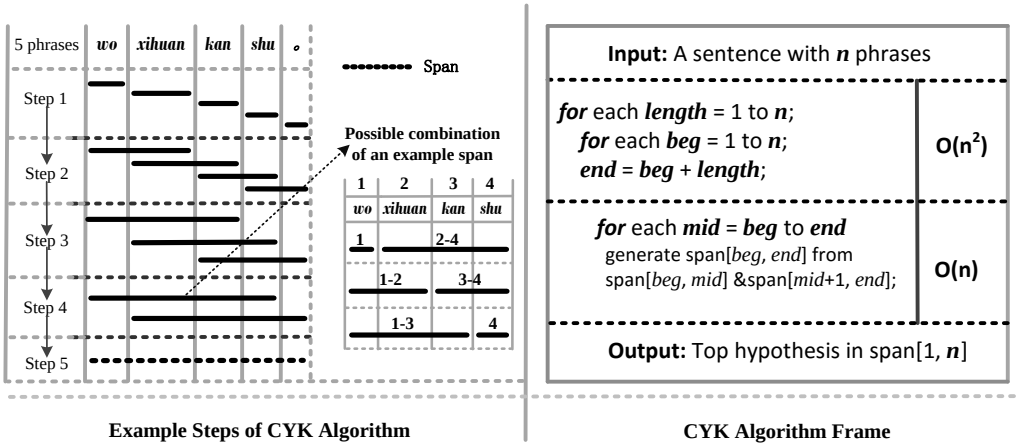

Fig. 1. Example Steps and Frame of CYK Algorithm

2 Baseline System

We used the open source toolkit NiuTrans (Xiao et al., 2012) in this work. In particular, we chose NiuTrans.Phrase, which supports Bracket Transduction Grammars (BTGs) and resorts to CYK decoding, and conducted experiments with various pruning methods on Chinese-English translation. By default, beam pruning (beam width = 30) and cube pruning are used. We used the GIZA++ toolkit to get symmetric word alignments from bilingual corpus and obtained the final alignment using the "final-diag-grow-and" heuristics. All translation phrases are extracted from bitext in a standard manner [Koehn et al., 2003]. Nine features are employed for decoding, including bidirectional translation probabilities, bidirectional lexical weights, an n-gram language model, a word bonus, a phrase bound, and a maximum entropy-based reordering model. All feature weights are tuned on a development set via minimum error rate training [Och.F.J, 2003].

Our bilingual corpus consists of 462300 sentences (LDC2003E14, LDC2005T10, LDC2003E07, LDC2005T06, LDC2005E83, LDC2006E26). The 5-gram language model was trained using the Xinhua portion of the English Gigaword corpus and the target-side of the bilingual data. We used the NIST MT04 evaluation set as the Dev set and MT05/06 as the Test set. BLEU values and decoding speed of development and test sets are shown in Fig. 2.

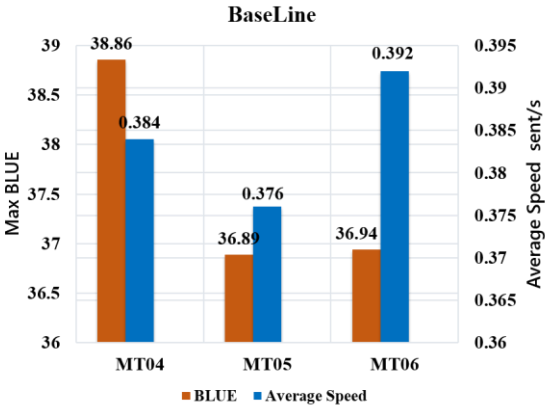

Fig. 2. Experimental Result of Baseline System

3 Parser Tree-based Pruning

Linguistically, a sentence can be decomposed into phrasal segments (several words) in certain organized form. A popular way is to represent a sentence with a syntactic tree or a parser tree. In such a way, one can capture the underlying structure of the sentence and the recursive manner in which grammar rules form this structure. See Fig. 3 for an example parser tree. Despite good liguistical explanations, the decoding of most MT systems does not require the guidance of parser trees. It is natural to impose restrictions on CKY decoding so that decoding is only performed on spans that are consistent with tree constituents.

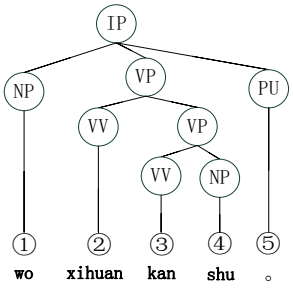

Fig. 3. Example Parser Tree

3.1 Implementation

In this experiment, we applied the parser tree in the loop for locating the *beg* and *end*. First, we define a constraint as the set of possible spans that refers to the *beg* and *end* of a span in the parser tree. We regard spans that are in accordance with the parser tree are qualified for our protection. We obtain the constraint information from the parser tree and label each phrase with a number tag. In addition, when generating the parser tree, we store the leftmost and rightmost children tags in their attributive father nodes (except the leaf nodes that store the information of phrases). Plus, given a tree node with ***m*** children nodes (excluding leaf nodes), we can have ***n*** combination of its children nodes to ensure the complete combinations of the span.

$$n = C_m^2$$

To better understand the span definition, we explain it using an example, which represents a sentence with 12 phrases (Fig. 4). The span node indicates the property of the phrase span (e.g., VP means verb phrases). For example, the span node VP is a sequence from phrase 4 to phrase 8 and it is a VP phrase span. The phrase node indicates the number tag of the phrase. When generating the parser tree, take the span node VP for example, we store the numbers 4 and 8 in the node VP, indicating its leftmost and rightmost phrases. Furthermore, we permit the combination of two child nodes (except the leaf node). All the children nodes of a node can be combined together, the result of which is a span with the leftmost and rightmost phrase tags; this ensures that the whole sentence can be eventually produced. For example, possible combinations of VP's children nodes are: DT and CP, DT and NN, and CP and NN, returning tag 4 and tag 7, tag 4 and tag 8, and tag 5 and tag 8 respectively.

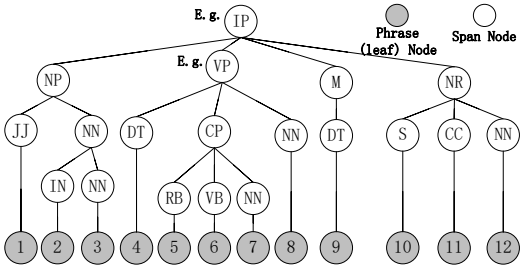

Fig. 4. Constraints in a Parser Tree

The set of span information is a representation of the decoding paths of the sentence. We store all the permitted spans in the information set. Because we apply the information set as a hard constraint in the decoding, we consider and generate the hypotheses of spans in the set.

3.2 Experimental Results

In this experiment, we did not set limitations on the pruning lengths, that is, we permit the pruning on all spans regardless of their lengths. The experimental result is shown in Fig. 5.

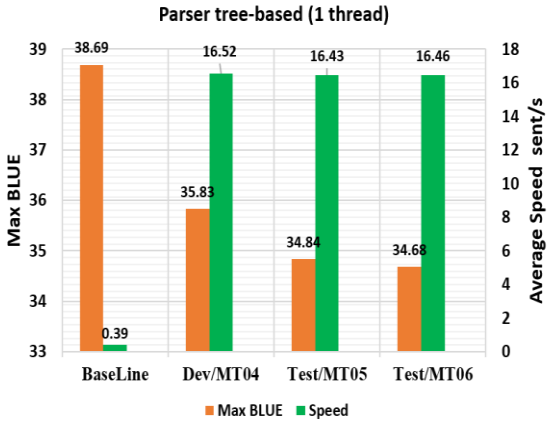

Fig. 5. Experimental Result of Parser Tree-based Pruning

It can be seen from Fig.5 that parser tree-based pruning makes significant improvement on the decoding speed (nearly 40 times faster). However, the translation quality suffers a great loss, with a decrease of 3 point in the BLUE.

3.3 Discussion and Improvement

In the implementation of parser tree-based pruning, only the spans that are in accordance with the parser tree information are allowed to generate translation hypotheses. Though this mechanism can speed up the decoding process significantly, it definitely prunes the most sentence spans, including the useful ones that can improve the translation quality. Furthermore, imprecise parser tree may impair the translation quality, such as neglecting useful spans, or combining the phrases into the whole sentence with a wrong structure.

Improper pruning is likely to eliminate potential high-quality translation hypotheses. However, we can make improvements through the following methods.

1) ***Weaken the Pruning Degree***

We can define a required condition for the pruning, i.e., we do pruning only when the span satisfies the required condition. If we know certain structures in the sentence cannot be translated well, but they bear a well analyzed parser result, we can use the parser information to direct this kind of span decoding. For example, we can just do pruning on phrase spans with property VP or NP. Another way is adding a parameter to control the threshold of pruning lengths. In addition, we can also increase the possibility of high-quality spans by permitting the combinations of children nodes of different span nodes in the parser tree.

2) ***Enlarge the Coverage Degree of Parser Tree Information***

We can increase the number of possible decoding paths for a given span in parser tree-based pruning. Because one sentence can have different parser trees under different grammar analysis (rightmost reduction analysis, leftmost reduction analysis, etc.). We can employ information of these parser trees to make a union parser tree information to gain more reasonable hypotheses paths in the span, lightening the side effects from imprecise parser tree, which may increase the BULE value.

4 Punctuation-based Pruning

In general, punctuations are used to separate a sentence into several meaningful segments. For instance, commas are widely used in starting or ending a clause. So most of the current languages have applied the punctuation to make efficient and clear expression. In MT, we can make use of this “natural” segmentation to prune hypotheses that cross punctuations over two or more individual segments.

4.1 Implementation

In machine translation, what we expect is that the translation result is loyal to the actual meaning of source language. So, in punctuation-based pruning, our goal is to skip the useless spans that will not form a complete meaning. For an n-phrase sentence with only one punctuation at point k ($1 \le k \le n$) and no other punctuations besides beginning and end points of the sentence, we can say there are $\{(k\text{-}1)*(n\text{-}k)\text{-}2\}$ bad spans in the sentence because they cross the punctuation. Actually, we will skip these spans because they will not present us a complete meaning of parts of the sentence. Since we use the punctuation to check the span integrity on partial meaning, the punctuation constraints can be used in the second stage when choosing spans.

When implementing punctuation-based pruning, we use the following restriction principles. As for spans with no punctuation in them, we will generate all hypotheses for them. This pruning method we defined just works on the spans with punctuations. In other words, we just consider spans that have punctuations in them and judge them according to the following criteria.

1) If span [i-1, i+1] has punctuations in it, we consider this span to be valuable and will generate all possible combinations.

2) If span [i, i+1] or span [j-1, j] is the beginning or the end of the decoding sentence, we treat this case like case 1).

3) If neither span [i, i+1] nor span [j-1, j] has a punctuation, or is the beginning or end of the decoding sentence, we will skip these spans.

To understand this method, we will illustrate using an example (Fig. 6). The example sentence has 13 phrases, and it is used to demonstrate the constraints on the second stage of CYK. Span [5, 10] is not qualified because it crosses a punctuation at point 7, and the same is true for span [1, 10]. Though span [3, 10] also crosses a punctuation, it is qualified because it is useful to form the bigger span [3, 12], span [3, 13], etc., which have relative complete meanings. Though both the beginning and end points of span [0, 10] are not punctuations, the beginning point is the beginning of the whole sentence, so it also deserves our protection in the constraints.

4.2 Experimental Results

In this experiment, we took into consideration the points before and after the spans. Further-

more, we impose restrictions on the left of spans (the span that between the point/phrase immediately before the span and the *beginning* of the span) and right of spans (the span that between the point/phrase after the span and the *end* of the span) separately. The experimental result is shown in Fig. 7.

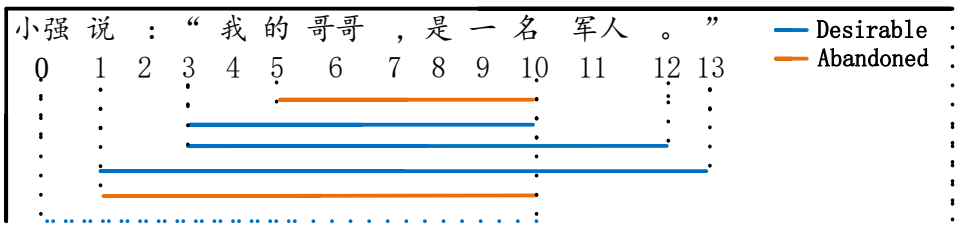

Fig. 6. Example Sentence with Punctuations

The baseline system applied cube-pruning and beam search. “Left” experiment means the left restriction we defined is qualified, and “Right” experiment means the right restriction is qualified. “Left && Right” means both left and right restrictions are qualified. “Left || Right” means at least one of the left and right restrictions are qualified. The results indicate that punctuation-based pruning makes a boost in the decoding speed with no big loss in BLUE.

4.3 Discussion and Improvement

As can be seen from Fig. 7, the Baseline experiment has a very slow decoding speed. Apart from some difference in decoding speed, the second and third experiments have no big difference. The fourth experiment is the intersection of the second and the third experiment, but it becomes so slow because it treats so many spans as qualified. The fifth experiment is the union of the second and third experiments. It is obvious that the pruning method in the fifth experiment (i.e., Left&&Right group) has the best result, nearly 8 times as fast as the Baseline System.

We know that different punctuations has different functions in a sentence. Some punctuations indicate the coordinating relation, and some indicate the explicative connection, etc. . However, we may omit this kinds of punctuation function to make punctuation-based pruning easier to apply. We can further improve the BLUE value if we can do better conversion of punctuations into detailed and accurate constraints.

In fact, we can also make further improvements in decoding speed in punctuation-based pruning. The above methods just influence span selection, after which we also have to generate the translation hypotheses. Thus, punctuation-based pruning can also be applied on the process of hypotheses generation.

Given a translation span, when generating translation hypotheses in CYK-based decoder, we prefer hypotheses that consist of two small spans with complete meaning. We can treat a span with no punctuation in it as a complete-meaning span. So punctuations can be used to impose constraints to prune small spans with incomplete meanings.

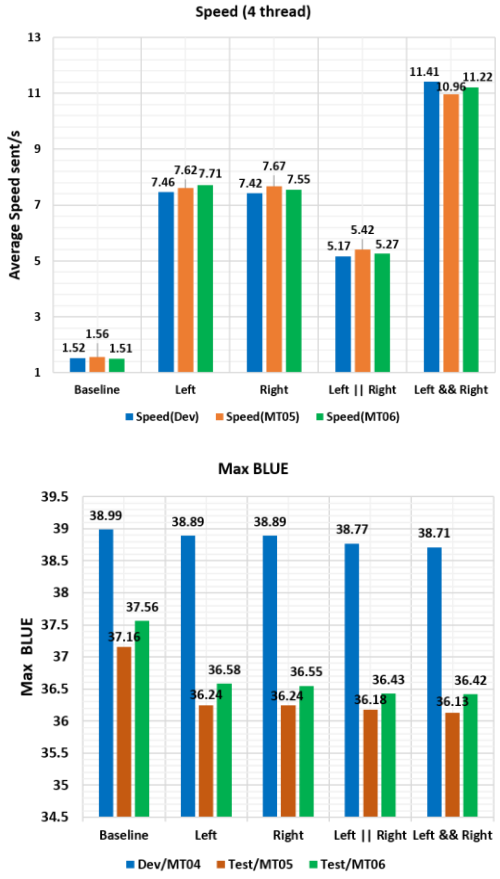

Fig. 7. Experimental Results of Punctuation-based Pruning

In Fig. 6, take span [*4*, *11*] for example, we can use span [*4*, *8*] and span [*9*, *11*] to generate the translation hypothesis of span [*4*, *11*]. However, these two spans are not the expected complete-meaning spans. So we can skip this kind of

hypothesis generating points. The expected spilt of span [*4, 11*] is spans [*4, 6*] and [*7, 11*], or spans [*4, 7*] and [*8, 11*]. In the improved experiment, we treat the case that the splitting point (excluding the beginning and end points of the sentence) is a punctuation as qualified, that is, we first check whether the splitting point is a punctuation or not, then filter out the incomplete splitting points. We did two sets of different experiments on the third stage of CYK decoding, the results of which is shown in Table 1.

Table 1. Experimental Results of Improved Punctuation-based Pruning

Type	Speed (sent/s)	BLUE (max)
Unimproved Punctuation Pruning	3.56	39.21
*Improved Punctuation Pruning	3.75	39.17
**Improved	4.16	39.18

The first experiment is the unimproved punctuation-based pruning. The second experiment examines the beginning and end points of small spans, checks they are punctuation or not; the third experiment just takes into account the end of the first small span and the beginning of the second span. As can be seen, the third experiment gained a better decoding speed, a 17% boost and nearly no loss in BLUE.

5 Phrase Boundary-based Pruning

In hierarchical MT systems, especially those bear the CYK algorithm, a sentence is translated over source-language span units (or sub-strings) from single words to the whole sentence. So one might expect to focus more on the decoding of promising source spans but avoid the computation on the rest that is unlikely to contribute to the final translation. To do this, we introduce labels to indicate whether a word is the beginning/end of a source-language phrase. Here, a phrase is not linguistically motivated; instead, it is a string of words that are projected by MT preferred derivations. These labels can be then used as soft or hard constraints to guide the MT to decode spans with good phrase boundaries and "skip" spans with no clear boundaries.

Normally, given a to-be-translated sentence of n phrases $p_1 p_2 \cdots p_n$, we define that a phrase p_i (1<i<n) is in the class "Beg" if there exists a constituent phrase span $p_i \cdots p_j$ for some (i<j). In the same way, we will regard a phrase p_j (i<j<n) is in the class "End" if there exists a constituent phrase span $p_i \cdots p_j$ for some (j>i). There are two different and separate classification tasks.

Note that the first phrase p_1 and the last phrase p_n are unambiguous since they begin/end a span. The first/last phrase p_1/p_n must begin/end a span covering the whole sentence.

Span boundary can be recognized and judged by some external trained models. We can use the span boundary information (labelled with four different tags) to skip some decoding spans.

5.1 Implementation

We trained a model, which labels the test set with tags automatically. We got the model-training data from word alignment and used the CRF tools to train this model.

5.1.1 Data Preparation

Grammar parser tree is close to the mind of human, but it may not be the way that the translation systems think. So, to get the beginning and end tags that are close to the thinking of translation systems, we first use word alignment to get the phrase spans that can generate a context-free grammar tree, which will be used as model-training data.

[Zhang, et al., 2008] proposed a method to learn the translation boundaries from Synchronous Grammar Tree, using the word-level alignments alone without reference to external syntactic analysis. Since word-level alignment is many-to-many from two language permutation, we can easily use the method in [Chiang, 2005] to get the decomposition tree from word-level alignment without any grammar assistance.

We extract the tags from the decomposition tree obtained from word-level alignment. In context-free grammar tree, we regard the phrase span in the same sub-tree as a translation span. In the translation span, we label the beginning phrase of the phrase sequence with tag B (Beginning Tag), and the end phrase tag E (End Tag). There is an example in Fig. 8 shows these tags.

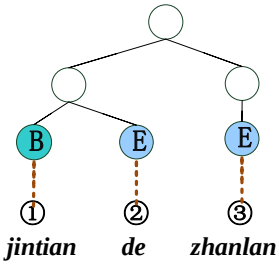

Fig. 8. Example Definition of Beginning and End Tags

However, some phrases may be the beginning and end tags at the same time, which we label with tag D (Double tag). And S (Single Tag) is for those phrases that are neither the beginning tag nor the end tag.

5.1.2 Training

The data we got from word-level alignment is dull, with just the tags and phrases, which will decrease the accuracy of the trained model. To increase the accuracy of the trained model, we use information gained from the grammar tree, such as Part of Speech (PoS) information, the phrase span it can begin or end with.

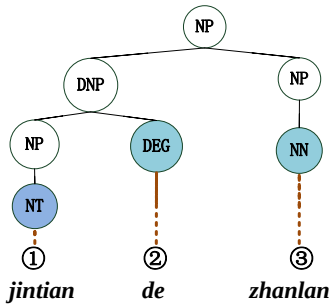

Fig. 9. PoS Attributes

PoS information is illustrated in Fig. 9. Capital letters like 'NP', 'NN', 'DNP', etc. are labels of different grammar parts of a sentence in the parser tree. For example, the phrase '*jintian*' is an NT phrase, and it can be the beginning of a phrase span in DNP subtree or a phrase span in NP subtree.

5.1.3 Pruning Criterion

After trained the model, we label each phrase in the sentence with a tag, which is 'B' or 'E' or 'D' or 'S'. As shown in Fig. 11, suppose the translation span [*beg*, *end*] has a splitting point *mid*, when generating the translation hypothesis, we will check the *end* point (*mid*) of span [*beg*, *mid*] and the *beginning* point (*mid*+1) of span [*mid* + 1, *end*], if the tags are not we expected (for example, the end of span [*beg*, *mid*] is a 'B' tag, while the beginning of span [*mid* +1, *end*] is an 'E' tag), we will skip it. However, we have to impose constraints on the lengths of the pruning phrases to ensure that the short spans can survive the pruning.

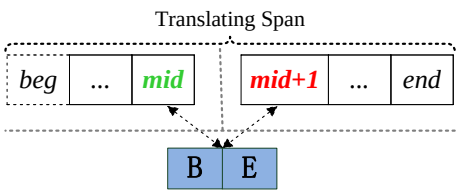

Fig. 11. Criterion Point

5.2 Experimental Results

In this experiment, we set a threshold on the phrase length that can be pruned, and impose this constraint on the second part of CYK (Fig.1). Here, we just prune in the case that the ending of span [*beg*, *mid*] has a 'B' tag and the beginning of span [*mid* +1, *end*] has an 'E' tag.

The baseline system in this experiment used the punctuation-based pruning method (case "Left&&Right") in the first part of CYK (Fig. 1 and Fig. 7). Fig. 10 shows the result of the Dev set.

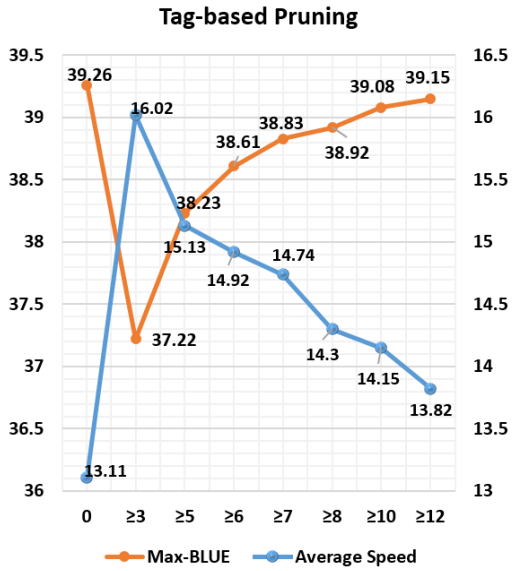

Fig. 10. Experimental Result of Phrase tag-based Pruning

5.3 Discussion and Improvement

In contrast with the first two methods, the result is far from satisfactory. In particular, the improvement for pruning length range 6~7 is 13.8%~15.4%, but a decrease of 0.67~0.97 in BLUE point incurred. There are two factors that may affect this method, and we can improve this method using the following proposals.

1) ***Insufficient or Improper Use of Phrase Tags***

The tags in this experiment are not precise enough to distinguish all the unqualified translation spans from qualified ones. We can add more detailed judgments on phrase tags by classifying the phrase into more categories. For example, if a phrase is neither a 'B' point nor an 'E' point, we can regard it as a single phrase, which can help do more pruning in other circumstances.

2) ***Inaccuracy of the Model***

In fact, the model we trained is not as good as we expected. The accuracy of the model is 80% at the maximum. So the tags it labelled on the

test set are not in accordance with the original training set, that is, we did not get the satisfied tags on the test set. This inaccuracy may account for the disappointing experimental result.

6 Other Possible Pruning Methods

In CYK-based decoders, the hypotheses of spans are generated from smaller spans at each splitting point. Thus, every splitting point needs a time-consuming process to traverse and combine hypotheses from smaller spans. Given a span [*beg*, *end*] with a splitting point *mid*, we can make use of the information of the splitting point *mid* or the relation between two smaller spans (span [*beg*, *mid*] and span [*mid + 1*, *end*]) on both sides of the splitting point, such as beam restrictions on splitting points, mutual information.

6.1 Mutual Information

Mutual information is the relation mutuality between two objects. Since there is some relation between different phrases in a sentence, we can use mutual information to indicate the tightness between different phrase spans. Using the information entropy as the pruning criterion, we can give the generating priority to adjacent spans with high information entropy.

6.2 N-best List Filtering in CYK

In CYK-based decoding, since every splitting point K can generate a set of different translation hypotheses, instead of generating the translation hypotheses from every possible splitting point and giving a detailed and complex scoring of the hypotheses, we can give an instant estimated score for every possible point based on some properties that determine the final score. We can just use the n best estimated points to generate the hypotheses, which can prune a lot of hypotheses with low scores in advance.

7 Related Work

We conduct the experiments in a Chinese-to-English phrase-based model, with beam search and cube pruning applied. Initial methods like beam search and cube pruning are widely used in the optimization of decoding [Koehn, 2003; Robert, 2007]. Improved methods based on beam search and cube pruning have been widely studied for phrased-based [Koehn, 2004; Tillmann and Ney, 2003; Tillmann, 2006] and syntax-based translation models [Chiang, 2007; Huang and Chiang, 2007; Watanabe et al., 2006; Huang and Mi, 2010].

Our work has several differences from previous efforts.

In parser tree-based pruning, [Hirofumi, 2008] used the source tree constraints to restrict the word reordering in Inversion Transduction Grammar (ITG)-supported decoders. However, we use the parser tree information to restrict the translating path and the reordering between irrelevant phrases, and implement it in a simple way.

In punctuation-based pruning, [Valentin, 2011] used the punctuation to restrict the reordering in dependency parsing. However, we treat punctuations as symbols for judging the intactness of span meanings, and use them to prune the decoding search space.

In phrase boundary-based pruning, [Xiong, 2010] used the boundary segment tags as the soft constraints for recognizing the translation zone, and [Wenduan, 2013] used the phrase tags to do context-sensitive pruning in string-to-tree models. We use this kind of labels as hard constraints in CYK-based decoders and verify their effects on different span lengths.

8 Conclusion

CYK is a popular algorithm for implementing decoders for SMT systems, but it has a cubic time complexity. We compared three popular pruning methods for CYK-based decoding that try to relieve this complexity

The grammar parser tree-based pruning gained outstanding decoding speed, but it somewhat decreased the translation quality. The decoding paths that the parser tree represents are possibly not the optimal ones in the view of machine translation systems. Future work can focus on increasing the coverage of the key decoding paths that gained from the parser trees.

Punctuation-based pruning achieved an overall satisfying result in both decoding speed and translation quality. However, phrase boundary-based pruning did not get any satisfied result in either decoding speed or translation quality. To improve this method, future work can focus on increasing the accuracy of the model and on making better use of the phrase tags.

We also proposed some other novel pruning methods for CYK-based decoding. Future work would be to implement these methods on top of the same toolkit as we used and compare their results with those of current pruning methods.

Acknowledgements

This work was supported in part by the National Science Foundation of China (Grants 61272376, 61300097 and 61432013). We would like to thank the anonymous reviewers and Jingbo Zhu for their pertinent comments and helpful discussion. We also would like to thank Qiang Li for building parts of the baseline system.

References

1. Feng Y., Mi H., Liu Y., and Liu Q. *An efficient shift-reduce decoding algorithm for phrased-based machine translation.* In Proceedings of the 23rd International Conference on Computational Linguistics: Posters (COLING '10). (2010): 285-293.

2. O. Daniel, G. Varea, and F. Casacuberta. *An Empirical Comparison of Stack-Based Decoding Algorithms for Statistical Machine Translation*. 2003:654-663.

3. Y. Steve. "*Large Vocabulary Continuous Speech Recognition: a Review.*" IEEE Signal Processing Magazine 21.4(1996):786 - 797.

4. J. Cheppalier, and M. Rajman. "*A generalized CYK algorithm for parsing stochastic CFG.*" In Proc. of Tabulation in Parsing and Deduction (TAPD)' 1998:98

5. V. David, et al. "*Jane: an advanced freely available hierarchical machine translation toolkit.*" Machine Translation 26.3(2012):197-216.

6. F. López, M. Corchado, F. De Paz, S. Rodríguez, J. Bajo *Statistical Machine Translation Using the Self-Organizing Map* Distributed Computing and Artificial Intelligence Advances in Intelligent and Soft Computing Volume 79, 2010, pp 131-138

7. Xiao, T., Zhu, J., Zhang, H., & Li, Q. (2012). NiuTrans: *An Open Source Toolkit for Phrase-based and Syntax-based Machine Translation*. Jeju, Republic of Korea (pp.19-24).

8. K. Philipp, J. Och, and D. Marcu. "*Statistical phrase-based translation.*" Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1Association for Computational Linguistics, 2003:127--133.

9. F. Och. "*Minimum Error Rate Training in Statistical Machine Translation.*" Proc ACL32.17 (2003):701-711.

10. C. David. "*Hierarchical Phrase-Based Translation.*" Computational Linguistics 33.2(2007):201-228.

11. K. Kevin. "*Decoding Complexity in Word-Replacement Translation Models.*" Computational Linguistics 25.4(1999):607--615.

12. K. Philipp. *Pharaoh: A Beam Search Decoder for Phrase-Based Statistical Machine Translation Models*. Machine Translation: Springer Berlin Heidelberg, 2004:115-124.

13. Robert C. Moore and Chris Quirk *Faster Beam-Search Decoding for Phrasal Statistical Machine Translation* Proceedings of MT Summit XI 2007.

14. G. Andrea, and J. Henderson. "*Faster Cube Pruning.*" In Proceedings of the International Workshop on Spoken Language Translation *(IWSLT* (2010):267--274.

15. Zhang H. , D. Gildea, and D. Chiang. "*Extracting Synchronous Grammar Rules From Word-Level Alignments in Linear Time*. "In Proceedings of the 22nd International Conference on Computational Linguistics ((2008) COLING-08).

16. C. David. 2005. *A hierarchical phrase-based model for statistical machine translation*. In Proceedings of ACL 2005, pages 263–270.

17. Xiong D., Zhang M， Li H. *Learning Translation Boundaries for Phrase-Based Decoding* Coling 2010: Poster Volume, pages 383–390

18. Xu W., Zhang Y., P. Williams and P. Koehn, *Learning to Prune: Context-Sensitive Pruning for Syntactic MT*, Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pages (2013): 352–357