Incremental Decoding for Phrase-based Statistical Machine Translation

Baskaran Sankaran, Ajeet Grewal and Anoop Sarkar
School of Computing Science
Simon Fraser University
8888 University Drive
Burnaby BC. V5A 2Y1. Canada
{baskaran, asg10, anoop}@cs.sfu.ca

Abstract

In this paper we focus on the *incremental decoding* for a statistical phrase-based machine translation system. In incremental decoding, translations are generated incrementally for every word typed by a user, instead of waiting for the entire sentence as input. We introduce a novel modification to the beam-search decoding algorithm for phrase-based MT to address this issue, aimed at efficient computation of future costs and avoiding search errors. Our objective is to do a faster translation during incremental decoding without significant reduction in the translation quality.

1 Introduction

Statistical Machine Translation has matured significantly in the past decade and half, resulting in the proliferation of several web-based and commercial translation services. Most of these services work on sentence or document level, where a user enters a sentence or chooses a document for translation, which are then translated by the servers. Translation in such typical scenarios is still offline in the sense that the user input and translation happen sequentially without any interaction between the two phases.

In this paper we study decoding for SMT with the constraint that translations are to be generated incrementally for every word typed in by the user. Such a translation service can be used for language learning, where the user is fluent in the target language and experiments with many different source language sentences interactively, or in real-time translation environments such as speech-speech translation or translation during interactive chats.

We use a phrase-based decoder similar to Moses (Koehn et al., 2007) and propose novel modifications in the decoding algorithm to tackle incremental decoding. Our system maintains a partial decoder state at every stage and uses it while decoding for each newly added word. As the decoder has access only to the partial sentence at every stage, the future costs change with every additional word and this has to be taken into account while continuing from an existing partial decoder state. Another major issue is that as incremental decoding is provided new input one word at at time, some of the entries that were pruned out at an earlier decoder state might later turn out to better candidates resulting in search errors compared to decoding the entire sentence at once. It is to be noted that, the search error problem is related to the inability to compute full future cost in incremental decoding. Our proposed modifications address these twin challenges and allow for efficient incremental decoding.

2 Incremental Decoding

2.1 Beam Search for Phrase-based SMT

In this section we review the usual beam search decoder for phrase-based MT because we present our modifications for incremental decoding using the same notation. Beam search decoding for phrase-based SMT (Koehn, 2004) begins by collecting the translation options from the phrase table for all possible phrases of a given input sentence and precomputes the future cost for all possible contiguous sequences in the sentence. The pseudo-code for the usual beam-search decoding algorithm is illustrated in Algorithm 1.

The decoder creates n bins for storing hypotheses grouped by the number of source words covered. Starting from a null hypothesis in bin 0, the decoder iterates through bins 1 though n filling them with new hypotheses by extending the entries in the earlier bins.

A hypothesis contains the target words generated (e), the source positions translated so far (f) commonly known as *coverage set* and the score of the current translation (p) computed by the weighted log-linear combination of different feature functions. It also contains a back-pointer to

Proceedings of the Joint 5th Workshop on Statistical Machine Translation and MetricsMATR, pages 216–223,
Uppsala, Sweden, 15-16 July 2010.

Algorithm 1 Phrase-based Decoder pseudocode (Koehn, 2004)

Given: sentence S_n: $s_1 s_2 ... s_n$ of length n
Pre-compute future costs for all contiguous sequences
Initialize bins b_i where $i = 1 \ldots n$
Create initial hypothesis: $\{e : (), f : (), p : 1.0\}$
for $i = 1$ to n **do**
  for *hyp* $\in b_i$ **do**
    for *newHyp* that extends *hyp* **do**
      nf := num src words covered by *newHyp*
      Add *newHyp* to bin b_{nf}
      Prune bin b_{nf} using future costs
Find best hypothesis in b_n
Output best path that leads to best hypothesis

its parent hypothesis in the previous state and other information used for pruning and computing cost in later iterations.

As a new hypothesis is generated by extending an existing hypothesis with a new phrase pair, decoder updates the associated information such as coverage set, the target words generated, future cost (for translating rest of the source words) and its translation score. For example, consider Spanish to English translation: for the source sentence *Maria no daba una bofetada*, the hypothesis $\{e : (\text{Mary}), f : (1), p : 0.534\}$ which is the hypothesis that covers *Maria* can be extended to a new hypothesis $\{e : (\text{Mary, slap}), f : (1, 3, 4, 5), p : 0.043\}$ by choosing a new phrase pair *(daba una bofetada, slap)* covering the source phrases *Maria* and *daba una bofetada*. The probability score is obtained by weighted log-linear sum of the features of the phrases contained in the derivation so far.

An important aspect of beam search decoding is the pruning away of low-scoring hypotheses in each bin to reduce the search space and thus making the decoding faster. To do this effectively, beam search decoding uses the future cost of a hypothesis together with its current cost. The future cost is an estimate of the translation cost of the input words that are yet to be translated, and is typically pre-computed for all possible contiguous sequences in the input sentence before the decoding step. The future cost prevents the any hypotheses that are low-scoring, but potentially promising, from being pruned.

2.2 Incremental Decoder - Challenges

Our goal for the *incremental decoder* (ID) is to generate output translations incrementally for partial phrases as the source sentence is being input by the user. We assume *white-space* to be the word delimiter and the partial sentence is decoded for every encounter of the space character. We further assume the *return* key to mark *end-of-sentence* (EOS) and use it to compute language model score for the entire sentence.

As we noted above, future costs cannot be precomputed as in regular decoding because the complete input sentence is not known while decoding incrementally. Thus the incremental decoder can only use a partial future cost until the EOS is reached. The partial future cost could result in some of the potentially better candidates being pruned away in earlier stages. This leads to search errors and result in lower translation quality.

2.3 Approach

We use a modified beam search for incremental decoding (ID) and the two key modifications are aimed at addressing the issues of future cost and search errors. Beam search for ID begins with a single bin for the first word and more bins are added as the sentence is completed by the user. Our approach requires that the decoder states for the partial source sentence can be stored in a way that allows efficient retrieval. It also maintains a current decoder state, which includes all the bins and the hypotheses contained in them, all pertaining to the present sentence.

At each step ID goes through a pre-process phase, where it recomputes the partial future costs for all the spans accounting for the new word and updates the current decoder state with new partial future costs. It then generates new hypotheses into all the earlier bins and in the newly created using any new phrases (resulting from the new word added by the user) not used earlier.

Algorithm 2 shows the pseudocode of our incremental decoder. Given a partial sentence S_i[1] ID starts with the pre-process phase illustrated separately in algorithm 3. We use $P_{type}(l)$ to denote phrases of length l words and H_{type} to denote the set of hypotheses; in both cases *type* correspond to either *old* or *new*, indicating if it was not known in the previous decoding state or not.

[1] we use S_i and s_i to denote a i word partial sentence and i^{th} word in a (partial) sentence respectively

Algorithm 2 Incremental Decoder pseudocode

1: *Input*: (partial) sentence S_p: $s_1 s_2 ... s_{i-1} s_i$ with l_s words where s_i is the new word
2: *PreProcess*(S_p) (Algorithm 3)
3: **for** every bin b_j in $(1 \dots i)$ **do**
4: Update future cost and cover set $\forall$ H_{old}
5: Add any new phrase of length b_j (subject to d)
6: **for** bin b_k in $(b_{j-MaxPhrLen} \dots b_{j-1})$ **do**
7: Generate H_{new} for b_j by extending:
8: every H_{old} with every other $P_{new}(b_j - b_k)$
9: every H_{new} with every other $P_{any}(b_j - b_k)$
10: Prune bin b_j

Algorithm 3 PreProcess subroutine

1: *Input*: partial sentence S_p of length l_s
2: Retrieve partial decoder object for S_{p-1}
3: Identify possible P_{new} (subject to *MaxPhrLen*)
4: Recompute f_c for all spans in $1...l_s$
5: **for** every P_{new} in local phrase table **do**
6: Load translation options to table
7: **for** every P_{old} in local phrase table **do**
8: Update f_c with the recomputed cost

Given S_i, the pre-process phase extracts the new set of phrases (P_{new}) for the i^{th} word and adds them to the existing phrases (P_{old}). It then recomputes the future-cost (f_c) for all the contiguous sequences in the partial input and updates existing entries in the local copy of phrase table with new f_c.

In decoding phase, ID generates new hypotheses in two ways: i) by extending the existing hypotheses H_{old} in the previous decoder state S_{i-1} with new phrases P_{new} and ii) by generating new hypotheses H_{new} that are unknown in the previous state.

The main difference between incremental decoding and regular beam-search decoding is inside the two 'for' loops corresponding to lines $3-9$ in algorithm 2. In the outer loop each of the existing hypotheses are updated to reflect the recomputed f_c and coverage set. Any new phrases belonging to the current bin are also added to it[2].

[2]Based on our implementation of lazier cube pruning they are added to a priority queue, the contents of which are flushed into the bin at the end of inner for-loop and before the pruning step

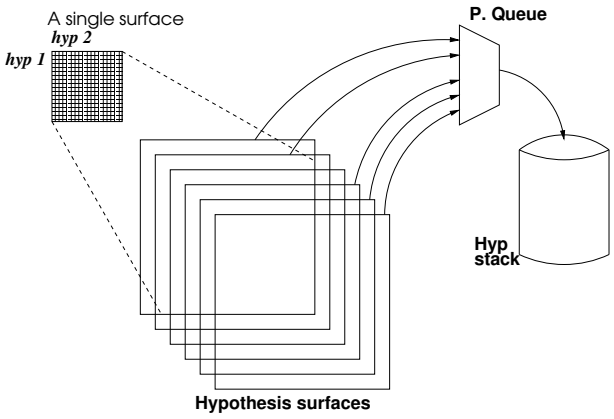

Figure 1: Illustration of Lazier Cube Pruning

The inner for-loop corresponds to the extension of hypotheses sets (grouped by same coverage set) to generate new hypotheses. Here a distinction is made between hypotheses H_{old} corresponding to previous decoder state S_{p-1} and hypotheses H_{new} resulting from the addition of word s_i. H_{old} is extended only using the newly found phrases P_{new}, whereas the newer hypotheses are processed as in regular beam-search.

2.4 Lazier Cube Pruning

We have adapted the pervasive lazy algorithm (or 'lazier cube pruning') proposed originally for Hiero-style systems by (Pust and Knight, 2009) for our phrase-based system. This step corresponds to the lines $5-9$ of algorithm 2 and allows us to only generate as many hypotheses as specified by the configurable parameters, beam size and beam threshold. Figure 1 illustrates the process of lazier cube pruning for a single bin.

At the highest level it uses a priority queue, which is populated by the different *hyper-edges* or *surfaces*[3], each corresponding to a *pair* of hypotheses that are being merged to create a new hypothesis. New hypotheses are generated iteratively, such that the hypothesis with the highest score is chosen in each iteration from among different hyper-edges bundles.

However, this will lead to search errors as have been observed earlier. Any hyper-edge that has been discarded due to poor score in an early stage might later become a better candidate. The problem worsens further when using smaller beam sizes (for interactive decoding in real-time settings, we even consider a beam size of 3). In

[3]Unlike Hiero-style systems, only two hypotheses are merged in a phrase-based system and hence the term *surface*

the next section, we introduce the idea of delayed pruning to reduce search errors.

3 Delayed Pruning

Delayed pruning (DP) in our decoder was inspired by the well known fable about the race between a *tortoise* and a *hare*. If the decoding is considered to be a race between competing candidate hypotheses with the winner being the best hypothesis for Viterbi decoding or among the top-n candidates for n-best decoding.[4]

In this analogy, a hypothesis having a poor score, might just be a tortoise having a slow start (due to a bad estimate of the true future cost for what the user intends to type in the future) as opposed to a high scoring hare in the same state. Pruning such hypotheses early on is not risk-free and might result in search errors. We hypothesize that, given enough chance it might improve its score and move ahead of a hare in terms of translation score.

We implement DP by relaxing the lazier cube pruning step to generate a small, fixed number of hypotheses for coverage sets that are not represented in the priority queue and place them in the bin. These hypotheses are distinct from the usual top-k derivations. Thus, the resulting bin will have entries from all possible hyper-edge bundles. Though this reduces the search error problem, it leads to increasing number of possibilities to be explored at later stages with vast majority of them being *worse* hypotheses that should be pruned away.

We use a two level strategy of *delay* and then *prune*, to avoid such exponentially increasing search space and at the same time to reduce search error. At the delay level, the idea is to delay the pruning for few promising tortoises, instead of retaining a fixed number of hypotheses from all unrepresented hyper-edges. We use the normalized language model scores of the top-hypotheses in each hyper-edge that is not represented in cube pruning and based on a threshold (which is obtained using a development test set), we selectively choose few hyper-edge bundles and generate a small number (typically 1-3) of hypotheses from each of them and flag them as *tortoises*. These tortoises are extended minimally at each iteration subject to their normalized LM score.

While this significantly reduces the total number of hypotheses at initial bins, many of these tortoises might not show improvement even after several bins. Thus at the prune level, we prune out tortoises that does not improve beyond a threshold number of bins called *race course* limit. The race course limit signifies the number of steps a tortoise has in order to get into the decoder beam.

When a tortoise improves in score and breaks into the beam during cube pruning, it is deflagged as a tortoise and enters the regular decoding stream. We found DP to be effective in reducing the search error for incremental decoder in our experiments.

4 Evaluation and Discussion

The evaluation was performed using our own implementation of the beam-search decoding algorithms. The architecture of our system is similar to Moses, which we also use for training and for minimum error rate training (MERT) of the log-linear model for translation (Och, 2003; Koehn et al., 2007). Our features include 7 standard phrase-based features: 4 translation model features, i.e. $p(f|e)$, $p(e|f)$, $p_{lex}(f|e)$ and $p_{lex}(e|f)$, where e and f are target and source phrases respectively; features for phrase penalty, word penalty and language model, and we do not include the reordering feature. We used Giza++ and Moses respectively for aligning the sentences and training the system. The decoder was written in Java and includes *cube pruning* (Huang and Chiang, 2007) and *lazier cube pruning* (Pust and Knight, 2009) functionalities as part of the decoder. Our decoder supports both regular beam search (similar to Moses) and incremental decoding.

In our experiments we experimented various approaches for storing partial decoder states including memcache and transactional persistence using JDBM but found that the serialization and deserialization of decoder objects directly into and from the memory to work better in terms of speed and memory requirements. The partial object is retrieved and deserialized from the memory when required by the incremental decoder.

We evaluated the incremental decoder for translations between French and English (in both directions). We used the Workshop on Machine Translation shared task (WMT07) dataset for training,

[4]The analogy is used to compare two or more hypotheses in terms of their translation scores and *not* speed. Though our objective is faster incremental decoding, we use the analogy here to compare the scores.

optimizing and testing. The system was trained using Moses and the feature weights were optimized using MERT. To benchmark our Java decoder, we compare it with Moses by running it in regular beam search mode. The Moses systems were also optimized separately on the WMT07 devsets.

Apart from comparing our decoder with Moses in regular beam search, we also compared the incremental decoding with regular regular beam using our decoder. To make it comparable with incremental decoding, we used the regular beam search to *re-decode* the sentence fragments for every additional word in the input sentence. We measured the following parameters in our empirical analysis: translation quality (as measured by BLEU (Papineni et al., 2002) and TER (Snover et al., 2006)), search errors and translation speed. Finally, we also measured the effect of different race course limits on BLEU and decoding speed for incremental decoding.

4.1 Benchmarking our decoder

In this section we compare our decoder with Moses for regular beam search decoding. Table 1 gives the BLEU and TER for the two language pairs. Our decoder implementation compares favourably with Moses for Fr-En: the slightly better BLEU and TER for our decoder in Fr-En is possibly due to the minor differences in the configuration settings. For En-Fr translation, Moses performs better in both metrics. There are differences in the beam size between the two decoders, in our system the beam size is set to 100 compared to the default value of 1000 (the cube pruning pop limit) in Moses; we are planning to explore this and remove any other differences between them. However based on our understanding of the Moses implementation and our experiments, we believe our decoder to be comparable in accuracy with the Moses implementation. The numbers in the boldface are statistically significant at 95% confidence interval.

4.2 Re-decoding v.s. Incremental decoding

We test our hypothesis that incremental decoding can benefit by using partial decoder states for decoding every additional word in the input sentence. In order to do this, we run our incremental decoder in both regular beam search mode and in incremental decoding mode. In regular beam search mode, we forced the beam search decoder to re-decode the sentence fragments for every additional word and in incremental decoding mode, we used the partial decoding states to incrementally decode lastly added word. We then compare the BLEU and TER scores between them to validate our hypothesis.

We further test effectiveness of delayed pruning (DP) in incremental decoding by comparing it to the case where we turn off the DP. For incremental decoding, we set the beam size and the race course limit (for DP) to be 3. Additionally, we used a threshold of -2.0 (in log-scale) for normalized LM in the delay phase of DP, which was obtained by testing on a separate development test set.

We would like to highlight two observations from the results in Table 2. First the regular beam search indicate possible search errors due to the small beam size (cube pruning pop limit) and the BLEU scores has decreased by 0.56 for Fr-En and by over 2.5 for En-Fr, than the scores corresponding to a beam size of 100 shown in Table 1. Secondly, we find the incremental decoding to perform better for the same beam size. However, incremental decoding without delay pruning still seems to incur search errors when compared with the regular decoding with a larger beam. Delayed pruning alleviates this issue and improves the BLEU and TER significantly. This we believe, is mainly because the strategy to delay the pruning retains the potentially better partial hypotheses for every coverage set. It should be noted that results in Table 2 pertain only to our decoder implementation and not with Moses.

We now give a comparative note between our approach and the pruning strategy in regular beam search. Delaying the hypothesis pruning is the important aspect in our approach to incremental decoding. In the case of regular beam search, the hypotheses are pruned when they fall out of the beam and the idea is to have a larger beam size to avoid the early pruning of potentially good candidates. With the advent of cube pruning (Huang and Chiang, 2007), the 'cube pruning pop limit' (in Moses) determines the number of hypotheses retained in each stack. In both the cases, it is possible that some of the coverage sets go unrepresented in the stack due to poor candidate scores. This is not desirable in the incremental decoding setting as this might lead to search errors while decoding a partial sentence.

Additionally, Moses offers an option (cube

Decoder	Fr-En		En-Fr	
	BLEU	**TER**	**BLEU**	**TER**
Moses	**26.98**	0.551	**27.24**	0.610
Our decoder	**27.53**	0.541	26.96	0.657

Table 1: Regular beam search: Moses v.s. Our decoder

Decoder	Fr-En		En-Fr	
	BLEU	**TER**	**BLEU**	**TER**
Re-decode w/ beam search	26.96	0.548	24.33	0.635
ID w/o delay pruning	27.01	0.547	25.00	0.618
ID w/ delay pruning	**27.62**	0.545	**25.45**	0.616

Table 2: BLEU and TER: Re-decoding v.s. Incremental Decoding (ID)

pruning diversity) to control the number of hypotheses generated for each coverage set (though set to '0' by default). It might be possible to use this in conjunction with cube pruning pop limit as an alternative to our delayed pruning in the incremental decoding setting (with the risk of combinatorial explosion in the search space).

In contrast, the delayed pruning not only avoids search errors but also provides a dynamically manageable search space (refer section 4.2.2) by retaining the best of the potential candidates. In a practical scenario like real-time translation of internet chat, translation speed is an important consideration. Furthermore, it is better to avoid large number of candidates and generate only few best ones, as only the top few translations will be used by the system. Thus we believe our delayed pruning approach to be a principled pruning strategy that combines the different factors in an elegant framework.

4.2.1 Search Errors

As BLEU only indirectly indicates the number of search errors made by algorithm, we used a more direct way of quantifying the search errors incurred by the ID in comparison to regular beam search. We define the search error to be the difference between the translation scores of the best hypotheses produced by the ID and the regular beam search and then compute the mean squared error (MSE) for the entire test set. We use this method to compare ID in the two settings of delayed pruning being turned off (using a smaller beam size of 3 to simulate the requirements of near instantaneous translations in real-time environments) and delayed pruning turned on. We compare the model score in these cases with the model score for the best result obtained from the regular beam search decoder (using a larger beam of size 100).

Direction	Beam search against Incremental Decoding	
	w/o DP	*w/ DP*
Fr-En	0.3823	0.3235
En-Fr	1.1559	0.6755

Table 3: Search Errors in Incremental Decoding

The results are shown in Table 3 and as can be clearly seen, ID shows much lesser mean square error with the DP turned on than when it is turned off. Together the BLEU and TER numbers and the mean square search error show that delayed pruning is useful in the incremental decoding setting. Comparing the En-Fr and Fr-En results show that the two language pairs show slightly different characteristics but the experiments in both directions support our overall conclusions.

4.2.2 Speed

In this experiment, we set out to evaluate the ID against the regular beam-search in which sentence fragments are incrementally decoded for additional words. In order compare with the incremental decoder, we modified the regular decoder to decode the partial phrases, so that it *redecodes* the partial phrase from the scratch instead of reusing the earlier state.

We ran the timing experiments on a Dell machine with an Intel Core i7 processor and 12 GB memory, clocking 2.67 GHz and running Linux (CentOS 5.3). We measured the time taken for decoding the fragment with every word added and

averaged it first over the sentence and then the entire test set. The average time (in msecs) includes the future cost computation for both. We also measured the average number of hypotheses for every bin at the end of decoding a complete sentence, which was also averaged over the test set.

The results in Table 4 show that the incremental decoder was significantly faster than the beam search in re-decoding mode almost by a factor of 9 in the best case (for Fr-En). The speedup is primarily due to two factors, i) computing the future cost for the new phrases as opposed to computing it for all the phrases and ii) using partial decoder states without having to re-generate hypotheses through the cube pruning step and the latencies associated with computing LM scores for them. The addition of delayed pruning slowed down the speed at most by 7 msecs (for En-Fr). In addition, delayed pruning can be seen generating far more hypotheses than the other two cases. Clearly, this is because of the delay in pruning the tortoises until the race course limit. Even with such significantly large number of hypotheses being retained for every bin, DP results in improved speed (over re-decoding from scratch) and better performance by avoiding search errors (compared to the incremental decoder that does not use DP).

4.3 Effect of Race course limit

Table 5 shows the effect of different race course limits on translation quality measured using BLEU. We generally expect the race course limit to behave similar to the beam size as they both allow more hypotheses in the bin thereby reducing search error although at the expense of increasing decoding time.

However, in our experiments for Fr-En, we did not find significant variations in BLEU for different race course limits. This could be due to the absence of long distance re-orderings between English and French and that the smallest race course limit of 3 is sufficient for capturing all cases of local re-ordering. As expected, we find the decoding speed to slightly decrease and the average number of hypotheses per bin to increase with the increasing race course limit.

5 Related Work

Google[5] does seem to perform incremental decoding, but the underlying algorithms are not public knowledge. They may be simply re-translating the input each time using a fast decoder or re-using prior decoder states as we do here.

Intereactive translation using text prediction strategies have been studied well (Foster et al., 1997; Foster et al., 2002; Och et al., 2003). They all attempt to interactively help the human user in the *postediting* process, by suggesting completion of the word/phrase based on the user accepted prefix and the source sentece. Incremental feedback is part of *Caitra* (Koehn, 2009) an interactive tool for human-aided MT and works on a similar setting to interactive MT. In *Caitra*, the source text is pre-translated first and during the interactions it dynamically generates user suggestions.

Our incremental decoder work differs from these text prediction based approaches, in the sense that the input text is not available to the decoder beforehand and the decoding is being done dynamically for every source word as opposed to generating suggestions dynamically for completing target sentece.

6 Conclusion and Future Work

We presented a modified beam search algorithm for an efficient *incremental decoder* (ID), which will allow translations to be generated incrementally for every word typed by a user, instead of waiting for the entire sentence as input by reusing the partial decoder state. Our proposed modifications help us to efficiently compute partial future costs in the incremental setting. We introduced the notion of *delayed pruning* (DP) to avoid search errors in incremental decoding. We showed that reusing the partial decoder states is faster than re-decoding the input from the scratch every time a new word is typed by the user. Our exhaustive experiments further demonstrated DP to be highly effective in avoiding search errors under the incremental decoding setting. In our experiments in this paper we used a very tight beam size; in future work, we would like to explore the tradeoff between speed, accuracy and the utility of delayed pruning by varying the beam size in our experiments.

[5] *translate.google.com*

References

George Foster, Pierre Isabelle, and Pierre Plamondon. 1997. Target-text mediated interactive machine translation. *Machine Translation*, 12(1/2):175–194.

Decoder	Fr-En		En-Fr	
	Avg time	*Avg Hyp/ bin*	*Avg time*	*Avg Hyp/ bin*
Re-decode	724.46	2.21	130.29	2.32
ID w/o DP	84.85	2.89	27.58	2.89
ID w/ DP	87.01	85.11	34.35	60.46

Table 4: Speed: Re-decoding v.s. Incremental Decoding (ID)

Race Course Limit	Fr-En			En-Fr		
	BLEU	*Avg time*	*Avg Hyp/ bin*	*BLEU*	*Avg time*	*Avg Hyp/ bin*
3	26.75	87.83	85.11	25.39	36.15	75.03
4	26.77	91.14	86.35	25.37	36.21	77.69
5	26.77	90.81	86.52	25.37	36.25	78.47
6	26.77	95.91	86.56	25.37	37.34	78.71
7	26.77	91.67	86.57	25.37	36.26	78.81

Table 5: Effect of different race course limits

George Foster, Philippe Langlais, and Guy Lapalme. 2002. User-friendly text prediction for translators. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 148–155, Morristown, NJ, USA. Association for Computational Linguistics.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic, June. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June. Association for Computational Linguistics.

Philipp Koehn. 2004. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In Robert E. Frederking and Kathryn Taylor, editors, *AMTA*, volume 3265 of *Lecture Notes in Computer Science*, pages 115–124. Springer.

Philipp Koehn. 2009. A web-based interactive computer aided translation tool. In *In Proceedings of ACL-IJCNLP 2009: Software Demonstrations*, Suntec, Singapore, August.

Franz Josef Och, Richard Zens, and Hermann Ney. 2003. Efficient search for interactive statistical machine translation. In *EACL '03: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics*, pages 387–393, Morristown, NJ, USA. Association for Computational Linguistics.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan, July. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wie-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. ACL*.

Michael Pust and Kevin Knight. 2009. Faster mt decoding through pervasive laziness. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 141–144, Boulder, Colorado, June. Association for Computational Linguistics.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas: AMTA 2006*.