A Structured Language Model for Incremental Tree-to-String Translation

Heng Yu[1]
[1]Institute of Computing Technology. CAS
University of Chinese Academy of Sciences
yuheng@ict.ac.cn

Haitao Mi
T.J. Watson Research Center
IBM
hmi@us.ibm.com

Liang Huang
Queens College & Grad. Center
City University of New York
huang@cs.qc.cuny.edu

Qun Liu[1,2]
[2]Centre for Next Generation Localisation.
Faculty of Engineering and Computing
Dublin City University
qliu@computing.dcu.ie

Abstract

Tree-to-string systems have gained significant popularity thanks to their simplicity and efficiency by exploring the source syntax information, but they lack in the target syntax to guarantee the grammaticality of the output. Instead of using complex tree-to-tree models, we integrate a structured language model, a left-to-right shift-reduce parser in specific, into an incremental tree-to-string model, and introduce an efficient grouping and pruning mechanism for this integration. Large-scale experiments on various Chinese-English test sets show that with a reasonable speed our method gains an average improvement of 0.7 points in terms of (T -B)/2 than a state-of-the-art tree-to-string system.

1 Introduction

Tree-to-string models (Liu et al., 2006; Huang et al., 2006) have made promising progress and gained significant popularity in recent years, as they run faster than string-to-tree counterparts (e.g. (Galley et al., 2006)), and do not need binarized grammars. Especially, Huang and Mi (2010) make it much faster by proposing an incremental tree-to-string model, which generates the target translation exactly in a left-to-right manner. Although, tree-to-string models have made those progresses, they can not utilize the target syntax information to guarantee the grammaticality of the output, as they only generate strings on the target side.

One direct approach to handle this problem is to extend tree-to-string models into complex tree-to-tree models (e.g. (Quirk et al., 2005; Liu et al., 2009; Mi and Liu, 2010)). However, tree-to-tree approaches still significantly under-perform than tree-to-string systems due to the poor rule coverage (Liu et al., 2009) and bi-parsing failures (Liu et al., 2009; Mi and Liu, 2010).

Another potential solution is to use structured language models (S) (Chelba and Jelinek, 2000; Charniak et al., 2003; Post and Gildea, 2008; Post and Gildea, 2009), as the monolingual S has achieved better perplexity than the traditional *n*-gram word sequence model. More importantly, the S is independent of any translation model. Thus, integrating a S into a tree-to-string model will not face the problems that tree-to-tree models have. However, integration is not easy, as the following two questions arise. First, the search space grows significantly, as a partial translation has a lot of syntax structures. Second, hypotheses in the same bin may not be comparable, since their syntactic structures may not be comparable, and the future costs are hard to estimate. Hassan et al. (2009) skip those problems by only keeping the best parsing structure for each hypothesis.

In this paper, we integrate a shift-reduce parser into an incremental tree-to-string model, and introduce an efficient grouping and pruning method to handle the growing search space and incomparable hypotheses problems. Large-scale experiments on various Chinese-English test sets show that with a rea-

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 1133–1143, Dublin, Ireland, August 23-29 2014.

sonable speed our method gains an average improvement of 0.7 points in terms of (T -B)/2 than a state-of-the-art tree-to-string system.

2 Linear-time Shift-reduce Parsing

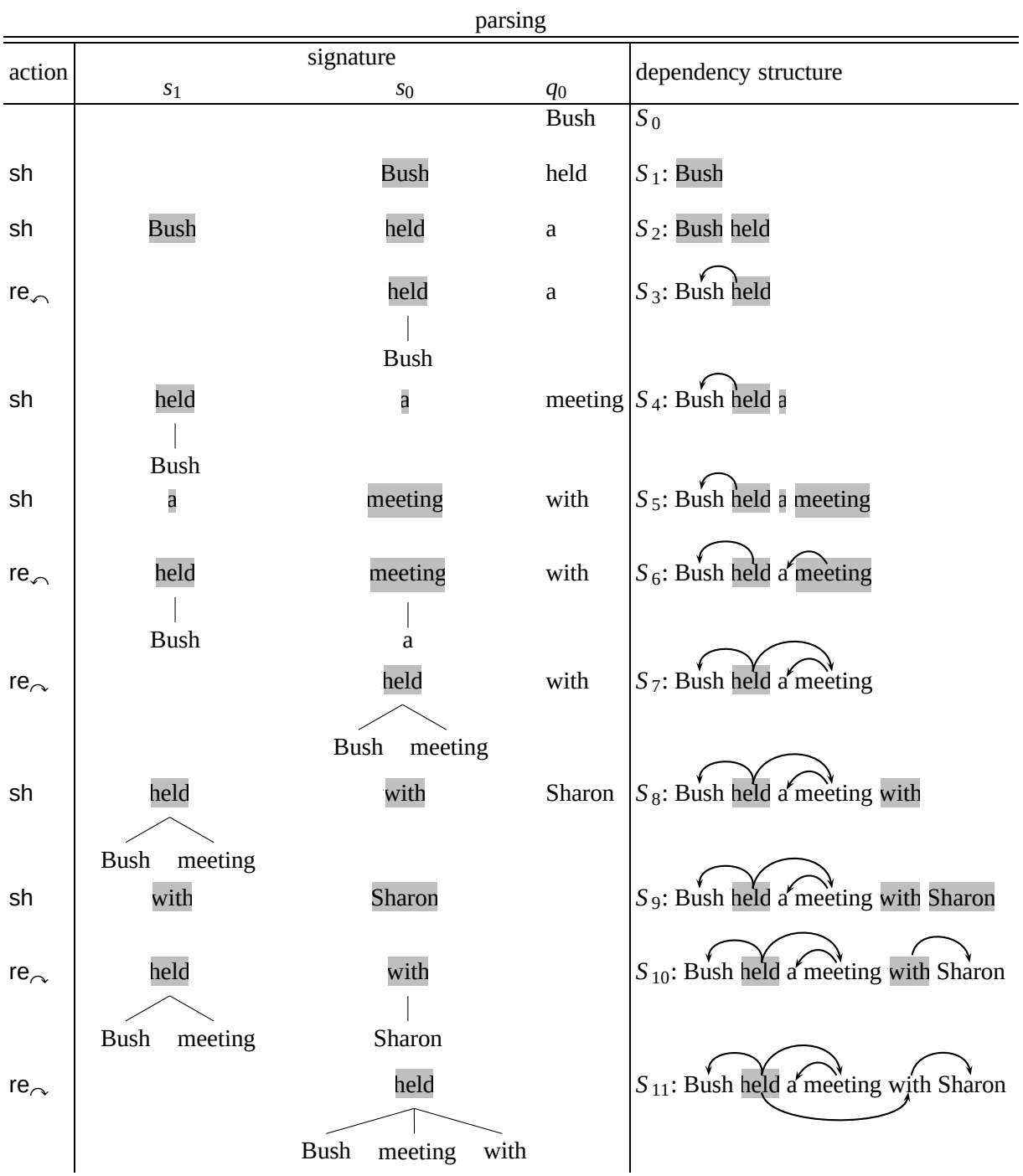


Figure 1: Linear-time left-to-right dependency parsing.

A shift-reduce parser performs a left-to-right scan of the input sentence, and at each *parsing step*, chooses one of two *parsing actions*: either **shift** (sh) the current word onto the stack, or **reduce** (re) the top two (or more) items at the end of the stack (Aho and Ullman, 1972). In the dependency parsing scenario, the reduce action is further divided into two cases: **left-reduce** (re$_\curvearrowleft$) and **right-reduce** (re$_\curvearrowright$), depending on which one of the two items becomes the head after reduction. Each parsing derivation can be represented by a sequence of parsing actions.

2.1 Shift-reduce Dependency Parsing

We will use the following sentence as the running example:

Bush held a meeting with Sharon

Given an input sentence **e**, where e_i is the ith token, $e_i...e_j$ is the substring of **e** from i to j, a shift-reduce parser searches for a dependency tree with a sequence of shift-reduce moves (see Figure 1). Starting from an initial structure S_0, we first shift (sh) a word e_1, "Bush", onto the parsing stack s_0, and form a structure S_1 with a singleton tree. Then e_2, "held", is shifted, and there are two or more structures in the parsing stack, we can use $\mathsf{re}_{\curvearrowleft}$ or $\mathsf{re}_{\curvearrowright}$ step to combine the top two trees on the stack, replace them with dependency structure $e_1 \curvearrowleft e_0$ or $e_1 \curvearrowright e_0$ (shown as S_3), and add one more dependency edge between e_0 and e_1.

Note that the shade nodes are exposed heads on which $\mathsf{re}_{\curvearrowleft}$ or $\mathsf{re}_{\curvearrowright}$ parsing actions can be performed. The middle columns in Figure 1 are the parsing signatures: q_0 (parsing queue), s_0 and s_1 (parsing stack), where s_0 and s_1 only have one level dependency. Take the line of S_{11} for example, "a" is not in the signature. As each action results in an update of cost, we can pick the best one (or few, with beam) after each action. Costs are accumulated in each step by extracting contextual features from the structure and the action. As the sentence gets longer, the number of partial structures generated at each steps grows exponentially, which makes it impossible to search all of the hypothesis. In practice, we usually use beam search instead.

(a)	atomic features	
	s_0.w	s_0.t
	s_1.w	s_1.t
	s_0.lc.t	s_0.rc.t
	q_0.w	q_0.t

(b)	feature templates		
unigram	s_0.w	s_0.t	s_0.w ∘ s_0.t
	s_1.w	s_1.t	s_1.w ∘ s_1.t
	q_0.w	q_0.t	q_0.w ∘ q_0.t
bigram	s_0.w ∘ s_1.w		s_0.t ∘ s_1.t
	s_0.t ∘ q_0.t		s_0.w ∘ s_0.t ∘ s_1.t
	s_0.w ∘ s_1.w ∘ s_1.t		s_0.t ∘ s_1.w ∘ s_1.t
	s_0.w ∘ s_0.t ∘ s_1.w		
trigram	s_0.t ∘ s_1.t ∘ q_0.t		s_1.t ∘ s_0.t ∘ s_0.lc.t
	s_1.t ∘ s_0.t ∘ q_0.t		s_1.t ∘ s_0.t ∘ s_0.rc.t

(c)

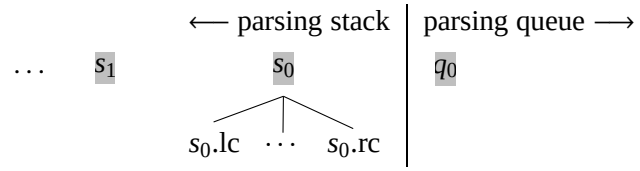


Table 1: (a) atomic features, used for parsing signatures. (b): parsing feature templates, adapted from Huang and Sagae (2010). x.w and x.t denotes the root word and POS tag of the partial dependency tree, x.lc and x.rc denote x's leftmost and rightmost child respectively. (c) the feature window.

2.2 Features

We view features as "abstractions" or (partial) observations of the current structure. Feature templates f are functions that draw information from the feature window, consisting of current partial tree and first word to be processed. All Feature functions are listed in Table 1(b), which is a conjunction of atomic

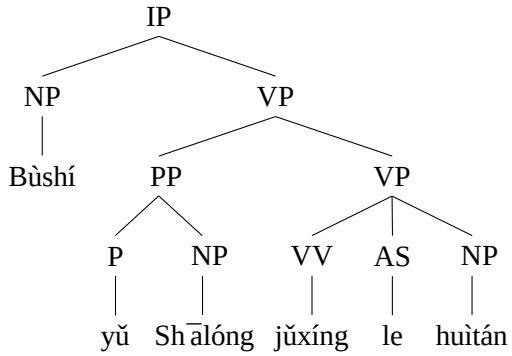


Figure 2: A parse tree

features in Table 1(a). To decide which action is the best of the current structure, we perform a three-way classification based on *f*, and conjoin these feature instances with each action:

$$[f \circ (\text{action=sh/re}_{\curvearrowright}/\text{re}_{\curvearrowleft})]$$

We extract all the feature templates from training data, and use the average perceptron algorithm and early-update strategy (Collins and Roark, 2004; Huang et al., 2012) to train the model.

3 Incremental Tree-to-string Translation with S

The incremental tree-to-string decoding (Huang and Mi, 2010) performs translation in two separate steps: parsing and decoding. A parser first parses the source language input into a 1-best tree in Figure 2, and the linear incremental decoder then searches for the best derivation that generates a target-language string in strictly left-to-right manner. Figure 3 works out the full running example, and we describe it in the following section.

3.1 Decoding with S

Since the incremental tree-to-string model generates translation in strictly left-to-right fashion, and the shift-reduce dependency parser also processes an input sentence in left-to-right order, it is intuitive to combine them together. The last two columns in Figure 3 show the dependency structures for the corresponding hypotheses. Start at the root *translation stack* with a dot **.** before the root node IP:

$$[\mathbf{.}\ \text{IP}\ ],$$

we first **predict** (pr) with rule r_1,

$$(r_1) \qquad \text{IP}\ (x_1\text{:NP}\ x_2\text{:VP}) \rightarrow\ x_1\ x_2,$$

and push its English-side to the translation stack, with variables replaced by matched tree nodes, here x_1 for NP and x_2 for VP. Since this *translation action* does not generate any translation string, we don't perform any dependency parsing actions. So we have the following translation stack

$$[\mathbf{.}\ \text{IP}\ ][\mathbf{.}\ \text{NP VP}],$$

where the dot **.** indicates the next symbol to process in the English word-order. Since node NP is the next symbol, we then predict with rule r_2,

$$(r_2) \qquad \text{NP(Bùshí)} \rightarrow \text{Bush},$$

and add it to the translation stack:

$$[\mathbf{.}\ \text{IP}\ ]\ [\mathbf{.}\ \text{NP VP}\ ]\ [\mathbf{.}\ \text{Bush}]$$

Since the symbol right after the dot in the top rule is a word, we **scan** (sc) it, and append it to the current translation, which results in the new translation stack

$$[\mathbf{.}\ \text{IP}\ ]\ [\mathbf{.}\ \text{NP VP}\ ]\ [\text{Bush}\ \mathbf{.}\ ]$$

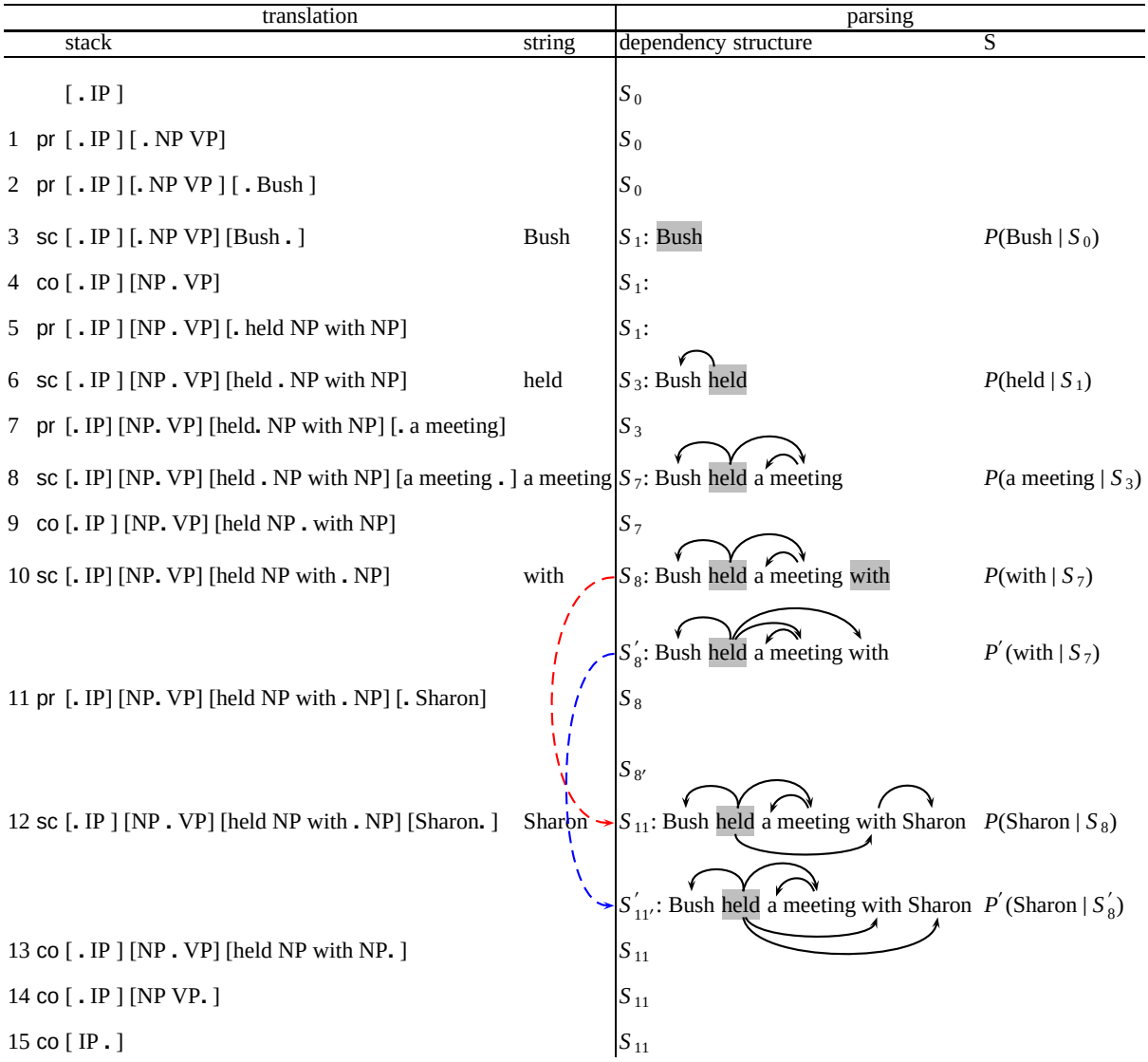


Figure 3: Simulation of the integraton of an S into an incremental tree-to-string decoding. The first column is the line number. The second column shows the translation actions: predict (pr), scan (sc), and complete (co). S_i denotes a dependency parsing structure. The shaded nodes are exposed roots of S_i.

Immediately after each sc translation action, our shift-reduce parser is triggered. Here, our parser applies the parsing action sh, and shift "Bush" into a partial dependency structure S_1 as a root "Bush" (shaded node) in Figure 3. Now the top rule on the translation stack has finished (dot is at the end), so we **complete** (co) it, pop the top rule and advance the dot in the second-to-top rule, denoting that NP is completed:

[. IP] [NP . VP].

Following this procedure, we have a dependency structure S_3 after we scan (sc) the word "held" and take a shift (sh) and a left reduce ($\text{re}_{\curvearrowleft}$) parsing actions. The shaded node "held" means exposed roots, that the shift-reduce parser takes actions on.

Following Huang and Mi (2010), the hypotheses with same *translation step*[1] fall into the same bin. Thus, only the prediction (pr) actions actually make a jump from a bin to another. Here line 2 to 4 fall into one bin (translation step = 4, as there are 4 nodes, IP, NP, VP and Bùshí, in the source tree are covered). Similarly, lines from 7 to 10 fall into another bin (translation step = 15).

[1]The step number is defined by the number of tree nodes covered in the source tree, and it is not equal to the number of translation actions taken so far.

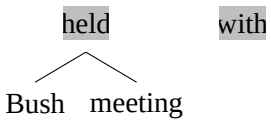
Noted that as we number the bins by the translation step, only pr actions make progress, the sc and co actions are treated as ”closure” operators in practice. Thus we always do as many sc/co actions as possible immediately after a pr step until the symbol after the dot is another non-terminal. The total number of bins is equal to the size of the parse tree, and each hypothesis has a constant number of outgoing hyper-edges to predict, so the time complexity is linear in the sentence length.

After adding our S to this translation, an interesting branch occurs after we scan the word “with”, we have two different partial dependency structures S_8 and S_8' for the same translation. If we denote $N(S_i)$ as the number of re actions that S_i takes, $N(S_8)$ is 3, while $N(S_8')$ is 4. Here $N(S_i)$ does not take into account the number of sh parsing actions, since all partial structures with same translations should shift the same number of translations. Thus, $N(S_i)$ determines the score of dependency structures, and only the hypotheses with same $N(S_i)$ are comparable to each other. In this case, we should distinguish S_8 with S_8', and if we make a prediction over the hypothesis of S_8, we can reach the correct parsing state S_{11} (shown in the red dashed line in Figure 3).

So the key problem of our integration is that, after each translation step, we will apply different sequences of parsing actions, which result in different and incomparable dependency structures with the same translation. In the following two Sections, we introduce three ways for this integration.

3.2 Naïve: Adding Parsing Signatures into Translation Signatures

One straightforward approach is to add the parsing signatures (in Figure 1) of each dependency structure (in Figure 1 and Figure 3) to translation signatures. Here, we only take into account of the s_0 and s_1 in the parsing stack, as the q_0 is the future word that is not available in translation strings. For example, the dependency structure S_8 has parsing signatures:



We add those information to its translation signature, and only the hypothesis that have same translation and parsing signatures can be recombined.

So, in each translation bin, different dependency structures with same translation strings are treated as different hypothesis, and all the hypothesis are sorted and ranked in the same way. For example, S_8 and S_8' are compared in the bin, and we only keep top b (the beam size) hypothesis for each bin.

Obviously, this simple approach suffers from the incomparable problem for those hypothesis that have different number of parsing actions (e.g. S_8 and S_8'). Moreover, it may result in very low translation variance in each beam.

3.3 Best-parse: Keeping the Best Dependency Structure for Each Translation

Following Hassan et al. (2009), we only keep the best parsing tree for each translation. That means after a consecutive translation sc actions, our shift-reduce parser applies all the possible parsing actions, and generates a set of new partial dependency structures. Then we only choose the best one with the highest S score, and only use this dependency structure for future predictions.

For example, for the translation in line 10 in Figure 3, we only keep S_8, if the parsing score of S_8 is higher than S_8', although they are not comparable. Another complicate example is shown in Figure 4, within the translation step 15, there are many alternatives with different parsing structures for the same translation (“a meeting with”) in the third column, but we can only choose the top one in the final.

3.4 Grouping: Regrouping Hypothesis by $N(S)$ in Each Bin

In order to do comparable sorting and pruning, our basic idea is to regroup those hypotheses in a same bin into small groups by $N(S)$. For each translation, we first apply all the possible parsing actions, and generate all dependency structures. Then we regroup all the hypothesis with different dependency structures based on the size of $N(S)$.

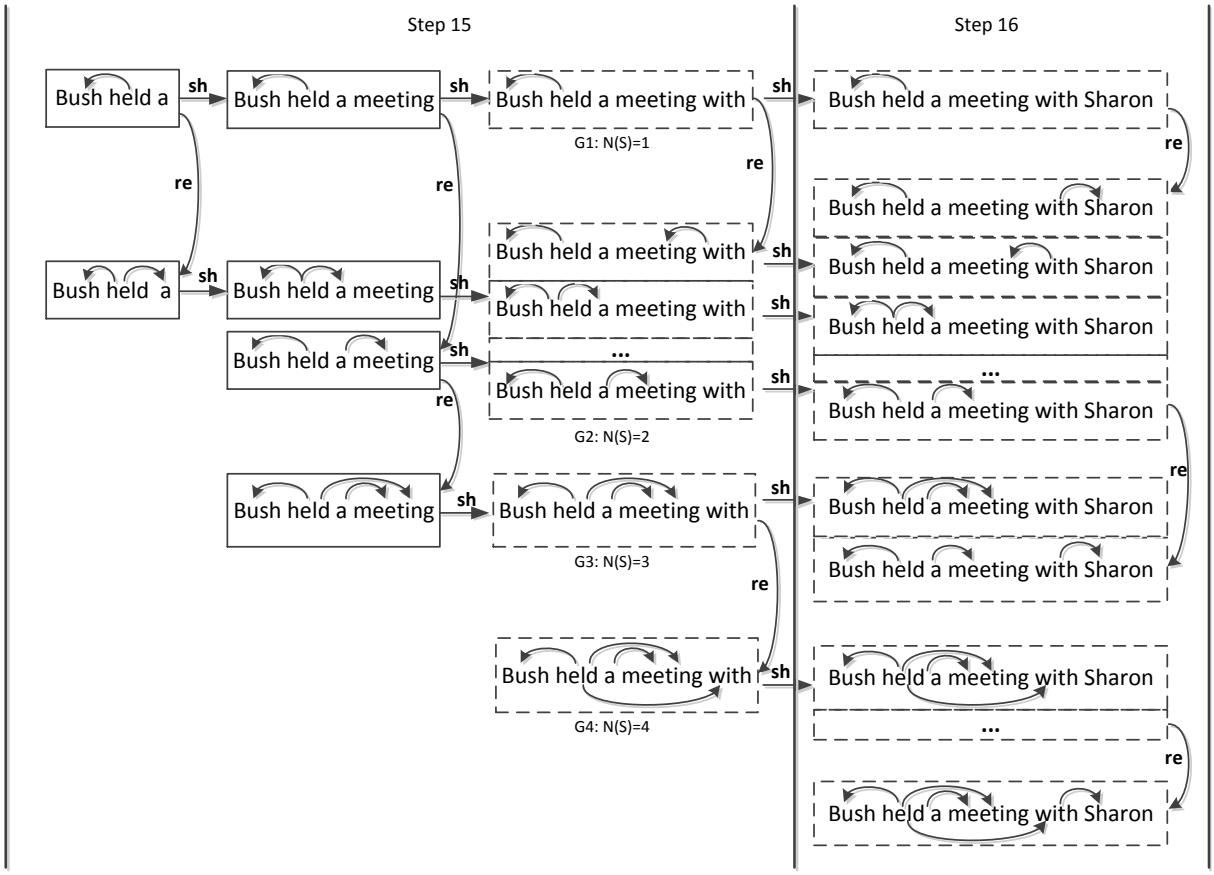


Figure 4: Multi-beam structures of two bins with different translation steps (15 and 16). The first three columns show the parsing movements in bin 15. Each dashed box is a group based on the number of reduce actions over the new translation strings ("a meeting with" for bin 15, and "Sharon" for bin 16). G_2 means two reduce actions have been applied. After this regrouping, we perform the pruning in two phases: 1) keep top b states in each group, and labeled each group with the state with the highest parsing score in this group; 2) sort the different groups, and keep top g groups.

For example, Figure 4 shows two bins with two different translation steps (15 and 16). In bin 15, the graph shows the parsing movements after we scan three new words ("a", "meeting", and "with"). The parsing sh action happens from a parsing state in one column to another state in the next column, while re happens from a state to another state in the same column. The third column in bin 15 lists some partial dependency structures that have all new words parsed. Here each dashed box is a group of hypothesis with a same $N(S)$, e.g. the G_2 contains all the dependency structures that have two reduce actions after parsed all the new words. Then, we sort and prune each group by the beam size b, and each group labeled as the highest hypothesis in this group. Finally, we sort those groups and only keep top g groups for the future predictions. Again, in Figure 4, we can keep the whole group G_3 and partial group of G_2 if $b = 2$. In our experiments, we set the group size g to 5.

3.5 Log-linear Model

We integrate our dependency parser into the log-linear model as an additional feature. So the decoder searches for the best translation $\mathbf{e}^*$ with a latent tree structure (evaluated by our S) according to the following equation:

$$\mathbf{e}^* = \underset{\mathbf{e} \in \mathbf{E}}{\operatorname{argmax}} \exp(\mathrm{S}(\mathbf{e}) \cdot w_s + \sum_i f_i \cdot w_i) \quad (1)$$

where S (**e**) is the dependency parsing score calculated by our parser, w_s is the weight of S (**e**), f_i are the features in the baseline model and w_i are the weights.

4 Experiments

4.1 Data Preparation

The training corpus consists of 1.5M sentence pairs with 38M/32M words of Chinese/English, respectively. We use the NIST evaluation sets of MT06 as our development set, and MT03, 04, 05, and 08 (newswire portion) as our test sets. We word-aligned the training data using GIZA++ with refinement option "grow-diag-and" (Koehn et al., 2003), and then parsed the Chinese sentences using the Berkeley parser (Petrov and Klein, 2007). we applied the algorithm of Galley et al. (2004) to extract tree-to-string translation rules. Our trigram word language model was trained on the target side of the training corpus using the SRILM toolkit (Stolcke, 2002) with modified Kneser-Ney smoothing. At decoding time, we again parse the input sentences using the Berkeley parser, and convert them into translation forests using rule pattern-matching (Mi et al., 2008).

Our baseline system is the incremental tree-to-string decoder of Huang and Mi (2010). We use the same feature set shown in Huang and Mi (2010), and tune all the weights using minimum error-rate training (Och, 2003) to maximize the B score on the development set.

Our dependency parser is an implementation of the "arc-standard" shift-reduce parser (Nivre, 2004), and it is trained on the standard split of English Penn Tree-bank (PTB): Sections 02-21 as the training set, Section 22 as the held-out set, and Section 23 as the test set. Using the same features as Huang and Sagae (2010), our dependency parser achieves a similar performance as Huang and Sagae (2010). We add the structured language model as an additional feature into the baseline system.

We evaluate translation quality using case-insensitive IBM B -4, calculated by the script `mteval-v13a.pl`. We also report the T scores.

4.2 Complete Comparisons on MT08

To explore the soundness of our approach, we carry out some experiments in Table 2. With a beam size 100, the baseline decoder achieves a B score of 21.06 with a speed of 1.7 seconds per sentence.

Since our dependency parser is trained on the English PTB, which is not included in the MT training set, there is a chance that the gain of B score is due to the increase of new *n*-grams in the PTB data. In order to rule out this possibility, we use the tool SRILM to train another tri-gram language model on English PTB and use it as a secondary language model for the decoder. The B score is 21.10, which is similar to the baseline result. Thus we can conclude that any gain of the following +S experiments is not because of the using of the additional English PTB.

Our second experiment re-ranks the 100-best translations of the baseline with our structured language model trained on PTB. The improvement is less than 0.2 B , which is not statistically significant, as the search space for re-ranking is relatively small compared with the decoding space.

As shown in Section 3, we have three different ways to integrate an S to the baseline system:

- **naïve**: adding the parsing signature to the translation signature;
- **best-parse**: keeping the best dependency structure for each translation;
- **grouping**: regrouping the hypothesis by $N(S)$ in each bin.

The naïve approach achieves a B score of 19.12, which is significantly lower than the baseline. The main reason is that adding parsing signatures leads to very restricted translation variance in each beam. We also tried to increase the beam size to 1000, but we do not see any improvement.

The fourth line in Table 2 shows the result of the best-parse (Hassan et al., 2009). This approach only slows the speed by a factor of two, but the improvement is not statistically significant. We manually looked into some dependency trees this approach generates, and found this approach always introduce local parsing errors.

The last line shows our efficient beam grouping scheme with a grouping size 5, it achieves a significant improvement with an acceptable speed, which is about 6 times slower than the baseline system.

System		B	Speed
baseline		21.06	1.7
+S	re-ranking	21.23	1.73
	naïve	19.12	2.6
	best-parse	21.30	3.4
	grouping (g=5)	**21.64**	10.6

Table 2: Results on MT08. The bold score is significantly better than the baseline result at level $p < 0.05$.

System	MT03		MT04		MT05		MT08		Avg.
	B	(T-B)/2	B	(T-B)/2	B	(T-B)/2	B	(T-B)/2	(T-B)/2
baseline	19.94	10.73	22.03	18.63	19.92	11.45	21.06	10.37	12.80
+S	**21.49**	**9.44**	22.33	18.38	**20.51**	**10.71**	**21.64**	**9.88**	**12.10**

Table 3: Results on all test sets. Bold scores are significantly better than the baseline system ($p < 0.5$).

4.3 Final Results on All Test Sets

Table 3 shows our main results on all test sets. Our method gains an average improvement of 0.7 points in terms of (T-B)/2. Results on NIST MT 03, 05, and 08 are statistically significant with $p < 0.05$, using bootstrap re-sampling with 1000 samples (Koehn, 2004). The average decoding speed is about 10 times slower than the baseline.

5 Related Work

The work of Schwartz et al. (2011) is similar in spirit to ours. We are different in the following ways. First, they integrate an S into a phrase-based system (Koehn et al., 2003), we pay more attention to a syntax-based system. Second, their approach slowdowns the speed at near 2000 times, thus, they can only tune their system on short sentences less than 20 words. Furthermore, their results are from a much bigger beam (10 times larger than their baseline), so it is not clear which factor contributes more, the larger beam size or the S . In contrast, our approach gains significant improvements over a state-of-the-art tree-to-string baseline at a reasonable speed, about 6 times slower. And we answer some questions beyond their work.

Hassan et al. (2009) incorporate a linear-time CCG parser into a DTM system, and achieve a significant improvement. Different from their work, we pay more attention to the dependency parser, and we also test this approach in our experiments. As they only keep 1-best parsing states during the decoding, they are suffering from the local parsing errors.

Galley and Manning (2009) adapt the maximum spanning tree (MST) parser of McDonald et al. (2005) to an incremental dependency parsing, and incorporate it into a phrase-based system. But this incremental parser remains in quadratic time.

Besides, there are also some other efforts that are less closely related to ours. Shen et al. (2008) and Mi and Liu (2010) develop a generative dependency language model for string-to-dependency and tree-to-tree models. But they need parse the target side first, and encode target syntactic structures in translation rules. Both papers integrate dependency structures into translation model, we instead model the dependency structures with a monolingual parsing model over translation strings.

6 Conclusion

In this paper, we presented an efficient algorithm to integrate a structured language model (an incremental shift-reduce parser in specific) into an incremental tree-to-string system. We calculate the structured language model scores incrementally at the decoding step, rather than re-scoring a complete translation. Our experiments suggest that it is important to design efficient pruning strategies, which have been

overlooked in previous work. Experimental results on large-scale data set show that our approach significantly improves the translation quality at a reasonable slower speed than a state-of-the-art tree-to-string system.

The structured language model introduced in our work only takes into account the target string, and ignores the reordering information in the source side. Thus, our future work seeks to incorporate more source side syntax information to guide the parsing of the target side, and tune a structured language model for both B and paring accuracy. Another potential work lies in the more efficient searching and pruning algorithms for integration.

Acknowledgments

We thank the three anonymous reviewers for helpful suggestions, and Dan Gildea and Licheng Fang for discussions. Yu and Liu were supported in part by CAS Action Plan for the Development of Western China (No. KGZD-EW-501) and a grant from Huawei Noah's Ark Lab, Hong Kong. Liu was partially supported by the Science Foundation Ireland (Grant No. 07/CE/I1142) as part of the CNGL at Dublin City University. Huang was supported by DARPA FA8750-13-2-0041 (DEFT), a Google Faculty Research Award, and a PSC-CUNY Award, and Mi by DARPA HR0011-12-C-0015. The views and findings in this paper are those of the authors and are not endorsed by the US or Chinese governments.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. Parsing of series in automatic computation. In *The Theory of Parsing, Translation, and Compiling*, page Volume I.

Eugene Charniak, Kevin Knight, and Kenji Yamada. 2003. Syntax-based language models for statistical machine translation. In *Proceedings of MT Summit IX. Intl. Assoc. for Machine Translation*.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. volume 14, pages 283 – 332.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*.

Michel Galley and Christopher D. Manning. 2009. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of ACL 2009 and AFNLP*, pages 773–781, Suntec, Singapore, August. Association for Computational Linguistics.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT-NAACL*, pages 273–280.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of COLING-ACL*, pages 961–968.

Hany Hassan, Khalil Sima'an, and Andy Way. 2009. A syntactified direct translation model with linear-time decoding. In *Proceedings of EMNLP 2009*, pages 1182–1191, Singapore, August. Association for Computational Linguistics.

Liang Huang and Haitao Mi. 2010. Efficient incremental decoding for tree-to-string translation. In *Proceedings of EMNLP*, pages 273–283.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL 2010*, pages 1077–1086, Uppsala, Sweden, July. Association for Computational Linguistics.

Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of AMTA*, pages 66–73.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of NAACL 2012*, Montreal, Quebec.

Philipp Koehn, Franz Joseph Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of NAACL*, pages 127–133.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of EMNLP*, pages 388–395.

Yang Liu, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment template for statistical machine translation. In *Proceedings of COLING-ACL*, pages 609–616.

Yang Liu, Yajuan Lü, and Qun Liu. 2009. Improving tree-to-tree translation with packed forests. In *Proceedings of ACL/IJCNLP*, pages 558–566, Suntec, Singapore, August.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530, Vancouver, British Columbia, Canada, October.

Haitao Mi and Qun Liu. 2010. Constituency to dependency translation with forests. In *Proceedings of ACL*, pages 1433–1442, Uppsala, Sweden, July.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proceedings of ACL: HLT*, pages 192–199.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain, July. Association for Computational Linguistics.

Franz Joseph Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT-NAACL*, pages 404–411.

Matt Post and Daniel Gildea. 2008. Language modeling with tree substitution grammars. In *Proceedings of AMTA*.

Matt Post and Daniel Gildea. 2009. Language modeling with tree substitution grammars. In *Proceedings of NIPS workshop on Grammar Induction, Representation of Language, and Language Learning*.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd ACL*, Ann Arbor, MI, June.

Lane Schwartz, Chris Callison-Burch, William Schuler, and Stephen Wu. 2011. Incremental syntactic language models for phrase-based translation. In *Proceedings of ACL 2011*, pages 620–631, June.

Libin Shen, Jinxi Xu, and Ralph Weischedel. 2008. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL-08: HLT*, pages 577–585, Columbus, Ohio, June. Association for Computational Linguistics.

Andreas Stolcke. 2002. SRILM – an extensible language modeling toolkit. In *Proceedings of ICSLP*, volume 30, pages 901–904.