ABSTRACT
We describe the statistical machine translation (SMT) systems developed at Heidelberg University for the Chinese-to-English and Japanese-to-English PatentMT subtasks at the NTCIR10 workshop. The core system used in both subtasks is a combination of hierarchical phrase-based translation and discriminative training, either using our own pairwise ranking method with feature selection [19], or a MERT implementation [15, 10].

Our software for discriminative training (dtrain) is freely available as a part of the cdec research platform1. [4].

We restricted our systems to a constrained data situation where only the parallel corpus provided by the organizers was used for training both translation and language models. If we compare our system to other constrained data submissions, i.e. systems that did not use additional monolingual resources for language modeling or additional external resources such as dictionaries or post-editing, our best system ranked 3rd with regard to BLEU [16] on the Intrinsic Evaluation test set (IE) for the Chinese-to-English translation subtask and 2nd for the Japanese-to-English translation subtask also on this subtask’s IE test set.

The general idea of our approach is to exploit the janiform nature of patent data: on the one hand the repetitive, formulaic side of patents eases translation; on the other hand long sentences and an unusual jargon complicate translation. Our key idea for exploiting the repetitive nature of patents is to approach it as a multi-task learning problem. Starting from a partition of parallel training data into shards, viewed as tasks, we apply $\ell_1/\ell_2$ regularization [19] in discriminative training to achieve small sets of features that are useful across tasks, thus yielding a small model that countersteers overfitting. This approach is used in our Japanese-to-English system.

For the Chinese-to-English subtask we focus on handling the complexity of patent jargon that emphasizes word ordering differences between Chinese and English in long sentences. Since long-distance reordering phenomena cannot be modeled well if lexical phrases are short and if non-lexical items in hierarchical phrases do not carry linguistic information, we incorporate soft syntactic features into our models to prevent reordering errors [13, 20, 1].

1https://github.com/redpony/cdec/tree/master/training/dtrain
2. STATISTICAL MACHINE TRANSLATION FRAMEWORK

2.1 Hierarchical Phrase-Based Translation

Our SMT framework is hierarchical phrase-based translation [2] using the cdec decoder. Translation rules are extracted from word aligned parallel sentences and can be seen as productions of a synchronous CFG. Examples are rules like (1)-(3) shown for Japanese-to-English translation in Figure 1. SCFG grammars were induced from symmetrized word alignments using the method described by [11]. The grammar rules necessary to translate each individual sentence are extracted into separate files (so-called per-sentence grammars).

2.2 Discriminative Training with Large Feature Sets

Discriminative training in SMT has the advantage of being able to handle models with arbitrary types and numbers of features, including dense or sparse lexicalized features, defined locally or globally, as well as overlapping features.

Our system makes use of three types of features: Firstly, we incorporate 12 dense features from the default cdec implementation into discriminative training. Furthermore, we use sparse lexicalized features, that are defined locally on SCFG rules. We use three rule templates:

Rule identifiers: These features identify each rule by a unique identifier. Such features correspond to the relative frequencies of rewrites rules used in standard models.

Rule n-grams: These features identify n-grams in source and target side of a rule, which allows the model to prefer rules that include certain n-grams. We use bigrams on source- and target-side.

Rule shape: For these features, we defined patterns, which identify the location of sequences of terminal symbols in relation to non-terminal symbols, on both the source and target side of each rule used. These features abstract away from the lexical items at terminal nodes. For example, rule (1) in figure 1 maps to the pattern (NT, term*, NT) on the source side, and (NT, term*, NT, term*) on the target side. Rule (2) maps to a different template, that of (term*, NT, term*) on source and target sides.

Finally, we define non-local features that encode soft syntactic constraints. We tried different variations of this idea, following [13], [20] and [1]. We found source-side soft syntactic constraints that reward rules with source spans matching

\[ X \rightarrow X_1 \text{ 要件の } X_2 \mid X_3 \text{ of } X_1 \text{ requirements} \]

\[ X \rightarrow \text{ このとき。 } X_1 \text{ is } \text{ this time, the } X_1 \text{ is} \]

\[ X \rightarrow \text{ テキスト メモリ } 41 \text{ X1 X1 in the text memory 41} \]

2.3 Feature Sharing via \( \ell_1/\ell_2 \) Regularization

The goal of our method for feature sharing is to increase efficiency of learning and to provide a measure against overfitting. The key idea is to view data partitions (shards) as tasks and to apply methods for joint feature selection from multi-task learning to achieve small sets of features that are useful across all tasks or shards. Our algorithm represents weights in a \( Z \)-by-\( D \) matrix

\[ W = [w_{z1}, \ldots, w_{zD}]^T \]

of stacked \( D \)-dimensional weight vectors across \( Z \) shards. We compute the \( \ell_2 \) norm of the weights in each feature column, sort features by this value, and keep \( K \) features in the model. This feature selection procedure is carried out after each epoch. Reduced weight vectors are mixed and the resulting average vector of dimensionality \( K \) is then used to initialize another epoch of parallel training on each shard.

This algorithm can be seen as an instance of \( \ell_1/\ell_2 \) regularization as follows: Let \( w_d \) be the \( d \)th column vector of \( W \), representing the weights for the \( d \)th feature across tasks/shards. \( \ell_1/\ell_2 \) regularization penalizes weights \( W \) by the weighted \( \ell_1/\ell_2 \) norm

\[ \lambda|W|_{1,2} = \lambda \sum_{d=1}^{D} |w_d|_2. \]

Each \( \ell_2 \) norm of a weight column represents the relevance of the corresponding feature across tasks/shards. The \( \ell_1 \) sum of the \( \ell_2 \) norms enforces a selection of features based on these norms.

For a more formal description of the algorithm and experimental comparison with related work on distributed stochastic learning see [19].

3. CHINESE-TO-ENGLISH PATENT MT

3.1 System Setup

The training data used in our experiments consists of one million sentence pairs provided for the NTCIR10 PatentMT...
This system adds 5 features from the Stanford Segmenter toolkit and editing. Hence we call our approach “constrained data patent editing aspect compared to standard patent collections.

This considerably reduces the redundancy of NTCIR10) each consist of 2,000 sentences taken from the development parses were obtained for development and test set (see [7]). This considerably reduces the redundancy aspect compared to standard patent collections.

We did not use any other resources, neither additional monolingual data nor external dictionaries nor human post-editing. Hence we call our approach “constrained data patent MT”.

The Chinese data was word segmented and tokenized using the Stanford Segmenter toolkit^3.

For the experiments using soft syntactic constraints, syntactical parses were obtained for development and test set using the Stanford Parser for Chinese^4.

Tokenization of the English side of the data, detokenization of the final output and recasing were carried out using unmodified Moses tools. The recaser was trained on the truecased side of the provided parallel data.

### 3.2 Soft Syntactic Constraints

We experimented with three different kinds of soft syntactic constraints. The setup that worked best in our experiments is a re-implementation of the ideas presented in [13]. These source-side soft syntactic constraints enable the system to learn whether rule non-terminal spans should preferably conform with all or just certain syntactic constituents (noun phrases, verb phrases etc.) in the source sentence, or whether they can cross them. These preferences are learnt on the development corpus and encoded as feature weights.

Weights can be either tuned independently or be tied to each other. In the first case separate weights are tuned for two features, one indicating matching, the other indicating crossing. In the second case, one feature is used where the model score is incremented by the weight of the matching feature and decremented by the weight of the crossing feature.

Experimental results for soft syntactic constraints on the 10 most common phrase types with independently tuned weights (short: XP2) are shown in table 1. Using the following 10 phrase types

\[
ADJP, ADVP, CP, DN, IP, LCP, NP, PP,QP, VP
\]

and indicators for matching (\(=\)) and crossing (\(+\)), 20 features are added to the standard 12 dense features. Table 1 also shows the results for a system with soft syntactic constraints on simple clauses (IP) and verb phrases (VP) with independently tuned weights and noun phrases with tied weights (short: IP2 VP2 NP2). This system adds 5 features to the standard 12 dense features. Weights of all systems using this type of features were tuned using MERT. The system with XP2 soft syntactic constraints scores 0.78 BLEU points (test: 0.71) better than the baseline system when translating with standard cube pruning [2] pop limit and span size limit. The IP2 VP2 NP2 soft syntactic constraints show an improvement of 0.51 BLEU points (test: 0.4) over the baseline under standard settings. The improvements in BLEU for both systems is significant with a p-value < 0.01 (using approximate randomization for assessing statistical significance of result differences [14, 17]).

Further experiments were done using “parsematch” features as presented by [20] and “sparse syntactic features” as described by [1]. In the first case, a hypothesis score is computed, which reflects how closely the decoder’s parse trees for source and target sentence resemble linguistic parse trees. This score is computed offline for each rule, by looking at the source and target-side parse trees of the training sentences the rule was extracted from. A hierarchical phrase is defined to be linguistically consistent if

- it corresponds to the “yield of a node” in the corresponding parse tree, meaning that it represents a syntactical constituent according to the syntactic parse of the sentence and
- the subphrases filling its gaps are also linguistically consistent.

Many phrases in the grammar do not exactly match the yield of a node in the parse tree, but come close to it by just lacking a few words. To differentiate between phrases lacking only one word and phrases lacking many words, a quantity is introduced which records the distance to the closest syntactic label. This quantity corresponds to the number of words that have to be added or deleted so that the resulting phrase corresponds to a syntactic constituent. For more details, see [20]. The “parsematch” system adds 1 feature to the standard 12 dense features. The weights of this system were also trained using MERT.

“Sparse syntactic features” as described in [1] encode

- the constituent spanning the rule’s source side in the syntax tree (if any),
- constituents spanning any variable in the rule,
- the rule’s target side surface form.

As implied by the naming, these features are sparse and produce large feature sets due to the inclusion of the target side surface forms. Similar to [1], we reduce the number of features by a count cutoff on the training set. For this purpose, an additional sample of 20,000 sentences from the training set was parsed. The sample sentences were translated with cdec and all features triggered more than five times were written into a whitelist. This whitelist yields 832,952 features. In the tuning and testing step, all sparse syntactic features apart from the ones on the whitelist were ignored.

<table>
<thead>
<tr>
<th></th>
<th>baseline</th>
<th>parsematch</th>
<th>sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev</td>
<td>34.06 (36.37)</td>
<td>34.07 (36.40)</td>
<td>34.56 (36.81)</td>
</tr>
<tr>
<td>test</td>
<td>32.35 (34.03)</td>
<td>32.44 (34.17)</td>
<td>31.79 (33.50)</td>
</tr>
</tbody>
</table>

Table 2: Parsematch and sparse syntax features evaluation (% BLEU): ss indicates the span size and pl indicates the pop limit used by the decoder. The results are given for the recased and detokenized output, the result for the lowercased and tokenized output is given in brackets. Statistically significant result differences to the baseline are indicated in bold face.
configuration was done
stable in this respect with no need of repetition of experi-
language model of this system, we experimented with a range
labeled systems to the three million parallel sentences provided by the or-
experiments, limiting our translation and language models to the three million parallel sentences provided by the or-
A manual evaluation done on the development data before
For the English side of the training data, we applied regular expressions to align ASCII-strings between
Due to their technical nature, the patent texts contained a large number of non-Japanese expressions, such as abbrevia-
Tuning was done using the pairwise-ranking perceptron al-
Table 2 shows an experimental comparison of “parsematch” and “sparse syntactic features” against the baseline. Whereas
4.2 Preprocessing
The MeCab toolkit was used for segmentation of Japanese text. We applied modifications to improve over-segmentation of ASCII-strings and under-segmentation of katakana terms: Due to their technical nature, the patent texts contained a large number of non-Japanese expressions, such as abbreviations, patent ids or English terms. We noticed that, since the non-Japanese-characters in the provided data were in Full-width Latin, MeCab tended to heavily over-segment them at each character position, leading to faulty alignments. Therefore, we converted all Fullwidth Latin characters to ASCII format before running the segmenter. Even with this format, tokenization of ASCII strings occurring in English and Japanese sentences was sometimes inconsistent. To avoid this problem for training, we followed [12]’s approach for Chinese segmentation and tried to apply one consistent tokenization to ASCII-strings in the Japanese training data and
Table 3: Experimental results at NTCIR10 Chinese-
to-English subtask (%BLEU on Intrinsic Evaluation test set): HDU-1 adds 20 source-side soft sintactic constraints to a discriminatively trained hierarch-
the dev set, this evaluation used a large number of non-Japanese expressions, such as abbreviations, patent ids or English terms. For the Japanese test data, we used regular expressions to align ASCII-strings between
The baseline systems BASELINE-1 and BASELINE-2 denote the Moses hierarchical phrase-based MT system and the Moses phrase-based MT system, respectively. System ONLINE-1 denotes the Google online translation system. Our systems score better than all baselines.
4.3 Experimental Results at NTCIR10
We restricted our systems to a constrained data situation where only the parallel corpus provided by the organizers was used for training both translation and language models. In comparison to other constrained data submissions, our system ranked 3rd with respect to BLEU for the Chinese-
Table 2 shows BLEU scores for our system compared to three baselines. Our system, called HDU-1, adds 20 source-
we always used the tokenization which had been seen most
Japanese sentences was sometimes inconsistent. To avoid this problem for training, we followed [12]’s approach for Chinese segmentation and tried to apply one consistent tokenization to ASCII-strings in the Japanese training data and

to their English counterparts. For the parallel training data, we used regular expressions to align ASCII-strings between Japanese and English and then replaced strings in Japanese with their English counterparts. For the Japanese test data, we always used the tokenization which had been seen most often in the training data.
We follow [5] who applied a modified version of the compound splitter described in [9] to katakana terms, which are often a transliteration of English compound words. As these are usually not split by MeCab, they can cause a large number of out-of-vocabulary words. On the devtest set, this reduced the number of OOV terms from 98 to 34. Table 4 shows the effects of modifying the segmentation.
For the English side of the training data, we applied a modified version of the tokenizer included in the Moses scripts.
ments. When using dtrain in the non-setup (tuning on the single dev sets separately), we averaged the weight vectors over 15 epochs for better generalization. The multi-task setup (as described in section 2.3) was run for 10 (development on dev1-3) and 5 (final run with dev1-3 and devtest, split into two parts to match the size of dev1-3) epochs.

system | BLEU  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HDU-2</td>
<td>35.39</td>
</tr>
<tr>
<td>HDU-1</td>
<td>35.21</td>
</tr>
<tr>
<td>ONLINE-1</td>
<td>33.88</td>
</tr>
<tr>
<td>BASELINE-1</td>
<td>32.52</td>
</tr>
<tr>
<td>BASELINE-2</td>
<td>31.34</td>
</tr>
</tbody>
</table>

4.2 Preprocessing
The MeCab toolkit was used for segmentation of Japanese text. We applied modifications to improve over-segmentation of ASCII-strings and under-segmentation of katakana terms: Due to their technical nature, the patent texts contained a large number of non-Japanese expressions, such as abbreviations, patent ids or English terms. We noticed that, since the non-Japanese-characters in the provided data were in Full-width Latin, MeCab tended to heavily over-segment them at each character position, leading to faulty alignments. Therefore, we converted all Fullwidth Latin characters to ASCII format before running the segmenter. Even with this format, tokenization of ASCII strings occurring in English and Japanese sentences was sometimes inconsistent. To avoid this problem for training, we followed [12]’s approach for Chinese segmentation and tried to apply one consistent tokenization to ASCII-strings in the Japanese training data and
The MeCab toolkit was used for segmentation of Japanese text. We applied modifications to improve over-segmentation of ASCII-strings and under-segmentation of katakana terms: Due to their technical nature, the patent texts contained a large number of non-Japanese expressions, such as abbreviations, patent ids or English terms. We noticed that, since the non-Japanese-characters in the provided data were in Full-width Latin, MeCab tended to heavily over-segment them at each character position, leading to faulty alignments. Therefore, we converted all Fullwidth Latin characters to ASCII format before running the segmenter. Even with this format, tokenization of ASCII strings occurring in English and Japanese sentences was sometimes inconsistent. To avoid this problem for training, we followed [12]’s approach for Chinese segmentation and tried to apply one consistent tokenization to ASCII-strings in the Japanese training data and
We follow [5] who applied a modified version of the compound splitter described in [9] to katakana terms, which are often a transliteration of English compound words. As these are usually not split by MeCab, they can cause a large number of out-of-vocabulary words. On the devtest set, this reduced the number of OOV terms from 98 to 34. Table 4 shows the effects of modifying the segmentation.
For the English side of the training data, we applied a modified version of the tokenizer included in the Moses scripts.
rule features [...] describing the present invention, it will be appreciated by those skilled in the art, there may also be numerous other combinations and permutations of the present invention.

soft syntax [...] describing the present invention, but one of ordinary skill in the art will recognize that numerous other combinations and permutations of the present invention are also possible.

reference [...] describing the subject invention, but one of ordinary skill in the art may recognize that many further combinations and permutations of the invention are possible.

Figure 2: Example for Chinese-to-English subtask comparing a system utilizing local rule features (HDU-2) with a system that uses non-local soft syntactic constraints (HDU-1), taken from the chronological evaluation test set. System HDU-2 fails to produce a coherent sentence, outputting a list of subordinate clauses. Whereas system HDU-1 produces a fluid sentence closely resembling the reference, which also results in a higher n-gram precision compared to the output of system HDU-2.

Table 4: Example illustrating the effect of splitting katakana transliterations of English compounds.

<table>
<thead>
<tr>
<th>IPC class</th>
<th>dev1</th>
<th>dev2</th>
<th>dev3</th>
<th>devtest</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.53</td>
<td>3.45</td>
<td>2.67</td>
<td>1.0</td>
</tr>
<tr>
<td>B</td>
<td>8.09</td>
<td>11.33</td>
<td>7.23</td>
<td>11.95</td>
</tr>
<tr>
<td>C</td>
<td>1.53</td>
<td>0.97</td>
<td>1.22</td>
<td>1.0</td>
</tr>
<tr>
<td>D</td>
<td>0.11</td>
<td>0.22</td>
<td>0.33</td>
<td>1.2</td>
</tr>
<tr>
<td>E</td>
<td>0.66</td>
<td>0.11</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>F</td>
<td>5.9</td>
<td>7.66</td>
<td>5.45</td>
<td>11.95</td>
</tr>
<tr>
<td>G</td>
<td>47.65</td>
<td>43.94</td>
<td>46.27</td>
<td>30.35</td>
</tr>
<tr>
<td>H</td>
<td>34.54</td>
<td>39.23</td>
<td>35.82</td>
<td>42.45</td>
</tr>
</tbody>
</table>

Table 5: IPC class distribution of segments in dev sets given in percentages. Each segment was assigned to the main classification of its originating patent.

<table>
<thead>
<tr>
<th>IPC class</th>
<th>dev set</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>dev1 dev2 dev3 devtest</td>
</tr>
<tr>
<td>B</td>
<td>1.53 3.45 2.67 1.0</td>
</tr>
<tr>
<td>C</td>
<td>8.09 11.33 7.23 11.95</td>
</tr>
<tr>
<td>D</td>
<td>1.53 0.97 1.22 1.0</td>
</tr>
<tr>
<td>E</td>
<td>0.11 0.22 0.33 1.2</td>
</tr>
<tr>
<td>F</td>
<td>0.66 0.11 1.0 0.1</td>
</tr>
<tr>
<td>G</td>
<td>5.9  7.66 5.45 11.95</td>
</tr>
<tr>
<td>H</td>
<td>47.65</td>
</tr>
<tr>
<td>I</td>
<td>34.54</td>
</tr>
</tbody>
</table>

The choice of the optimizer does not seem to make a difference for tuning the dense feature weights, both methods yield about the same score on devtest. Adding the set of rule features to dtrain tuning results in a gain of 1 BLEU point when using dev1 or dev3. Tuning on dev1 produced a model of 12 dense plus 249,756 additional features – MERT is not capable of learning models of this size. Using MERT, all dev sets behave similar. Larger models are more prone to overfitting and thus more sensitive to domain shifts. We observe this when tuning with on each dev set separately: The gain using dev2 and the extended feature set is about 0.5 BLEU points below the other sets. This may be an effect of differing IPC class distributions between dev1-3 and devtest, for a detailed listing see table 5. [21] showed that there are considerable differences between these classes and that machine translation models tuned on the same class are always preferable over models that were built using mixed data (in terms of IPC class) when translating data of a specific IPC class. We assume that this also extends to IPC class distributions within models. We counter this effect with combination of all available development sets for tuning. This leads to further, but small improvements on devtest. We reached about the same result using our multi-task tuning method, taking each dev set as a separate task. This could be done in considerably less time as all the separate tasks (dev sets) could be tuned in parallel with minimal overhead, and a weight vector dimensionality of $K = 100,000$ allowed us to start each epoch with a reasonably sized model.

4.3 Tuning

Development of tuning and pre-processing was done separately, experiments reported in table 6 are without the pre-processing described in the previous subsection, except the segmentation of Japanese and tokenization of English. Tuning was carried out using a cube pruning pop limit of 200 and a maximum non-terminal span size of 15, as we found that higher settings were prohibitively slow for tuning and only lead marginally better results. For decoding of the test set we used a pop limit of 500 and a maximal span size of 100 to consider as much as possible of the search space of the decoder.

Results of the development phase are depicted in table 6. The choice of the optimizer does not seem to make a difference for tuning the dense feature weights, both methods yield about the same score on devtest. Adding the set of rule features to dtrain tuning results in a gain of 1 BLEU point when using dev1 or dev3. Tuning on dev1 produced a model of 12 dense plus 249,756 additional features – MERT is not capable of learning models of this size. Using MERT, all dev sets behave similar. Larger models are more prone to overfitting and thus more sensitive to domain shifts. We observe this when tuning with on each dev set separately: The gain using dev2 and the extended feature set is about 0.5 BLEU points below the other sets. This may be an effect of differing IPC class distributions between dev1-3 and devtest, for a detailed listing see table 5. [21] showed that there are considerable differences between these classes and that machine translation models tuned on the same class are always preferable over models that were built using mixed data (in terms of IPC class) when translating data of a specific IPC class. We assume that this also extends to IPC class distributions within models. We counter this effect with combination of all available development sets for tuning. This leads to further, but small improvements on devtest. We reached about the same result using our multi-task tuning method, taking each dev set as a separate task. This could be done in considerably less time as all the separate tasks (dev sets) could be tuned in parallel with minimal overhead, and a weight vector dimensionality of $K = 100,000$ allowed us to start each epoch with a reasonably sized model.

4.4 Experimental Results at NTCIR10

For translation of the final test sets we combined the pre-processing and tuning development efforts in a single system, used for all test sets: consistent ASCII-tokenization, compound-splitting with dtrain multi-task tuning. We added the former devtest set (split into two parts) for tuning. This system is HDU-1 in table 7. HDU-2 is the identical system, but tuning stopped early after 3 epochs. Table 7 gives results for the intrinsic evaluation (IE test set). Both of our systems are nearly indistinguishable and ranked 2nd and 3rd
We presented the SMT systems used for Chinese-to-English and Japanese-to-English PatentMT at NTCIR10 by Heidelberg University. The core system is a hierarchical phrase-based translation with discriminative training for large evaluation test sets. The baseline is the averaged score of several MERT runs using the standard 12 features. Results in bold font are significant improvements over baseline with a p-value < 0.01.

### Table 6: Development tuning results (without any pre-processing besides tokenization/segmentation).
The baseline is the averaged score of several MERT runs using the standard 12 features. Results in bold font are significant improvements over baseline with a p-value < 0.01.

<table>
<thead>
<tr>
<th>system</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDU-2</td>
<td>32.07</td>
</tr>
<tr>
<td>HDU-1</td>
<td>31.92</td>
</tr>
<tr>
<td>BASELINE-2</td>
<td>28.86</td>
</tr>
<tr>
<td>BASELINE-1</td>
<td>28.56</td>
</tr>
<tr>
<td>ONLINE-1</td>
<td>24.24</td>
</tr>
</tbody>
</table>

### Table 7: Experimental results at NTCIR10 Japanese-to-English subtask (%BLEU on Intrinsic Evaluation test set): HDU-1 and HDU-2 are an identical system, HDU-2 stopped early in the tuning phase. Our system improves over both baselines (for a description see table 3). Compared to all systems using a constrained data setup, our system ranks 2nd (HDU-2).

5. DISCUSSION

We presented the SMT systems used for Chinese-to-English and Japanese-to-English PatentMT at NTCIR10 by Heidelberg University. The core system is a hierarchical phrase-based SMT system, using discriminative training for large feature sets under $\ell_1/\ell_2$ regularization [19]. The key idea of our approach was to address the janiform nature of patents through feature sharing in a multi-task learning setup (used in Japanese-to-English subtask), and by countersteering complex word order differences by syntactic features (used in Chinese-to-English subtask). Our systems were restricted to a constrained data setup where only the provided bilingual data were used for training system modules. A comparison in this constrained data setup ranks our systems 3rd with respect to BLEU for Chinese-to-English and 2nd for Japanese-to-English PatentMT on the test sets for intrinsic evaluation.

Acknowledgments

The work presented in this paper was supported in part by DFG grant “Cross-language Learning-to-Rank for Patent Retrieval”.

6. REFERENCES


