An Efficient Graph Search Decoder for Phrase-Based Statistical Machine Translation

Brian Delaney, Wade Shen

MIT Lincoln Laboratory
244 Wood St
Lexington, MA 02420
{bdelaney,swade}@ll.mit.edu

Timothy Anderson

Air Force Research Laboratory
2255 H St.
Wright-Patterson AFB, OH 45433
Timothy.Anderson@wpafb.af.mil

Abstract

In this paper we describe an efficient implementation of a graph search algorithm for phrase-based statistical machine translation. Our goal was to create a decoder that could be used for both our research system and a real-time speech-to-speech machine translation demonstration system. The search algorithm is based on a Viterbi graph search with an A* heuristic. We were able to increase the speed of our decoder substantially through the use of on-the-fly beam pruning and other algorithmic enhancements. The decoder supports a variety of reordering constraints as well as arbitrary n-gram decoding. In addition, we have implemented disk based translation models and a messaging interface to communicate with other components for use in our real-time speech translation system.

1. Introduction

The current state-of-the-art in machine translation uses a phrase-based approach to translate individual sentences from the source language to the target language. This technique [1] gave significant improvement over word to word translation originally developed at IBM [2]. However, regardless of the underlying models, the search or decoding process in statistical machine translation is computationally demanding, particularly with regard to word or phrase reordering. Effective pruning and novel reordering techniques have helped reduce the search space to something more manageable, but search still remains a difficult problem in statistical machine translation.

Our goal was to create a fast and lightweight decoder that could be used for our research system as well as for a real-time speech-to-speech translation demonstration system. For the research system, the decoder had to support two-pass decoding via n-best list re-ranking or lattice rescoring. The decoder also had to be memory efficient enough to handle search with very large numbers of active nodes. For the real-time demonstration system, we required fast decoding as well as a common API that would allow integration with various other components such as speech recognition and text-to-speech systems. Real-time decoding also requires that the translation models, which can approach several gigabytes in size, be read efficiently from disk on demand as opposed to the pre-filtering of models typically done during batch processing.

[1]This work is sponsored by the United States Air Force Research Laboratory under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

2. System Overview

The decoding strategy we implemented is a Viterbi beam search with an A* heuristic based on words not yet translated. The decoder begins by dividing the input sentence into many overlapping word segments up to a given maximum phrase length. Each phrase segment is then cross referenced with the phrase table inventory and the top-N translation candidates for the source phrase are stored in memory. A trigram language model is also loaded into memory, and all partial translations are given an initial language model score. During search, the language model probabilities for the individual words are adjusted given the new context of previously translated words.

For our research system, we employ a two-pass decoding strategy. N-best lists are generated from the output phrase lattice, and additional feature functions are added (i.e. higher order language model, word posterior scores, etc.) The resulting N-best lists are then re-ranked according to these new features and the best scoring output is selected [3].

2.1. Translation and Language Models

The basic phrase-translation model we employ is described in detail [4] and [5]. We use GIZA++ [2], [6] and [4] to generate alignments between foreign language and English language words/tokens in both directions (f-to-e and e-to-f). An expanded alignment is then computed using heuristics that interpolate between the intersection and union of these bidirectional word alignments as detailed in [4] and [5]. Then phrases are extracted from the expanded alignments to build the phrase model.

We extract translation models for both translation direc-

tions (i.e. $P(f|e)$ and $P(e|f)$). In addition to these models, we add lexical weights extracted from the expanded alignment process in both translation directions [5] and a fixed phrase penalty [7].

We use the SRILM language model toolkit [8] to generate an N-gram language model for use during decoding. These models are trained using modified Knesser-Ney interpolation.

2.2. Distortion Modeling

The distortion model we use is still rather weak and is based on the model used in the Pharaoh decoder [9]. It is simply a distance penalty based on the overall movement of source words:

$$D_p = -\sum_i |last\_word_{i-1} + 1 - first\_word_i| \quad (1)$$

That is, the distortion measure between two phrases is the number of words between the last word of the previously translated phrase and the first word of the new phrase. In practice, we place limits on the allowed reordering patterns to limit the search space and improve accuracy. This can be through a fixed distance limit or other linguistically inspired constraints.

2.3. Minimum Error Rate Training

Table 1: Translation model parameters.

Parameter	Description
$p(f\|e)$	Forward phrase translation probability
$p(e\|f)$	Backward phrase translation probability
$LexW(f\|e)$	Forward lexical weighting
$LexW(e\|f)$	Backward lexical weighting
W_p	Number of output words
P_p	Fixed penalty for each source phrase
$p_{lbo}(e\|f)$	Lexical backoff penalty
D_p	Distortion penalty
$p(e)$	N-gram language model

Our translation model employs a log-linear combination of many different model parameters. The model parameters used are shown in Table 1. The models are tuned using a development corpus using an algorithm similar to the one described in [10]. The error criterion minimized is $1 - BLEU$. N-best lists are generated after decoding on the development corpus and the weights for the model parameters shown in Table 1 are set to minimize the overall error rate. Additional features, such as higher-order language models, can also be added at this stage and their weights adjusted as well.

3. Decoder Implementation

The decoder uses a Viterbi beam search with an A* heuristic. Pseudocode for the algorithm is shown in Figure 1. First, the various data structures are initialized and the models are loaded. The initial node list contains a single node with the begin sentence marker, <s>. The input source sentence is divided into individual phrases up to a maximum phrase length. Each input phrase is looked up in the phrase inventory. For smaller translation tasks with limited training data, the entire phrase inventory is loaded into memory and a chained hash table based on the input phrase is created. This hash table structure allows fast access to the phrase inventory. The hash table entry for a particular source phrase contains a pointer to all possible translations of that phrase sorted by the weighted model parameter scores. This allows for a selection of the top-N best scoring phrases for efficient search.

Larger translation tasks have phrase inventories that are too large to be stored in memory during decoding. When the input sentences are known prior to the decoding process, as in batch processing of evaluation corpora, it is reasonable to produce smaller sets of models specific to particular input sentences. However, for real-time speech translation, this is not practical nor desireable. We do not have prior knowledge of the spoken input and cannot filter the phrase inventory in advance. Instead, we pre-index the phrase table inventory using Berkeley DB software, and we simply treat the model as a disk based hash table. Since Berkeley DB is designed for very fast access to large amounts of data, there is almost no performance penalty with this approach.

The search proceeds according to the number of source words covered at a given time (line 2). For each source phrase in the phrase inventory, possible translation candidates can be linked to previous nodes according to the number of source words in the currently selected phrase. That is, a phrase covering three input words, will link back to nodes from three iterations prior. The distortion is calculated according to equation (1). In addition, the coverage vectors from the left node and this new candidate phrase are checked to make sure they cover different words. The same word cannot be covered more than once on a single path.

For the A* heuristic (line 9), we use the highest probability translation options for the remaining words as in the Pharaoh decoder [9]. This includes the translation and language model probabilities for the remaining phrase scores in isolation of one another (i.e. no distortion or language models scores across phrases). These costs can be computed up front with a simple dynamic programming algorithm and stored in a table for later use. We experimented with several extensions to the future cost estimate. We tried augmenting this heuristic with a language model lookahead as well as a best case distortion cost. While both of these were admissible heuristics, we observed no consistent gain in performance across all our experiments.

The inner loop of the search (line 10) cycles through all possible translations (up to a predefined limit) for each phrase that meets the distortion criteria. The language model scores are updated only at phrase boundaries. Node expansion is controlled by indexing all created nodes into a hash

```
 1. Initialize data structures
 2. for cw from 0 to number of words
 3.   for each src_phrase in src_phrase_list
 4.      l_index  = cw - len(src_phrase)
 5.      for l_node in node_list[l_index]
 6.         calculate distortion between l_node and src_phrase
 7.         if (dist > dlimit) or (l_node and src_phrase cover the same word)
 8.               continue
 9.         get future_cost of remaining words
10.         for each translation in src_phrase
11.            update lm scores at phrase boundary using l_node context
12.            find or create curr_node with current coverage and lm context
13.            link curr_node to l_node with current translation
14.            if current_score > curr_node.best_score
15.              set current translation to best path of curr_node
16.    beam and histogram pruning
17. back-trace and output best translation
```

Figure 1: Pseudocode for the decoding algorithm.

table. The hash function is computed using a data structure that contains the word coverage vector and the right most words or language model context for the node. For example, the hash table index might be ”00110 would like”, indicating that the third and fourth input source words were covered and the language model context is ”would like.” If a node already containing the current coverage vector and language model context exists in the hash table (line 12), then the current translation is added to this node and the node is linked back to the left node in the graph. If a proper node does not exist, a new one is created and added to the hash table. The back-off structure of the N-gram language model is also used here to eliminate unnecessary node expansion for language model contexts that have no possible n-gram expansion. This significantly reduces the number of nodes created during search.

The final step in the inner loop is to update the best path for the current node if the current score is better (line 14). This is the Viterbi approximation step. It is important to note that the score used here is the total log probability plus the future cost estimate computed earlier. This total path score plus future cost is also used during beam and histogram pruning. After the main search is complete, all remaining nodes are joined with the end of sentence marker, `</s>`, and language model scores are updated across phrase boundaries. The remaining search graph can be traced back along the highest scoring path to get the best path and thus the best candidate translation.

3.1. OOV Words

The problem of out-of-vocabulary (OOV) words requires some small changes to the above algorithm. We chose to handle the issue with an open vocabulary language model. The SRILM toolkit allows for unknown words by assigning a non-zero probability to word sequences containing the `<unk>` tag. In the case of a single unknown word, we simply pass the word through with translation model probabilities of 1.0 and language model probability according to the open vocabulary language model. OOV words are required to remain adjacent to their previously translated source phrases. Their distortion cost must be zero. Since the nodes associated with these OOV input words now have probabilities that are artificially higher than the remaining nodes. As such, we do not use nodes containing OOV words in order to set our beam threshold during pruning.

3.2. On-the-fly Beam Pruning

During some initial profiling of the code, we found that a significant amount of time was spent looking up various n-grams in the memory-based language model. Individual N-grams are also stored in a hash table for fast access, but many of these language model lookups are unnecessary as the paths are eventually pruned out. Additionally, there is significant node expansion for these unpromising paths, which slows the sorting that takes place prior to pruning. In order to lessen the impact of frivolous language model lookup and node expansion, we employed an active or on-the-fly beam pruning approach.

We start by keeping track of the best path score during each iteration of the main outer loop. When a translation option with higher probability occurs, we update the best path score. Then we apply beam pruning right after steps 9 and 10. If the partial score does not fall within the currently selected beam, the translation option is ignored and the loop continues with the next translation option. By checking the score after step 9, we eliminate the inner loop for translations options whose combined partial path costs, distortion, and future cost estimates do not fall within the beam. If at this stage, the probability falls outside the beam, then any

further reduction from the application of translation model scores in the inner loop will also be outside the beam. After step 10, we include the translation model scores, which further lowers the probability. Once again, if the hypotheses is outside the beam at this point, then there is no reason for further analysis, and we can skip the costly language model lookup.

Table 2: Results from beam pruning enhancements on the IWSLT06-dev4 data set.

Description	Chars/sec	BLEU
No on-the-fly pruning	1.06	20.13
prune after step 9	1.08	20.08
prune after step 9,10	2.29	20.08
Sort phrase list	2.99	20.19

Table 3: Beam width comparison for different pruning techniques on the IWSLT06-dev4 data set.

	On-the-fly pruning		Normal pruning	
Beam width	Chars/sec	BLEU	Chars/sec	BLEU
0.0003	2.99	20.19	1.06	20.13
0.001	7.31	19.44	1.97	19.77
0.005	28.21	19.36	7.29	19.74
0.01	50.75	18.84	13.1	18.91

While this technique works reasonably well, it operates under the assumption that the actual best partial path score is found early on. In the worst case, the best score is found last, and there is no benefit. To increase the chances that the highest probability score is found early, we use a best-first approach with respect to the source phrase list. In particular, the source phrases are initially sorted according to their future cost estimate. This helps reduce the average length of time until the best score is found, and even if the best scoring path is not found, there is some benefit to having an active beam width that is at least close to optimal. Using the training and development sets from the 2006 International Workshop on Spoken Language Translation (IWSLT06), we show the BLEU scores decoding times, expressed as characters per second, in Table 2 for a Chinese to English translation. (Our Chinese system uses a combination of word and character segmented alignments to produce a final phrase table based on individual Chinese characters.) The IWSLT06 training data consists of approximately 40,000 parallel sentences in the travel expression domain. Supplied with the training data were three development sets of approximately 500 sentences each, with 2 containing 16 references and another with only 7 references. We optimized the model weights on the CSTAR03 development set and decoded on the IWSLT06-dev4 set. The lower cased output was post-processed to restore mixed-case information, and the result was scored against the mixed-case references.

The biggest reduction in runtime came from removing unnecessary language model lookup after step 10. In the final configuration, using a sorted phrase list to quickly arrive at the best scoring hypothesis yields a runtime that is almost 3 times faster than without on-the-fly pruning. The small changes in BLEU scores are due to borderline hypotheses that are pruned out before complete language model expansion. Additionally, we varied the beam width to compare the optimized vs. non-optimized versions. The results are in Table 3. We observe some small reduction in BLEU score when comparing the optimized vs. un-optimized version as the beam becomes tighter. However, the optimized version is still much faster, even if slightly sub-optimal with respect to accuracy.

3.3. Reordering Constraints

In this section, we explore the performance of our decoder under different word reordering constraints, including the so-called ITG constraints, IBM constraints, free ordering with a distance limit, as well as an additional approach that provides good accuracy with definite speed advantages. Additional reordering constraints are implemented in place of or in addition to lines 6,7 in the algorithm shown in Figure 1. In the case of free word order, any source word can follow any other word with respect to the construction of the search graph. Often a hard limit is placed on the maximum distance allowed. A comparison of different word reordering constraints was made in [11], and a simple algorithm for imposing constraints derived from Inversion Transduction Grammars (ITG) [12] during search was introduced. If we assume a simple, 4-word, input sequence of (1,2,3,4), the ITG constraints do not allow certain reordering patterns (i.e. 3,1,4,2 and 2,4,1,3). These "inside-out" patterns are not likely to follow reordering patterns of language. The number of permutations with ITG constraints grows much slower than with free word reordering [13]. Under the IBM constraints, input source words can be chosen out of order so long as they fill one of the first k uncovered words [11]. By increasing k, more reordering patterns are possible. This can provide fast decoding when k is small for closely related language pairs, but it can be quite slow when k is large, as for Chinese-English.

One issue we found when analyzing the performance of our decoder with respect to word reordering was the issue of incomplete paths in the graph. Using a free word order with a hard limit on word movement resulted in graphs with incomplete translation paths, resulting in wasted search effort. For example, consider the translation of 4 input source words using a distortion limit of 2. If we build the graph from left to right using only the distortion limit as a guide, the result is shown in Figure 2. If we initially choose the reordering sequence 2,4,3, we have not violated the constraints placed on the distortion (Using equation (1) the distortion penalty from 2 to 4 is $D_p = |2 + 1 - 4| = 1$, and from 4 to 3 it is $D_p = |4 + 1 - 3| = 2$). However, the remaining word, 1,

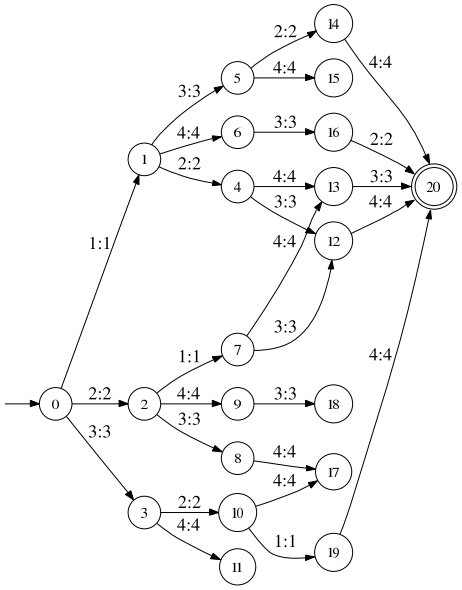

Figure 2: Reordering graph for four input words with a distortion limit of 2.

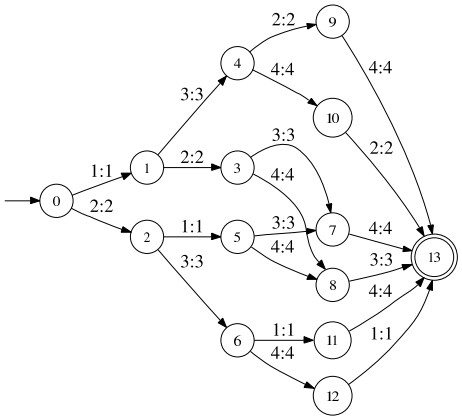

Figure 3: IBM constraints for 4-word input sequence with $k = 2$.

cannot be selected within the distortion limit constraints (in this case, $D_p = |3 + 1 - 1| = 3$). Even in this simple example, there are many instances of incomplete paths and thus wasted search effort. While this graph could be trimmed using a forward-backward pass, this is not practical for longer sentences with greater limits on distortion. In the future, we would like to address this problem with a better implementation that would detect the incomplete paths in advance.

In practice, a hard distortion limit is often placed on the ITG constraints as a way to limit the search time, but this too suffers from the problem of incomplete paths in the search graph. However, the IBM constraints, by design, do not have this problem. Figure 3 shows the same reordering graph under the IBM constraints with $k = 2$. The IBM style constraints still produce larger numbers of permutations and thus decoding times somewhere between free word order and ITG constraints.

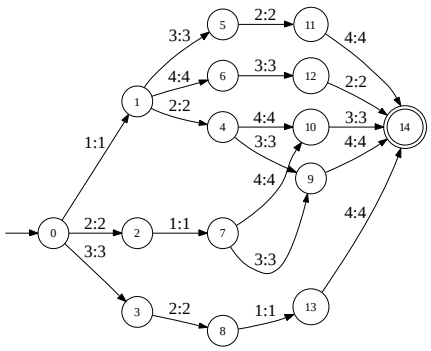

Figure 4: Reordering graph for four input words with distortion limit of 2 using proposed reordering constraints.

During development of our decoder, we implemented an additional set of constraints, which is similar to the IBM style constraints but produces fewer possible permutations. These produced accurate results with shorter decoding times. Our proposed algorithm limits reordering patterns in the following way. Let l_{min} be the left-most uncovered source sentence position in S, and let cw be the total number of covered words in both C and S. We wish to join candidate phrase C with an existing path covering source words S. C and S must not cover the same source words and they must satisfy the distortion constraint in (1). Do *not* allow expansion along this path if each of the following conditions are true:

1. $distortion > dlimit$
2. $l_{min} < cw$
3. $cw < C_{first}$
4. $|C_{last} - l_{min}| \geq dlimit$

where C_{first} and C_{last} are the first and last word indices of candidate phrase C, and $dlimit$ is the maximum allowed distortion.

That is, C must fill the first available gap in source coverage when expanding from the current coverage, and new gaps in source coverage must be less than the distortion limit. (The IBM constraints allow the first k empty source positions to be filled.) Since C can be shorter than the gap, this is not strictly an adjacent swap constraint. After applying this algorithm, the resulting reordering graph for our toy example is shown in Figure 4. In this example, the compressed reordering graph contains all of the permutations available in Figure 2 but without the incomplete paths. However, this is not generally the case; the above algorithm limits the total number of reordering permutations. For example, consider the same four word input sentence, (1,2,3,4), with a distortion limit of three. The above algorithm does not allow the sequences (2,4,3,1) and (3,4,2,1).

3.4. Phrase Lattice Generation

Since we use a graph search algorithm, we are able to very quickly produce an output lattice suitable for second pass

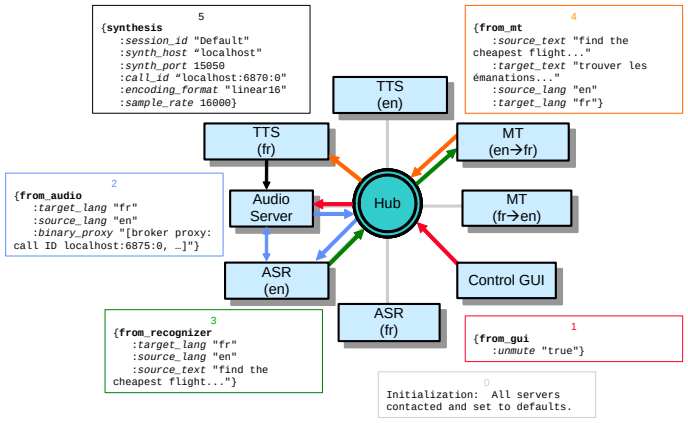

Figure 6: Speech Translation with Galaxy Communicator.

rescoring or minimum error rate training. We simply traverse the remaining graph structure after the search is complete. A forward-backward pass is used to trim any unconnected nodes that may have not been expanded due to pruning. Nodes are given a unique number and links between nodes contain phrases with individual model scores from Table 1. This data structure allows for fast nbest list generation without an additional pass to retrieve model components. We use the HTK lattice format which can then be processed by the SRILM toolkit's *lattice-tool*. Output phrases are joined as a *multi-word* by the underscore, "_", character (i.e. "i_would_like".) An example of an output phrase lattice is shown in Figure 5. The bit vectors in the node names indicate source word coverage. This information is contained in the comments of the HTK lattice output if needed.

3.5. Galaxy Communicator Interface

For the real-time speech translation system, we used the Galaxy Communicator architecture [14] to provide a common API between the speech recognition, machine translation, and speech synthesis components. A simple GUI is used for microphone control and to present recognition and translation results to the user. Although somewhat dated, the Communicator architecture provides a simple interface to try a variety of different technology components and can be used in overall system evaluation and testing. Communicator uses a hub and spoke architecture where all interaction is sent through the hub and routed appropriately, see Figure 6. In our case, the data is routed to the appropriate recognition, translation, and speech synthesis servers via the language attributes attached to each frame. The servers can be distributed across a network or run locally on one machine. We have integrated a variety of technology components into this architecture, both research and commercial quality systems. Our decoder supports this interface and allows for translation of individual spoken sentences on large vocabulary tasks via disk-based translation models. A version of the system, including speech recognition and synthesis, runs on a single Linux-based laptop.

4. Results

In this section we outline some results from our decoder under various configurations using the IWSLT06 training and development data. In the 2006 evaluation, both the Chinese-English (CE) and Japanese-English (JE) language pairs had approximately 40,000 sentence pairs, all of which were used for both translation and target language model training. The Italian-English (IE) language pair was only supplied with 20,000 sentence pairs. For the IE direction, we chose to use only the 20,000 English sentences for language model training. Supplied with the IWSLT06 evaluation were 4 development sets, with the final set, *devset4*, approximating that of a speech transcript. There are only 7 references for this last development set. For each of our test conditions, we optimized model weights using *devset1* and tested on *devset4*. This devset4 did not contain punctuation, so punctuation was automatically added with the SRILM *hidden-ngram* tool using models derived from the training set. The target language case information was restored using a statistical model and the SRILM *disambig* tool. Pruning parameters were kept fixed for all tests. For scoring, we used BLEU4 with case and punctuation information in the references. For both the IE and CE directions, we used alignments from GIZA and the competitive linking algorithm [15]. In the CE case, the phrase table was generated using both character and word segmented alignments, and all source words were character segmented prior to counting. No rescoring was performed on the output. The results are from the 1-best output directly from the decoder.

The results for some of the experiments are shown in Table 4. The first column describes the evaluation configuration. The distortion limit or k value for IBM constraints was set to 10 for CE and JE and 3 for IE. We performed experiments using free word order, IBM and ITG reordering constraints as well as the constraints outlined in section 3.3, labeled as NEW in the table. In addition, we varied the language model order from 3-gram to 5-gram, which is indicated by 3g, 4g, and 5g in the test configuration column. The average number of words (or characters in the CE direction) processed per second and the BLEU score are reported. The best scoring configuration is in bold. Given the relatively small test set, 489 sentences, we should be cautious in the analysis and recognize that there is some "noise" in the automatic evaluation measures. We can, however, notice some general trends from the data.

The first line of the table shows the results for the Pharaoh decoder [9] after optimization and testing on the same development set and with the same pruning thresholds as our decoder. The Pharaoh results are comparable to the second row of the table (free-3g). In general, our decoder produces output with similar BLEU scores in 1/2 to 1/4 of the time in its base configuration. However, we present the Pharaoh results to help ground the performance of our decoder with some-

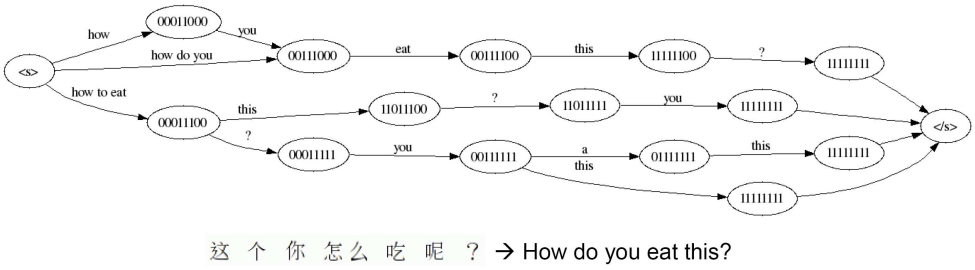

Figure 5: Output phrase lattice example.

Table 4: Decoding times and BLEU scores on IWSLT06 development set 4 using different reordering constraints and language model order. Limits on distortion are 10 for CE and JE and 3 for IE.

Configuration	Language Pair					
	CE		JE		IE	
	BLEU	Chars/sec	BLEU	Words/sec	BLEU	Words/sec
Pharaoh	20.41	0.85	23.07	1.39	35.63	55.48
free-3g	20.19	2.99	22.79	5.39	35.90	113.36
free-4g	20.73	1.45	21.76	2.26	35.64	63.06
free-5g	20.39	1.23	21.99	1.65	36.92	42.93
IBM-3g	20.31	3.70	22.55	6.14	36.60	201.05
IBM-4g	20.15	0.92	21.64	2.04	36.77	124.09
IBM-5g	20.29	0.66	23.04	2.05	36.56	81.15
ITG-3g	20.18	4.36	21.99	7.01	35.70	162.99
ITG-4g	18.89	1.04	22.56	3.50	36.81	60.99
ITG-5g	20.31	1.11	22.39	2.38	36.78	48.45
NEW-3g	19.10	8.52	**23.23**	12.72	36.56	305.29
NEW-4g	20.38	1.70	22.03	5.29	**36.96**	216.92
NEW-5g	**20.90**	1.54	22.81	4.36	36.66	142.47

thing familiar, rather than to perform a head-to-head comparison.

The data sparseness issue makes it difficult to infer improvement with increasing N-gram order while the decoding times clearly increase from 1.5 to 4 times with each increase in order. Trigram decoding works quite well in some cases, including the NEW-3g/JE case and the free-3g/JE case. In general, reordering constraints had only slight impact on BLEU score but a much larger effect on decoding times. The proposed reordering constraints (NEW) generally ran 2-3 times faster than other techniques and produced scores that were adequate and sometimes better than other reordering constraints. Therefore, the proposed reordering constraints are a good candidate for real-time 1-best speech translation, subject to tuning of other parameters such as pruning thresholds. The proposed constraints seemed to work particularly well in the JE case, offering both fast decoding and high BLEU scores. As the method is not linguistically inspired, it is not clear why this is the case.

To provide some additional data points, we tried using a distortion limit of 5 for all language pairs in Table 5. We also configured the decoder to use all patterns allowed by the ITG constraints (i.e. with no limits on distortion). When the distortion limit was set to 5, as in free-3g-d5, the changes due to reordering constraints are not as clear but the proposed method (NEW) does still show some speed advantage, particularly in the JE and CE directions. As the true value of the ITG constraints is to reduce the total number of permutations for longer input sentences, we ran some additional experiments with all patterns allowed by ITG. The results are in the last 3 lines of Table 5. Decoding times were the same or slightly longer than with ITG with a fixed distortion limit of 10 or 3 and in most cases the scores were slightly better. The exception was the IE language pair, where word movement of more than a few words is usually unnecessary.

5. Conclusion

We have presented a decoding algorithm for phrase based statistical machine translation based on a Viterbi graph search. The decoder includes some simple speed optimizations, including an active or on-the-fly beam pruning approach as well as several different reordering constraints, including a more constrained, fast, approach that can be used

Table 5: Timing and BLEU scores for additional distortion limits for IWSLT06 development set 4.

Configuration	Language Pair					
	CE		JE		IE	
	BLEU	Chars/sec	BLEU	Words/sec	BLEU	Words/sec
free-3g-d5	19.87	6.27	22.36	7.84	35.64	56.74
IBM-3g-d5	19.23	9.22	22.27	14.82	34.89	140.05
ITG-3g-d5	19.91	7.41	21.94	8.55	35.55	55.48
NEW-3g-d5	19.81	21.03	21.93	48.04	34.95	198.97
ITG-3g-unlim	20.13	3.39	22.64	4.63	35.38	83.26
ITG-4g-unlim	20.74	0.89	21.96	2.29	34.95	22.46
ITG-5g-unlim	20.50	0.66	22.33	1.63	33.62	13.38

for real-time translation. In addition to speed enhancements, the decoder supports fast access to disk based models for large vocabulary translation as well as a common interface to other speech technology components via a Galaxy Communicator interface. In the future, we would like to incorporate more complex reordering models through either factored or lexical models of word movement. Such a model should help increase both speed and accuracy. In many cases, free word order or ITG reordering constraints with a hard limit on distortion can provide good translation results. However, we would like to better address the problem of incomplete translation paths (Figure 2) to improve both speed and accuracy. We would also like to investigate decoding of confusion networks or other compact representations of automatic speech recognition output.

6. References

[1] F. Och, C. Tillmann, and H. Ney, "Improved alignment models for statistical machine translation," in *Proc. of EMNLP/WVLC*, 1999.

[2] P. Brown, J. Cocke, S. Pietra, V. Pietra, F. Jelinek, R. Mercer, and P. Roossin, "A statistical approach to machine translation," *Computational Linguistics*, vol. 16, no. 2, 1990.

[3] W. Shen, B. Delaney, and T. Anderson, "The MIT-LL/AFRL MT system," in *International Workshop on Spoken Language Translation*, 2005.

[4] F. J. Och and N. Hermann, "Improved statistical alignment models," in *Proc. of the 38th Annual Meeting of the Association for Computational Linguistics*, October 2000, pp. 440–447.

[5] P. Koehn, F. J. Och, and D. Marcu, "Statistical phrase-based translation," in *Proceedings of the Human Language Technology Conference*, May 2003.

[6] Y. Al-Onaizan, J. Curin, M. Jahr, K. Knight, J. Lafferty, I. D. Melamed, F. J. Och, D. Purdy, N. A. Smith, and D. Yarowsky, "Statistical machine translation: Final report," *Summer Workshop on Language Engineering. John Hopkins University Center for Language and Speech Processing*, 1999.

[7] I. Thayer, E. Ettelaie, K. Knight, D. Marcu, D. S. Munteanu, F. J. Och, and Q. Tipu, "The ISI/USC MT system," in *Proc. of the International Workshop on Spoken Language Translation*, 2004.

[8] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *Proc. Intl. Conf. Spoken Language Processing*, September 2002.

[9] P. Koehn, "Pharaoh: A beam search decoder for phrase-based statistical machine translation models," in *Proceedings of the Association of Machine Translation in the Americas (AMTA-2004)*, October 2004.

[10] F. J. Och, "Minimum error rate training for statistical machine translation," in *Proc. of the 41st Annual Meeting of the Association for Computational Linguistics*, July 2003.

[11] R. Zens, H. Ney, T. Watanabe, and E. Sumita, "Reordering constraints for phrase-based statistical machine translation," in *Proceedings of Coling 2004*. Geneva, Switzerland: COLING, Aug 23–Aug 27 2004, pp. 205–211.

[12] D. Wu, "Stochastic inversion transduction grammars and bilingual parsing of parallel corpora," *Computational Linguistics*, vol. 23, no. 3, pp. 377–403, September 1997.

[13] R. Zens and H. Ney, "A comparative study on reordering constraints in statistical machine translation," in *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 2003.

[14] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "Galaxy-II: A reference architecture for conversational system development," in *Proc. of ICSLP 98*, November 1998.

[15] I. Melamed, "A Word-to-Word Model of Translational Equivalence," *Proc. of the ACL97*, pp. 490–497, 1997.