TSUKU Statistical Machine Translation System for the NTCIR-10 PatentMT Task

Zhongyuan Zhu, Jun-ya Norimatsu, Toru Tanaka, Takashi Inui, Mikio Yamamoto
University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
{raphael, norimatsu, imasaka}@mibel.cs.tsukuba.ac.jp, {inui, myama}@cs.tsukuba.ac.jp

ABSTRACT
This paper describes details of the TSUKU machine translation system in the NTCIR-10 PatentMT task [8] . This system is an implementation of our tree-to-string statistical machine translation model that combines a context-free grammar (CFG) parse tree and a dependency parse tree.

Categories and Subject Descriptors
I.2.7 [**Natural Language Processing**]: Machine Translation

General Terms
Algorithms

Keywords
Tree combination

Team Name
[TSUKU]

Subtasks/Languages
[English-to-Japanese]

External Resources Used
[GIZA++][Berkeley Parser][Stanford Dependency Parser]

1. INTRODUCTION
Context-free grammar (CFG) parse trees are widely used in tree-to-string statistical machine translation models and generally provide comprehensive linguistic information, including the structure of a given natural language sentence. On the other hand, dependency parse trees provide semantic information about the relations between words, which would be helpful to capture long-range word relationships in long sentence translation tasks.

Many researchers are using dependency parse trees on the source side or the target side [15] to achieve better translation accuracy and better results in human evaluation.

In this paper, we describe a tree-to-string translation system in which the input is a combination of these two kinds of parse tree. The combined tree keeps both structural information and dependency relationships, which would be helpful in long sentence translations. In our own evaluations, we found this translation system could generally achieve better automatic evaluation scores for sentences containing 40 words or more (long sentences) comparing to the hierarchical phrase model [3], which is our baseline model.

In the rest of this paper, we first describe the method for combining CFG and dependency parse trees (Section 2). Then we describe rule acquisition (Section 3), the model (Section 4) and the decoding process (Section 5). The official evaluation and our post-evaluation are given in Section 6.

2. COMBINING CFG AND DEPENDENCY PARSE TREES
Although dependency trees provide more information on relationships between words that are far apart, most dependency tree parsers struggle to parse correctly non-content words, such as function words, idiomatic usages and some symbols which have no explicit semantic definitions. These types of words could be considered as constituent parts of sentence structure. Unless these words are correctly translated, the translation of the whole sentence will run into obstacles.

This problem could be mitigated if the input of the translation system were to combine both CFG and dependency parse trees. Actually, the method described in this paper will yield a tree in the style of a hierarchical phrase tree, in which each node represents a phrase with several nonterminal symbols. For each node, we pick out a terminal to be the headword of this node, and then use its linguistic tag to represent the type of node.

2.1 Combining dependency relations with phrase-level structures in CFG tree
The core concept of the method described in this paper tries to construct a new tree, which intuitively keeps dependency relations in large scale and also reasonable phrase-level structures extracted from CFG tree. This method could also be thought as reconstruction of CFG tree using dependency relations, because the new tree will also be a CFG tree.

We apply heuristics to do this reconstruction. For each level in given CFG tree (here level means nodes belonging to one parent node), we mark one terminal to be headword. Then we could get rough dependency relations from CFG tree using following rules.

1. Each headword depends on headword in the parent level
2. For other nodes, we do not add dependency relations for them

If the dependency relations we got from CFG tree conflicts with that in parsed dependency tree, we rearrange the tree according to the parsed dependency tree until no conflict was found. In this process, relations in one level are ignored, so phrase-level structures will not be broken (especially for some proper names).

2.2 Removing redundant non-terminals in CFG trees

We simplify the parsed CFG tree by removing redundant non-terminals, because keeping too many non-terminals in a parse tree does not improve the translation quality. Our method accomplishes this in 2 steps.

First, we remove non-terminals which have only one child node, as shown in figure 1. In this step, we avoid translating ambiguous non-terminals which represent the linguistic tag of a terminal or a subtree.

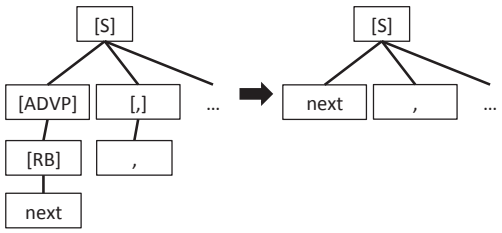

Figure 1: Removing non-terminals having only one child node

The second step is to eliminate layers containing no terminals (here a layer means nodes belong to the same parent node). The existence of such layers sometime misleads the decoding process, because the translation of a tree fragment consisting of pure non-terminals is ambiguous. We expect there to be at least one terminal in an arbitrary tree fragment to provide a clue for translation.

In the case that a layer contains only one non-terminal, we just remove this node, and directly concatenate its child nodes to its parent node. In a more complex case, a layer is composed of two or more non-terminals. In this situation, a non-terminal node is selected to be replaced by its child nodes. We perform the selection by counting internal dependencies in this layer, and find the non-terminal with a subtree having the most dependent nodes in the scope. That means the subtree should contain a semantic head for the entire layer. A illustration of this step is shown in figure 2.

Figure 2 shows a fragment with a internal layer formed from non-terminals. We count dependent words for all child nodes and mark the word with the most dependent words as the semantic head. Then we replace the non-terminal above the semantic head with its child nodes.

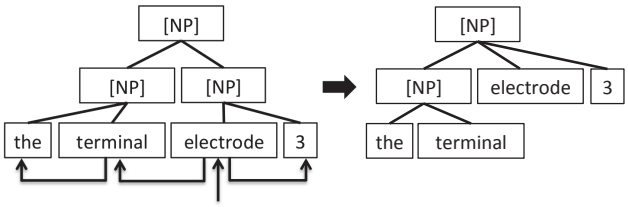

Figure 2: Eliminating a layer with multiple non-terminals

When above steps are finished, we can merge nodes in each layer together into one node to form a tree in the style of a hierarchical phrase tree, which gives a more intuitive image for the translation process.

2.3 Headword determination

The headword of a phrase can be determined using dependency relations. In contrast to some other methods [10], we allow a phrase to have only one headword. Given a token sequence s which is formed by terminals and non-terminals, covering a span of words f_i^j, we define the headword in the following way.

Definition 1. *For token sequence s, a terminal s_k is regarded as headword if s_k has the most dependent words in f_i^j.*

If two terminals have equal numbers of dependent words, we select the right-most word as the headword. A illustration of a combined tree fragment is shown in figure 3. In order to facilitate the understanding of rule acquisition in next section, we put nodes of same level into one box here. The headword for each node is marked with underscore, and the tags of non-terminal symbols are determined by the corresponding headwords of child nodes.

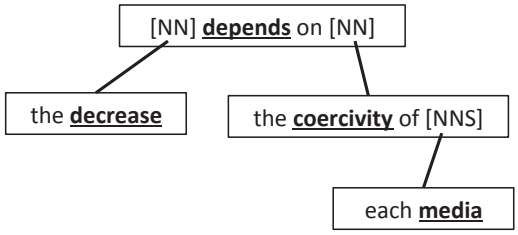

Figure 3: Illustration of combined fragment

2.4 Separating words without dependencies

Some words do not appear in the parsed dependency trees, so we place these words into separate nodes. These words are usually symbols that represent the structure of the sentence, so they should be separated from content words which represent meanings. Although it depends on what parser we use, it would normally introduce ambiguity if we were to force a dependency head on those words.

3. RULE ACQUISITION

We extract the minimal necessary translation rules from pairs of a combined source-side tree and a target-side string.

Given a combined tree and words with alignment relations A, for an arbitrary subtree t and target-side words $e_i^j = e_i, e_{i+1}, ..., e_j$, we extract this pair as an initial translation rule if the following conditions are satisfied.

1. $\exists\ f_m \in t, e_n \in e_i^j \quad s.t. \quad < f_m, e_n > \in \ A$
2. $\forall\ f_m \notin t, e_n \in e_i^j \quad , \quad < f_m, e_n > \notin \ A$
3. $\forall\ f_m \in t, e_n \notin e_i^j \quad , \quad < f_m, e_n > \notin \ A$
4. e_i^j has minimal length

If a tree fragment in the initial translation rules contains other fragments which also exist in the initial translation rules, then we mark those child fragments as sites constrained by the linguistic tag of headwords of child fragments. Corresponding sites on the target side are not constrained. A illustration of hierarchical rule extraction is shown in figure 4.

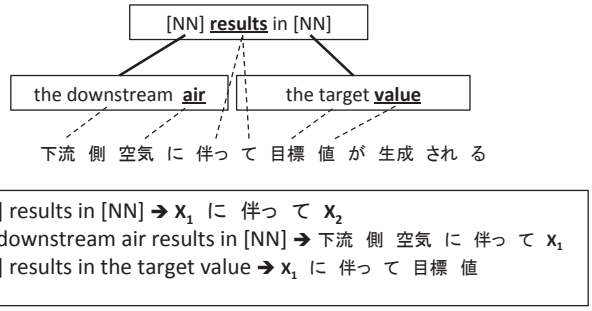

Figure 4: Hierarchical rule extraction

4. THE MODEL

As in other recent statistical machine translation models, we adopt general log-linear model:

$$\hat{e} = argmax_e \prod_i \phi_i(e)^{\lambda_i} \quad . \tag{1}$$

Here we use following features for decoding:

- Rule confidence, $P_{\mathrm{cnf}}(e|f) = \frac{\min(\mathrm{count}(e|f), 1000)}{1000}$
- Translation probabilities, $P(e|f)$ and $P(f|e)$
- Lexical translation probabilities, $P_{\mathrm{lex}}(e|f)$ and $P_{\mathrm{lex}}(f|e)$
- Language model, $P_{\mathrm{lm}}(e)$
- Rule penalty, $\exp(-1)$
- Word penalty[1], $\exp(-1)$

[1]We did not include the word penalty in the features of our translation model for the formal run.

5. DECODING

5.1 Decoder

In our decoder, we use bottom-up decoding to find the best derivations. Given a combined tree, for each node n, we traverse any tree fragment rooted at n for a maximum of 3 layers. Then we try to find matched translation rules in the rule table. If no matched rule is found, we use the Cocke–Younger–Kasami (CYK) algorithm to construct translation rules during the decoding process. As each node in the combined tree is actually a phrase, we could do this by a method that is similar to that used in the hierarchical phrase-based model [3] which involves glue rules.

We use cube pruning [4] to find the best derivation for each tree fragment, and also set a beam size for every hypothesized stack of nodes.

5.2 Optimizing for memory usage

Normally, the decoder of a statistical machine translator reads the whole rule table into memory at the beginning. This requires a large amount of memory even though most of the translation rules are not used during the translation process. In order to address this problem, we hashed all source-side strings of the translation rules, and loaded only those hashed strings into memory. As a result, our decoder consumes only 3.8GB of memory during the evaluation task.

Our decoding process is divided into two steps: rule fetching and translation. In the rule fetching step, we find all possibly usable translation rules for each node in the combined tree, and dynamically read them into memory from the on-disk rule table. In the translation step, as all translation rules have been prepared, there are no I/O operations. So we could perform these two steps in two parallel processes, which will increase the speed of decoding.

5.3 Use of a large $n$gram language model with LSH compression

To load the language model into physical memory, we used a *lossy* compression method with locality sensitive hashing (LSH) [5] [2] [1] . This LSH language model (LSHLM) is a kind of *lossy* language model which has a high compression ratio and a low probability of information loss triggered by *falsepositives*. ShefLM [6] is an alternative *lossy* language model but that model exhibits significant information loss, so the translation quality may be degraded. To mitigate the impact of falsepositives, we assume that similar $n$grams have similar probabilities. We used the LSH function to generate a group of similar $n$grams(called a "bucket"). The LSH function can map similar $n$grams to the same hash value. When a falsepositive occurs, similar $n$grams are in same bucket and, therefore, incorrect value should be similar to correct one.

In the LSH function, when $n$gram is given, we create a weight vector ($\boldsymbol{u}$) to be multiplied by random vector of LSH function which is very sparse. Weights are allocated depending on word position. For example, $n$gram = "mafuyu no aisu ha kakubetsu(真冬のアイスは格別)", $u[mafuyu] = 1$, $u[no] = 2$, $u[aisu] = 3$, $u[ha] = 4$ and $u[kakubetsu] = 5$.[2]

[2]This method is inspired by a similar method of vector space model in the field of information search

In our method, at the time of query, a given *n*gram is hashed by the LSH function and is allocated to a bucket. Probability values are found in the same way as in ShefLM. Because we use a bucket, falsepositives will occur for similar *n*grams. Therefore, the impacts of false positives are lessened and the translation score approaches that of *lossless* language models. A perspective view of LSHLM is shown in figure 5.

We used KenLM [7] as the baseline model of LSHLM to mesure the *losslessness* of this language model. KenLM is a language model of *lossless* compression. Because of its *losslessness*, model size of KenLM is greater than LSHLM. We use PROBE method to build KenLM language model binary. In PROBE method, KenLM has three arrays: fingerprint array, probability array, BOW (Backoff Weight) array. *n*grams are hashed into array index and fingerprints, probabilities, BOWs are stored into the index. Fingerprints are calculated into 64bit value, and *n*grams themselves are not stored.

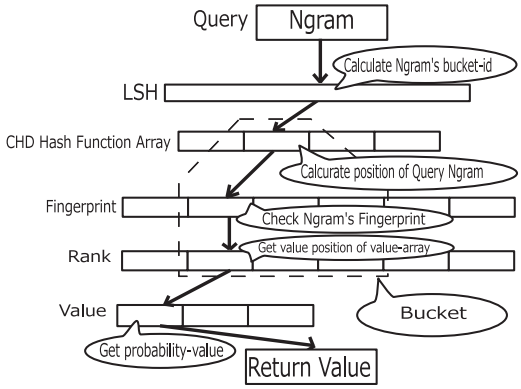

Figure 5: Perspective view of LSHLM

6. EVALUATION

6.1 Run configurations

We used a training corpus from the NTCIR-10 PatentMT English-to-Japanese task, which contains 3M bilingual sentences. We used Mecab (a part-of-speech and morphological analyzer for Japanese) to segment sentences into words in the Japanese side of the corpus.

We parsed English-side sentences with the Berkeley Parser [13] to get CFG parse trees, and then produced normal dependency parse trees with the Stanford Parser [9] using parsed CFG trees from the Berkeley Parser as input.

We generated word alignments using GIZA++ [12] on the training corpus in both directions with the "grow-diag-final-and" refinement.

For the language model, we obtained a 5-gram model on Japanese-side sentences with interpolated Kneser-Ney smoothing using the SRILM Toolkit [14].

In order to tune the weights of each feature in our translation model, we used Minimal Error Rate Training [11]. The development data contains 2000 bilingual sentences.

In the decoding phase, we used 2000 for the cube size for cube pruning, and 1000 as the beam size.

To solve the problem of the sparseness of the rule table, we simplified the linguistic tag set produced by the Berkeley Parser from 45 types to 8 types. We generally mapped noun tags like "NNS" and "NN" to a single tag "NP". Verb tags were mapped to "VP". Some rare linguistic tags like "SYM" were also mapped to "NP". Finally, we adopted 8 tags in our translation system:

NP, VP, JJ, CD, IN, S, TO, DT .

In the intrinsic subtask of the NTCIR-10 PatentMT English-to-Japanese task, we submitted three run results: TSUKU-ej-int-1, TSUKU-ej-int-2 and TSUKU-ej-int-3.

TSUKU-ej-int-1 was translated using a normal 5-gram language model, which was trained from the same bilingual corpus we used to train our translation model. While the other two results are translated using a 5-gram language model trained from a large-scale monolingual resource of about 174M sentences from Japanese patent applications in the period 1993–2000.

We used KenLM in the decoding process for TSUKU-ej-int-1 and TSUKU-ej-int-3, and the LSH compressed language model for TSUKU-ej-int-2.

6.2 Formal run results

The official automatic evaluation results are shown in Table 1. Our three runs were worse in terms of BLEU but better in terms of the NIST and RIBES metrics than the organizer baseline using a hierarchical phrase-based model.

Table 2 shows the subjective evaluation results. Our translation of TSUKU-ej-int-1 achieved a better Adequacy score than the organizer baselines.

Table 1: Official automatic evaluation results for the English-to-Japanese task

System	BLEU	NIST	RIBES
System runs			
TSUKU-ej-int-1	0.3141	8.126	0.7555
TSUKU-ej-int-2	0.319	8.1894	0.7565
TSUKU-ej-int-3	0.3176	8.1769	0.7566
Organizer baselines			
BASELINE HPBMT	0.3298	8.0837	0.7231
BASELINE PBMT	0.3361	8.1816	0.7042
Best competitors			
NTITI-1	0.4289	9.2645	0.7984
NTITI-2	0.4207	9.0779	0.7938

6.3 Post-evaluation results

The results of the formal run were not ideal because we did not include a word penalty in the features of our model due to an oversight. Table 3 shows the post-evaluation results. The scores are slightly different to the official automatic evaluation scores because we are using "mteval-v11b.pl" to do the automatic evaluation. TSUKU-ej-int-1^{submitted} is the translation result we submitted in the formal run and

Table 2: Official subjective evaluation results for the English-to-Japanese task

System	Adequacy	Acceptability
System runs		
TSUKU-ej-int-1	2.7933	0.4088
Organizer baselines		
BASELINE HPBMT	2.69	-
BASELINE PBMT	2.5333	-
Best competitors		
NTITI-1	3.8433	0.6590
NTITI-2	3.8066	0.6575

TSUKU-ej-int-1^{post} refers to the translation result of our post-evaluation.

By appending word penalty to the features of our model, we could see a promotion in BLEU score but not too varied according to other evaluation criterions.

Table 3: Post-evaluation results for the English-to-Japanese task

System	BLEU	NIST	RIBES
System runs			
TSUKU-ej-int-1^{post}	0.33	8.0174	0.7563
TSUKU-ej-int-1^{submitted}	0.3145	8.0821	0.756
baselines			
BASELINE HPBMT	0.3306	8.0849	0.7242

As we are expecting our model to achieve better translation quality in long sentence translations, we also evaluated it with a test corpus formed by 1048 sentences, which are picked out from formal run data of NTCIR8 and NTCIR10 and each sentence is longer than 40 words in English side.

Evaluation results in table 4 show our model could generally generate better translations than that of hierarchical phrase-based model. Conversely, short sentence translation is the weakness of our model due to the constrain of parsing errors.

Table 4: Post-evaluation results for the long sentence translations

System	BLEU	NIST	RIBES
System runs			
TSUKU-ej-int-1^{post}	0.3238	7.966	0.7141
Baselines			
BASELINE HPBMT	0.3185	7.9946	0.6718

7. CONCLUSION AND FUTURE WORK

In this paper, we have described the TSUKU machine translation system which implements a tree-to-string translation model that uses as input a combination of CFG and dependency parse trees. The combined trees maintain dependency relations by adjusting node positions according to their dependency heads, and defining headwords in each combined phrase-like node to provide linguistic tags. As our combined trees are transformed based on parsed CFG trees, the structure words remain in reasonable positions.

In future work we will try to build a model based on full CYK decoding, driven by these combined trees, in order to make the translation model less affected by parsing errors. We will include parsing confidence to control the decoding process dynamically.

8. REFERENCES

[1] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02. ACM.

[2] Moses Samson Charikar. Methods and apparatus for estimating similarity, 2001.

[3] David Chiang. A hierarchical phrase-based model for statistical machine translation. In ACL, 2005.

[4] David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33, 2007.

[5] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proc. of the 25th International Conference on Very Large Data Bases*, VLDB '99.

[6] David Guthrie and Mark Hepple. Storing the web in memory: Space efficient language models with constant time retrieval. In *Proc. of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010. ACL.

[7] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, 2011. ACL.

[8] K. P. Chow E. Sumita I. Goto, B. Lu and B. K. Tsou. Overview of the patent machine translation task at the ntcir-10 workshop. In *Proc. of NTCIR-10*, 2012.

[9] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *In Advances in Neural Information Processing Systems 15 NIPS*, 2003.

[10] Junhui Li, Zhaopeng Tu, Guodong Zhou, and Josef van Genabith. Head-driven hierarchical phrase-based translation. In *Proc. of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12.

[11] Franz Josef Och. Minimum error rate training in statistical machine translation, 2003.

[12] Franz Josef Och and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 2004.

[13] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *In ACL ' 06.*

[14] Andreas Stolcke. Srilm - an extensible language modeling toolkit. 2002.

[15] Deyi Xiong, Qun Liu, and Shouxun Lin. A dependency treelet string correspondence model for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07. ACL.