Efficient Phrase-table Representation for Machine Translation with Applications to Online MT and Speech Translation

Richard Zens and Hermann Ney

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6 – Computer Science Department
RWTH Aachen University, D-52056 Aachen, Germany
{zens,ney}@cs.rwth-aachen.de

Abstract

In phrase-based statistical machine translation, the phrase-table requires a large amount of memory. We will present an efficient representation with two key properties: *on-demand loading* and a *prefix tree* structure for the source phrases.

We will show that this representation scales well to large data tasks and that we are able to store hundreds of millions of phrase pairs in the phrase-table. For the large Chinese–English NIST task, the memory requirements of the phrase-table are reduced to less than 20 MB using the new representation with no loss in translation quality and speed. Additionally, the new representation is not limited to a specific test set, which is important for online or real-time machine translation.

One problem in speech translation is the matching of phrases in the input word graph and the phrase-table. We will describe a novel algorithm that effectively solves this combinatorial problem exploiting the prefix tree data structure of the phrase-table. This algorithm enables the use of significantly larger input word graphs in a more efficient way resulting in improved translation quality.

1 Introduction

In phrase-based statistical machine translation, a huge number of source and target phrase pairs is memorized in the so-called phrase-table. For medium sized tasks and phrase lengths, these phrase-tables already require several GBs of memory or even do not fit at all. If the source text, which is to be translated, is known in advance, a common trick is to filter the phrase-table and keep a phrase pair only if the source phrase occurs in the text. This filtering is a time-consuming task, as we have to go over the whole phrase-table. Furthermore, we have to repeat this filtering step whenever we want to translate a new source text.

To address these problems, we will use an efficient representation of the phrase-table with two key properties: *on-demand loading* and a *prefix tree* structure for the source phrases. The prefix tree structure exploits the redundancy among the source phrases. Using on-demand loading, we will load only a small fraction of the overall phrase-table into memory. The majority will remain on disk.

The on-demand loading is employed on a per sentence basis, i.e. we load only the phrase pairs that are required for one sentence into memory. Therefore, the memory requirements are low, e.g. less than 20 MB for the Chin.-Eng. NIST task. Another advantage of the on-demand loading is that we are able to translate new source sentences without filtering.

A potential problem is that this on-demand loading might be too slow. To overcome this, we use a binary format which is a memory map of the internal representation used during decoding. Additionally, we load coherent chunks of the tree structure instead of individual phrases, i.e. we have only few disk access operations. In our experiments, the on-demand loading is *not* slower than the traditional approach.

As pointed out in (Mathias and Byrne, 2006), one problem in speech translation is that we have to match the phrases of our phrase-table against a word graph representing the alternative ASR tran-

Proceedings of NAACL HT 2007, pages 492–499,
Rochester, NY, April 2007.

scriptions. We will present a phrase matching algorithm that effectively solves this combinatorial problem exploiting the prefix tree data structure of the phrase-table. This algorithm enables the use of significantly larger input word graphs in a more efficient way resulting in improved translation quality.

The remaining part is structured as follows: we will first discuss related work in Sec. 2. Then, in Sec. 3, we will describe the phrase-table representation. Afterwards, we will present applications in speech translation and online MT in Sec. 4 and 5, respectively. Experimental results will be presented in Sec. 6 followed by the conclusions in Sec. 7.

2 Related Work

(Callison-Burch et al., 2005) and (Zhang and Vogel, 2005) presented data structures for a compact representation of the word-aligned bilingual data, such that on-the-fly extraction of long phrases is possible. The motivation in (Callison-Burch et al., 2005) is that there are some long source phrases in the test data that also occur in the training data. However, the more interesting question is if these long phrases really help to improve the translation quality. We have investigated this and our results are in line with (Koehn et al., 2003) showing that the translation quality does *not* improve if we utilize phrases beyond a certain length. Furthermore, the suffix array data structure of (Callison-Burch et al., 2005) requires a fair amount of memory, about 2 GB in their example, whereas our implementation will use only a tiny amount of memory, e.g. less than 20 MB for the large Chinese-English NIST task.

3 Efficient Phrase-table Representation

In this section, we will describe the proposed representation of the phrase-table. A prefix tree, also called trie, is an ordered tree data structure used to store an associative array where the keys are symbol sequences. In the case of phrase-based MT, the keys are source phrases, i.e. sequences of source words and the associated values are the possible translations of these source phrases. In a prefix tree, all descendants of any node have a common prefix, namely the source phrase associated with that node. The root node is associated with the empty phrase.

The prefix tree data structure is quite common in automatic speech translation. There, the lexicon, i.e. the mapping of phoneme sequences to words, is usually organized as a prefix tree (Ney et al., 1992).

We convert the list of source phrases into a prefix tree and, thus, exploit that many of them share the same prefix. This is illustrated in Fig. 1 (left). Within each node of the tree, we store a sorted array of possible successor words along with pointers to the corresponding successor nodes. Additionally, we store a pointer to the possible translations.

One property of the tree structure is that we can efficiently access the successor words of a given prefix. This will be a key point to achieve an efficient phrase matching algorithm in Sec. 4. When looking for a specific successor word, we perform a binary search in the sorted array. Alternatively, we could use hashing to speed up this lookup. We have chosen an array representation as this can be read very fast from disk. Additionally, with the exception of the root node, the branching factor of the tree is small, i.e. the potential benefit from hashing is limited. At the root node, however, the branching factor is close to the vocabulary size of the source language, which can be large. As we store the words internally as integers and virtually all words occur as the first word of some phrase, we can use the integers directly as the position in the array of the root node. Hence, the search for the successors at the root node is a simple table lookup with direct access, i.e. in $\mathcal{O}(1)$.

If not filtered for a specific test set, the phrase-table becomes huge even for medium-sized tasks. Therefore, we store the tree structure on disk and load only the required parts into memory on-demand. This is illustrated in Fig. 1 (right). Here, we show the matching phrases for the source sentence 'c a a c', where the matching phrases are set in **bold** and the phrases that are loaded into memory are set in *italics*. The dashed part of the tree structure is not loaded into memory. Note that some nodes of the tree are loaded even if there is no matching phrase in that node. These are required to actually verify that there is no matching phrase. An example is the 'bc' node in the lower right part of the figure. This node is loaded to check if the phrase 'c a a' occurs in the phrase-table. The translations, however, are loaded only for matching source phrases.

In the following sections, we will describe applications of this phrase-table representation for speech translation and online MT.

4 Speech Translation

In speech translation, the input to the MT system is not a sentence, but a word graph representing alter-

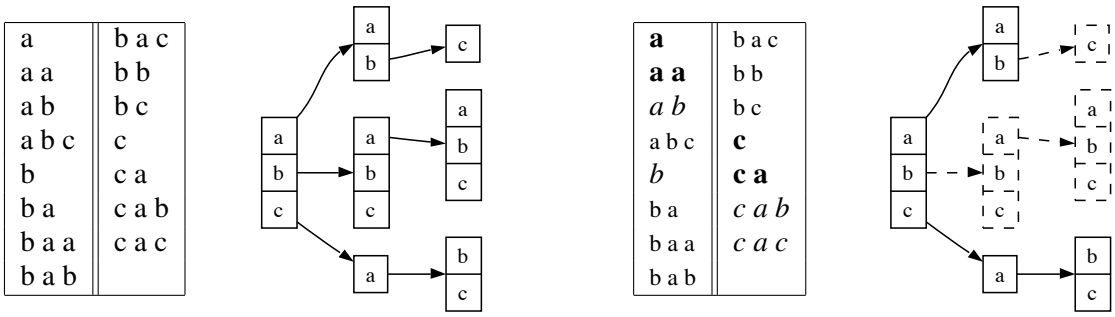

Figure 1: Illustration of the prefix tree. Left: list of source phrases and the corresponding prefix tree. Right: list of matching source phrases for sentence 'c a a c' (**bold** phrases match, phrases in *italics* are loaded in memory) and the corresponding partially loaded prefix tree (the dashed part is not in memory).

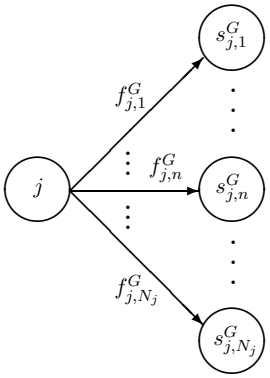

Figure 2: Illustration for graph G: node j with successor nodes $s^G_{j,1}, ..., s^G_{j,n}..., s^G_{j,N_j}$ and corresponding edge labels $f^G_{j,1}, ..., f^G_{j,n}, ..., f^G_{j,N_j}$.

native ASR transcriptions. As pointed out in (Mathias and Byrne, 2006), one problem in speech translation is that we have to match the phrases of our phrase-table against the input word graph. This results in a combinatorial problem as the number of phrases in a word graph increases exponentially with the phrase length.

4.1 Problem Definition

In this section, we will introduce the notation and state the problem of matching source phrases of an input graph G and the phrase-table, represented as prefix tree T. The input graph G has nodes $1, ..., j, ..., J$. The outgoing edges of a graph node j are numbered with $1, ..., n, ..., N_j$, i.e. an edge in the input graph is identified by a pair (j, n). The source word labeling the n^{th} outgoing edge of graph node j is denoted as $f^G_{j,n}$ and the successor node of this edge is denoted as $s^G_{j,n} \in \{1, ..., J\}$. This notation is illustrated in Fig. 2.

We use a similar notation for the prefix tree T with nodes $1, ..., k, ..., K$. The outgoing edges of a tree node k are numbered with $1, ..., m, ..., M_k$, i.e. an edge in the prefix tree is identified by a pair (k, m). The source word labeling the m^{th} outgoing edge of tree node k is denoted as $f^T_{k,m}$ and the successor node of this edge is denoted as $s^T_{k,m} \in \{1, ..., K\}$. Due to the tree structure, the successor nodes of a tree node k are all distinct:

$$s^T_{k,m} = s^T_{k,m'} \quad \Leftrightarrow \quad m = m' \tag{1}$$

Let k_0 denote the root node of the prefix tree and let $\tilde{f}_k$ denote the prefix that leads to tree node k. Furthermore, we define $E(k)$ as the set of possible translations of the source phrase $\tilde{f}_k$. These are the entries of the phrase-table, i.e.

$$E(k) = \left\{ \tilde{e} \,\middle|\, p(\tilde{e}|\tilde{f}_k) > 0 \right\} \tag{2}$$

We will need similar symbols for the input graph. Therefore, we define $F(j', j)$ as the set of source phrases of all paths from graph node j' to node j, or formally:

$$F(j', j) = \left\{ \tilde{f} \,\middle|\, \exists (j_i, n_i)_{i=1}^{I} : \tilde{f} = f^G_{j_1,n_1}, ..., f^G_{j_I,n_I} \right.$$
$$\left. \wedge\, j_1 = j' \wedge \textstyle\bigwedge_{i=1}^{I-1} s^G_{j_i,n_i} = j_{i+1} \wedge s_{j_I,n_I} = j \right\}$$

Here, the conditions ensure that the edge sequence $(j_i, n_i)_{i=1}^{I}$ is a proper path from node j' to node j in the input graph and that the corresponding source phrase is $\tilde{f} = f^G_{j_1,n_1}, ..., f^G_{j_I,n_I}$. This definition can be expressed in a recursive way; the idea is to extend the phrases of the predecessor nodes by one word:

$$F(j', j) = \bigcup_{(j'',n): s^G_{j'',n} = j} \left\{ \tilde{f} f^G_{j'',n} \,\middle|\, \tilde{f} \in F(j', j'') \right\} \tag{3}$$

Here, the set is expressed as a union over all inbound edges (j'', n) of node j. We concatenate each source phrase $\tilde{f}$ that ends at the start node of such an edge, i.e. $\tilde{f} \in F(j', j'')$, with the corresponding edge label $f^G_{j'',n}$. Additionally, we define $E(j', j)$ as the set of possible translations of all paths from graph node j' to graph node j, or formally:

$$E(j', j) = \left\{ \tilde{e} \;\middle|\; \exists \tilde{f} \in F(j', j) : p(\tilde{e}|\tilde{f}) > 0 \right\} \quad (4)$$

$$= \bigcup_{k:\tilde{f}_k \in F(j',j)} E(k) \quad (5)$$

$$= \bigcup_{(j'',n):s^G_{j'',n}=j} \; \bigcup_{\substack{k:\tilde{f}_k \in F(j',j'') \\ m:f^G_{j'',n}=f^T_{k,m}}} E(s^T_{k,m}) \quad (6)$$

Here, the definition was first rewritten using Eq. 2 and then using Eq. 3. Again, the set is expressed recursively as a union over the inbound edges. For each inbound edge (j'', n), the inner union verifies that there exists a corresponding edge (k, m) in the prefix tree with the same label, i.e. $f^G_{j'',n} = f^T_{k,m}$.

Our goal is to find all non-empty sets of translation options $E(j', j)$. The naive approach would be to enumerate all paths in the input graph from node j' to node j, then lookup the corresponding source phrase in the phrase-table and add the translations, if there are any, to the set of translation options $E(j', j)$. This solution has some obvious weaknesses: the number of paths between two nodes is typically huge and the majority of the corresponding source phrases do not occur in the phrase-table.

We omitted the probabilities for notational convenience. The extensions are straightforward. Note that we store only the target phrases $\tilde{e}$ in the set of possible translations $E(j', j)$ and not the source phrases $\tilde{f}$. This is based on the assumption that the models which are conditioned on the source phrase $\tilde{f}$ are independent of the context outside the phrase pair $(\tilde{f}, \tilde{e})$. This assumption holds for the standard phrase and word translation models. Thus, we have to keep only the target phrase with the highest probability. It might be violated by lexicalized distortion models (dependent on the configuration); in that case we have to store the source phrase along with the target phrase and the probability, which is again straightforward.

4.2 Algorithm

The algorithm for matching the source phrases of the input graph G and the prefix tree T is presented in

Figure 3: Algorithm `phrase-match` for matching source phrases of input graph G and prefix tree T. Input: graph G, prefix tree T, translation options $E(k)$ for all tree nodes k; output: translation options $E(j', j)$ for all graph nodes j' and j.

```
0  FOR j' = 1 TO J DO
1      stack.push(j', k_0)
2      WHILE not stack.empty() DO
3          (j, k) = stack.pop()
4          E(j', j) = E(j', j) ∪ E(k)
5          FOR n = 1 TO N_j DO
6              IF (f^G_{j,n} = ε)
7              THEN stack.push(s^G_{j,n}, k)
8              ELSE IF (∃m : f^G_{j,n} = f^T_{k,m})
9                      THEN stack.push(s^G_{j,n}, s^T_{k,m})
```

Fig. 3. Starting from a graph node j', we explore the part of the graph which corresponds to known source phrase prefixes and generate the sets $E(j', j)$ incrementally based on Eq. 6. The intermediate states are represented as pairs (j, k) meaning that there exists a path in the input graph from node j' to node j which is labeled with the source phrase $\tilde{f}_k$, i.e. the source phrase that leads to node k in the prefix tree. These intermediate states are stored on a stack. After the initialization in line 1, the main loop starts. We take one item from the stack and update the translation options $E(j', j)$ in line 4. Then, we loop over all outgoing edges of the current graph node j. For each edge, we first check if the edge is labeled with an ϵ in line 6. In this special case, we go to the successor node in the input graph $s^G_{j,n}$, but remain in the current node k of the prefix tree. In the regular case, i.e. the graph edge label is a regular word, we check in line 8 if the current prefix tree node k has an outgoing edge labeled with that word. If such an edge is found, we put a new state on the stack with the two successor nodes in the input graph $s^G_{j,n}$ and the prefix tree $s^T_{k,m}$, respectively.

4.3 Computational Complexity

In this section, we will analyze the computational complexity of the algorithm. The computational complexity of lines 5-9 is in $\mathcal{O}(N_j \log M_k)$, i.e. it depends on the branching factors of the input graph and the prefix tree. Both are typically small. An exception is the branching factor of the root node k_0 of the prefix tree, which can be rather large, typically it is the vocabulary size of the source language. But, as described in Sec. 3, we can access the successor

nodes of the root node of the prefix tree in $\mathcal{O}(1)$, i.e. in constant time. So, if we are at the root node of the prefix tree, the computational complexity of lines 5-9 is in $\mathcal{O}(N_j)$. Using hashing at the interior nodes of the prefix tree would result in a constant time lookup at these nodes as well. Nevertheless, the sorted array implementation that we chose has the advantage of faster loading from disk which seems to be more important in practice.

An alternative interpretation of lines 5-9 is that we have to compute the intersection of the two sets f_j^G and f_k^T, with

$$f_j^G = \left\{ f_{j,n}^G \,\middle|\, n = 1, ..., N_j \right\} \quad (7)$$

$$f_k^T = \left\{ f_{k,m}^T \,\middle|\, m = 1, ..., M_k \right\}. \quad (8)$$

Assuming both sets are sorted, this could be done in linear time, i.e. in $\mathcal{O}(N_j + M_k)$. In our case, only the edges in the prefix tree are sorted. Obviously, we could sort the edges in the input graph and then apply the linear algorithm, resulting in an overall complexity of $\mathcal{O}(N_j \log N_j + M_k)$. As the algorithm visits nodes multiple times, we could do even better by sorting all edges of the graph during the initialization. Then, we could always apply the linear time method. On the other hand, it is unclear if this pays off in practice and an experimental comparison has to be done which we will leave for future work.

The overall complexity of the algorithm depends on how many phrases of the input graph occur in the phrase-table. In the worst case, i.e. if all phrases occur in the phrase-table, the described algorithm is not more efficient than the naive algorithm which simply enumerates all phrases. Nevertheless, this does not happen in practice and we observe an exponential speed up compared to the naive algorithm, as will be shown in Sec. 6.3.

5 Online Machine Translation

Beside speech translation, the presented phrase-table data structure has other interesting applications. One of them is online MT, i.e. an MT system that is able to translate unseen sentences without significant delay. These online MT systems are typically required if there is some interaction with human users, e.g. if the MT system acts as an interpreter in a conversation, or in real-time systems. This situation is different from the usual research environment where typically a fair amount of time is spent to prepare the MT system to translate a certain set of source sentences. In the research scenario, this preparation usually pays off as the same set of sentences is translated multiple times. In contrast, an online MT system translates each sentence just once. One of the more time-consuming parts of this preparation is the filtering of the phrase-table. Using the on-demand loading technique we described in Sec. 3, we can avoid the filtering step and directly translate the source sentence. An additional advantage is that we load only small parts of the full phrase-table into memory. This reduces the memory requirements significantly, e.g. for the Chinese–English NIST task, the memory requirement of the phrase-table is reduced to less than 20 MB using on-demand loading. This makes the MT system usable on devices with limited hardware resources.

Table 1: NIST task: corpus statistics.

			Chinese	English
Train	Sentence pairs		7 M	
	Running words		199 M	213 M
	Vocabulary size		222 K	351 K
Test	2002	Sentences	878	3 512
		Running words	25 K	105 K
	2005	Sentences	1 082	4 328
		Running words	33 K	148 K

6 Experimental Results

6.1 Translation System

For the experiments, we use a state-of-the-art phrase-based statistical machine translation system as described in (Zens and Ney, 2004). We use a log-linear combination of several models: a four-gram language model, phrase-based and word-based translation models, word, phrase and distortion penalty and a lexicalized distortion model. The model scaling factors are optimized using minimum error rate training (Och, 2003).

6.2 Empirical Analysis for a Large Data Task

In this section, we present an empirical analysis of the described data structure for the large data track of the Chinese-English NIST task. The corpus statistics are shown in Tab. 1.

The translation quality is measured using two accuracy measures: the BLEU and the NIST score. Additionally, we use the two error rates: the word error rate (WER) and the position-independent word error rate (PER). These evaluation criteria are computed with respect to four reference translations.

In Tab. 2, we present the translation quality as a

Table 2: NIST task: translation quality as a function of the maximum source phrase length.

src len	NIST 2002 set (dev)				NIST 2005 set (test)			
	WER[%]	PER[%]	BLEU[%]	NIST	WER[%]	PER[%]	BLEU[%]	NIST
1	71.9	46.8	27.07	8.37	78.0	49.0	23.11	7.62
2	62.4	41.2	34.36	9.39	68.5	42.2	30.32	8.74
3	62.0	41.1	34.89	9.33	67.7	42.1	30.90	8.74
4	61.7	41.1	35.05	9.27	67.6	41.9	30.99	8.75
5	61.8	41.2	34.95	9.25	67.6	41.9	30.93	8.72
∞	61.8	41.2	34.99	9.25	67.5	41.8	30.90	8.73

Table 3: NIST task: phrase-table statistics.

src len	number of distinct		avg. tgt
	src phrases	src-tgt pairs	candidates
1	221 505	17 456 415	78.8
2	5 000 041	39 436 617	7.9
3	20 649 699	58 503 904	2.8
4	31 383 549	58 436 271	1.9
5	32 679 145	51 255 866	1.6
total	89 933 939	225 089 073	2.5

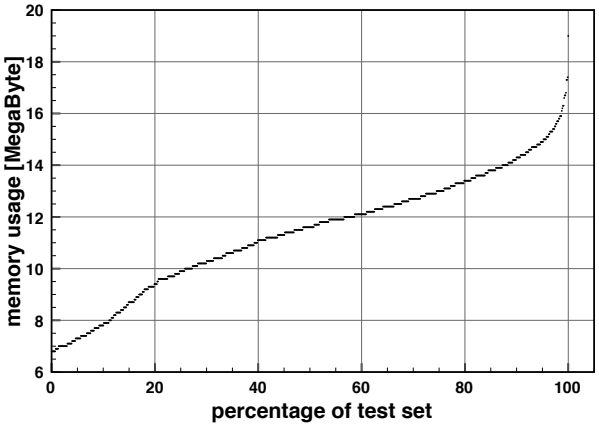

Figure 4: NIST task: phrase-table memory usage per sentence (sorted).

function of the maximum source phrase length. We observe a large improvement when going beyond length 1, but this flattens out very fast. Using phrases of lengths larger than 4 or 5 does *not* result in further improvement. Note that the minor differences in the evaluation results for length 4 and beyond are merely statistical noise. Even a length limit of 3, as proposed by (Koehn et al., 2003), would result in almost optimal translation quality. In the following experiments on this task, we will use a limit of 5 for the source phrase length.

In Tab. 3, we present statistics about the extracted phrase pairs for the Chinese–English NIST task as a function of the source phrase length, in this case for length 1-5. The phrases are not limited to a specific test set. We show the number of distinct source phrases, the number of distinct source-target phrase pairs and the average number of target phrases (or translation candidates) per source phrase. In the experiments, we limit the number of translation candidates per source phrase to 200. We store a total of almost 90 million distinct source phrases and more than 225 million distinct source-target phrase pairs in the described data structure. Obviously, it would be infeasible to load this huge phrase-table completely into memory. Nevertheless, using on-demand loading, we are able to utilize all these phrase pairs with minimal memory usage.

In Fig. 4, we show the memory usage of the described phrase-table data structure per sentence for the NIST 2002 test set. The sentences were sorted according to the memory usage. The maximum amount of memory for the phrase-table is 19 MB; for more than 95% of the sentences no more than 15 MB are required. Storing all phrase pairs for this test set in memory requires about 1.7 GB of memory, i.e. using the described data structures, we not only avoid the limitation to a specific test set, but we also reduce the memory requirements by about two orders of a magnitude.

Another important aspect that should be considered is translation speed. In our experiments, the described data structure is *not* slower than the traditional approach. We attribute this to the fact that we use a binary format that is a memory map of the data structure used internally and that we load the data in rather large, coherent chunks. Additionally, there is virtually no initialization time for the phrase-table which decreases the overhead of parallelization and therefore speeds up the development cycle.

6.3 Speech Translation

The experiments for speech translation were conducted on the European Parliament Plenary Sessions (EPPS) task. This is a Spanish-English speech-to-speech translation task collected within the TC-Star

Table 4: EPPS task: corpus statistics.

Train	Spanish	English
Sentence pairs	1.2 M	
Running words	31 M	30 M
Vocabulary size	140 K	94 K
Test confusion networks	**Full**	**Pruned**
Sentences	1 071	
Avg. length	23.6	
Avg. / max. depth	2.7 / 136	1.3 / 11
Avg. number of paths	10^{75}	264 K

project. The training corpus statistics are presented in Tab. 4. The phrase-tables for this task were kindly provided by ITC-IRST.

We evaluate the `phrase-match` algorithm in the context of confusion network (CN) decoding (Bertoldi and Federico, 2005), which is one approach to speech translation. CNs (Mangu et al., 2000) are interesting for MT because the reordering can be done similar to single best input. For more details on CN decoding, please refer to (Bertoldi et al., 2007). Note that the `phrase-match` algorithm is not limited to CNs, but can work on arbitrary word graphs.

Statistics of the CNs are also presented in Tab. 4. We distinguish between the full CNs and pruned CNs. The pruning parameters were chosen such that the resulting CNs are similar in size to the largest ones in (Bertoldi and Federico, 2005). The average depth of the full CNs, i.e. the average number of alternatives per position, is about 2.7 words whereas the maximum is as high as 136 alternatives.

In Fig. 5, we present the average number of phrase-table look-ups for the full EPPS CNs as a function of the source phrase length. The curve 'CN total' represents the total number of source phrases in the CNs for a given length. This is the number of phrase-table look-ups using the naive algorithm. Note the exponential growth with increasing phrase length. Therefore, the naive algorithm is only applicable for very short phrases and heavily pruned CNs, as e.g. in (Bertoldi and Federico, 2005).

The curve 'CN explored' is the number of phrase-table look-ups using the `phrase-match` algorithm described in Fig. 3. We do *not* observe the exponential explosion as for the naive algorithm. Thus, the presented algorithm effectively solves the combinatorial problem of matching phrases of the input CNs and the phrase-table. For comparison, we plotted also the number of look-ups using the `phrase-match` algorithm in the case of single-best input, labeled 'single-best explored'. The maximum phrase length for these experiments is seven. For CN input, this length can be exceeded as the CNs may contain ϵ-transitions.

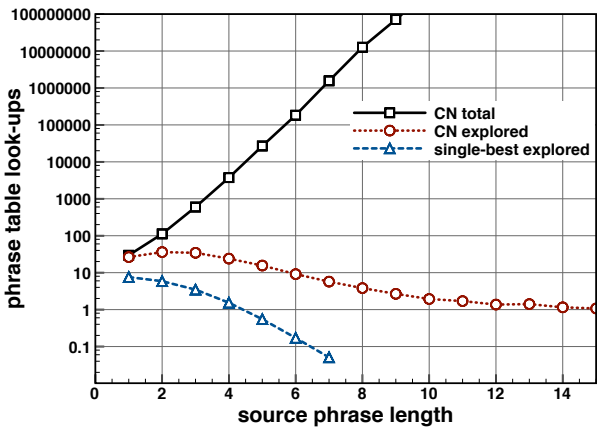

Figure 5: EPPS task: avg. number of phrase-table look-ups per sentence as a function of the source phrase length.

Table 5: EPPS task: translation quality and time for different input conditions (CN=confusion network, time in seconds per sentence).

Input type	BLEU[%]	Time [sec]
Single best	37.6	2.7
CN pruned	38.5	4.8
full	38.9	9.2

In Tab. 5, we present the translation results and the translation times for different input conditions. We observe a significant improvement in translation quality as more ASR alternatives are taken into account. The best results are achieved for the full CNs. On the other hand, the decoding time increases only moderately. Using the new algorithm, the ratio of the time for decoding the CNs and the time for decoding the single best input is 3.4 for the full CNs and 1.8 for the pruned CNs. In previous work (Bertoldi and Federico, 2005), the ratio for the pruned CNs was about 25 and the full CNs could not be handled.

To summarize, the presented algorithm has two main advantages for speech translation: first, it enables us to utilize large CNs, which was prohibitively expensive beforehand and second, the efficiency is improved significantly.

Whereas the previous approaches required careful pruning of the CNs, we are able to utilize the unpruned CNs. Experiments on other tasks have shown that even larger CNs are unproblematic.

7 Conclusions

We proposed an efficient phrase-table data structure which has two key properties:

1. **On-demand loading.**
   We are able to store hundreds of millions of phrase pairs and require only a very small amount of memory during decoding, e.g. less than 20 MB for the Chinese-English NIST task. This enables us to run the MT system on devices with limited hardware resources or alternatively to utilize the freed memory for other models. Additionally, the usual phrase-table filtering is obsolete, which is important for online MT systems.

2. **Prefix tree data structure.**
   Utilizing the prefix tree structure enables us to efficiently match source phrases against the phrase-table. This is especially important for speech translation where the input is a graph representing a huge number of alternative sentences. Using the novel algorithm, we are able to handle large CNs, which was prohibitively expensive beforehand. This results in more efficient decoding and improved translation quality.

We have shown that this data structure scales very well to large data tasks like the Chinese-English NIST task. The implementation of the described data structure as well as the `phrase-match` algorithm for confusion networks is available as open source software in the MOSES toolkit[1].

Not only standard phrase-based systems can benefit from this data structure. It should be rather straightforward to apply this data structure as well as the `phrase-match` algorithm to the hierarchical approach of (Chiang, 2005). As the number of rules in this approach is typically larger than the number of phrases in a standard phrase-based system, the gains should be even larger.

The language model is another model with high memory requirements. It would be interesting to investigate if the described techniques and data structures are applicable for reducing the memory requirements of language models.

Some aspects of the `phrase-match` algorithm are similar to the composition of finite-state automata. An efficient implementation of on-demand loading (not only on-demand computation) for a finite-state toolkit would make the whole range of finite-state operations applicable to large data tasks.

[1] http://www.statmt.org/moses

Acknowledgments

This material is partly based upon work supported by the DARPA under Contract No. HR0011-06-C-0023, and was partly funded by the European Union under the integrated project TC-STAR (IST-2002-FP6-506738, http://www.tc-star.org). Additionally, we would like to thank all group members of the JHU 2006 summer research workshop *Open Source Toolkit for Statistical Machine Translation*.

References

N. Bertoldi and M. Federico. 2005. A new decoder for spoken language translation based on confusion networks. In Proc. *IEEE Automatic Speech Recognition and Understanding Workshop*, pages 86–91, Mexico, November/December.

N. Bertoldi, R. Zens, and M. Federico. 2007. Speech translation by confusion networks decoding. In Proc. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Honolulu, Hawaii, April.

C. Callison-Burch, C. Bannard, and J. Schroeder. 2005. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In Proc. *43rd Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 255–262, Ann Arbor, MI, June.

D. Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In Proc. *43rd Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 263–270, Ann Arbor, MI, June.

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In Proc. *Human Language Technology Conf. / North American Chapter of the Assoc. for Computational Linguistics Annual Meeting (HLT-NAACL)*, pages 127–133, Edmonton, Canada, May/June.

L. Mangu, E. Brill, and A. Stolcke. 2000. Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer, Speech and Language*, 14(4):373–400, October.

L. Mathias and W. Byrne. 2006. Statistical phrase-based speech translation. In Proc. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 561–564, Toulouse, France, May.

H. Ney, R. Haeb-Umbach, B. H. Tran, and M. Oerder. 1992. Improvements in beam search for 10000-word continuous speech recognition. In Proc. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 9–12, San Francisco, CA, March.

F. J. Och. 2003. Minimum error rate training in statistical machine translation. In Proc. *41st Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 160–167, Sapporo, Japan, July.

R. Zens and H. Ney. 2004. Improvements in phrase-based statistical machine translation. In Proc. *Human Language Technology Conf. / North American Chapter of the Assoc. for Computational Linguistics Annual Meeting (HLT-NAACL)*, pages 257–264, Boston, MA, May.

Y. Zhang and S. Vogel. 2005. An efficient phrase-to-phrase alignment model for arbitrarily long phrases and large corpora. In Proc. *10th Annual Conf. of the European Assoc. for Machine Translation (EAMT)*, pages 294–301, Budapest, Hungary, May.