MOOD: A Modular Object-Oriented Decoder for Statistical Machine Translation

Alexandre Patry, Fabrizio Gotti and Philippe Langlais

RALI/DIRO
Université de Montréal
Succursale Centre-Ville
H3C 3J7 Montréal, Canada
{patryale,gottif,felipe}@iro.umontreal.ca

Abstract

We present an Open Source framework called MOOD developed in order to facilitate the development of a Statistical Machine Translation Decoder. MOOD has been modularized using an object-oriented approach which makes it especially suitable for the fast development of state-of-the-art decoders. As a proof of concept, a clone of the PHARAOH decoder has been implemented and evaluated. This clone named RAMSES is part of the current distribution of MOOD.

1. Introduction

Ever since the pioneering work of IBM researchers (Brown et al., 1993), the number of Statistical Machine Translation (SMT) practitioners have constantly increased. One reason for this lies in the invaluable toolkits that are available within our community. As a matter of fact, training an SMT engine from a bitext is (more or less) a matter of gluing together dedicated softwares. Word-based models, or the so-called IBM models, can be trained using the GIZA or GIZA++ toolkits (Och and Ney, 2000). One can then train phrase-based models using the THOT toolkit (Ortiz-Martínez et al., 2005). For their part, language models currently in use in SMT systems can be trained using packages such as SRILM (Stolcke, 2002) and the CMU-SLM toolkit (Clarkson and Rosenfeld, 1997).

Last but not least, several available SMT decoders can exploit trained models in order to perform translation. Most notably, the REWRITE decoder (Germann et al., 2001) recognizes an IBM model 4, while the PHARAOH decoder is especially handy for phrase-based SMT (Koehn et al., 2003). Such decoders are invaluable toolkits. For instance, they can serve as baseline or reference systems. Unfortunately, one major drawback of these two decoders is their licensing policy. Indeed, they are both available for non-commercial use in their binary form only. This severely limits their use in practice (Walker, 2005).

In this paper, we describe our efforts in designing a generic architecture called MOOD (Modular Object-Oriented Decoder) especially suited for instantiating specific SMT decoders. The goal of MOOD is thus two-fold: offering Open Source, state-of-the-art decoders and providing an architecture to easily build these decoders.

We recall the role of the decoder in SMT in Section 2. In Section 3, we describe the main architecture of MOOD. We then show in Section 4 how we used MOOD to instantiate RAMSES, a clone of PHARAOH. We evaluate RAMSES against PHARAOH in Section 5. Finally, we conclude this work in Section 6.

2. Decoding in SMT

The problem we are concerned with in this study is the maximization (argmax) — also called decoding — that arises in SMT and which is formulated in the following equation:

$$translation(s) = \underset{t \in \mathcal{T}}{\operatorname{argmax}} P(t|s) \quad (1)$$

where s is the sentence to translate, t is a possible translation and $\mathcal{T}$ is a set containing all the sentences of the language of t.
Enumerating all the valid sentences in the target language, as Equation 1 suggests is impracticable. Therefore, the target sentence is usually built incrementally by transforming partial translations using an algorithm that can be outlined as:

1. initialize the set of candidates $\mathcal{H}$ to a blank translation
2. select an incomplete candidate translation h from $\mathcal{H}$
3. for each transformation tr that applies to h
   (a) set h_{copy} to the result of applying tr on h
   (b) if h_{copy} is promising, add it to $\mathcal{H}$
   (c) if $\mathcal{H}$ contains incomplete candidates, go to 2
4. return the best candidate in $\mathcal{H}$

From this outline, we see that a decoder must resolve two independent problems: representing the model and exploring the space of possible translations efficiently. The model has to do with expressiveness. It defines how (complete or incomplete) translations and transformations are represented and evaluated. The first popular SMT models were word-based (Brown et al., 1993) and had a hard time translating idioms and the like. Later came the phrase-based models (PBM), which deal with word sequences (Marcu and Wong, 2002; Koehn et al., 2003). Even though PBMs give state-of-the-art performances (Och and Ney, 2004), they still struggle with global reorderings that must be dealt with when translating between languages that have different syntactical structures (SOV versus SVO languages).

One research strategy to solve this problem is to use syntactic knowledge and work with parse trees instead of word sequences (*e.g.* see (Yamada and Knight, 2001; Quirk et al., 2005)).
The search space exploration strategy is a compromise between the decoder's speed and the quality of the translations produced. The search method decides which incomplete translations are promising, the order in which they are explored and when a solution is good enough to stop searching. The problem is complex; Knight (1999) has shown that the search space exploration is NP-complete for word-based models.
Many different decoders have been implemented for different models. An overview of some decoders available for word-based models can be found in (Tillmann and Ney, 2003). Several decoders have been published for phrase-based models (see for instance (Koehn, 2004)).

3. The MOOD Framework

A decoder must implement a specific combination of a model representation and a search space exploration strategy. In this section, we present a framework that applies to any such combination. To do so, the MOOD framework clearly separates the model and the search space exploration strategy. Algorithm 1 presents a functional view of how a typical decoder is implemented within MOOD.

Algorithm 1 A typical decoder as implemented using the MOOD framework. The algorithm takes as input h_{start}, the starting hypothesis and *gen*, a transformation generator.

```
1: H_active ← {h_start}
2: H_complete ← ∅
3: while H_active is not empty do
4:    retrieve a hypothesis from H_active into h
5:    if h.value() is high enough then
6:       Tr ← gen.find(h.partialTranslation())
7:       for all tr ∈ Tr do
8:          h_copy ← h.clone()
9:          h_copy.apply(tr)
10:         if h_copy.isCompleted() then
11:            add h_copy to H_complete
12:         else
13:            add h_copy to H_active
14: return H_complete
```

The key idea behind Algorithm 1 is to manipulate a collection of partial translations that will be transformed until they are completed or until no more transformation can be applied to them.

3.1. Model representation

A model is implemented using four concepts : partial translations, transformations, cost functions and transformation generators. The dependencies between those concepts are presented in Figure 1.
A *partial translation* represents a translation that is being built. It consists of a 3-tuple of the form $\langle s,t,p\rangle$ where s is the sentence to translate, t is a partial translation of s and p is a progress indicator describing how t can be extended.

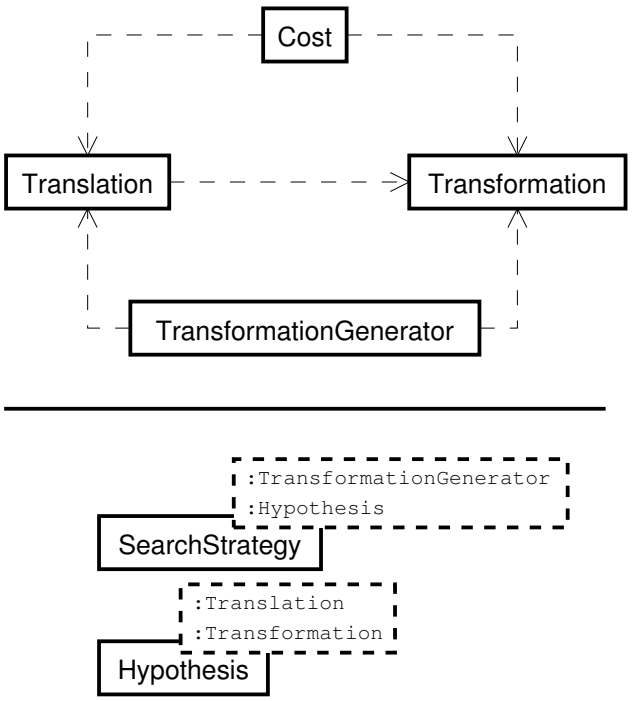

Figure 1: The different components of the MOOD framework. Arrows indicate dependencies and dashed boxes template parameters.

Incomplete translations are stored in a collection named $\mathcal{H}_{active}$ and complete translations are stored in $\mathcal{H}_{complete}$.
At each step, a *transformation* is applied to an incomplete partial translation (line 9). This transformation will modify the target sentence and the progress indicator. Usually, the parameters of a decoder are rules like *the translation of "wonderful" is "merveilleux" with a probability of 0.8*. A transformation is such a rule, but instantiated on a specific partial translation, like *the translation of "wonderful" is the i^{th} word of the source with a probability of 0.8*. As shown in Figure 1, the implementation of partial translations depends on the implementation of the transformations. This is so because a partial translation must know how to apply a transformation on itself.
The quality of a partial translation must be quantified in order to prune $\mathcal{H}_{active}$ (line 5) and to select the best translation in $\mathcal{H}_{complete}$. This quantification is carried out by a *cost* object, which associates a numerical value to a partial translation. Each time a partial translation is transformed, its cost must be updated. At the very least, the cost usually reflects the target sentence fluency, the probability of the transformations applied as well as the word reordering between the source and the target sentences.
The exploration algorithm does not need to know about the inner workings of the model, but it must be able to retrieve the transformations that are applicable to a partial translation (line 6). The *transformation generator* serves this purpose; it offers a *find* function that takes as input a partial translation and returns a set of applicable transformations.

3.2. Search space exploration

A search space exploration method is specified using two independent concepts: a hypothesis and a search strategy.

The principal role of a *hypothesis* is to synchronize a partial translation with its cost, but it is not limited to that. It can, for example, build a search graph that can be used afterward to extract *n*-best translations (Ueffing et al., 2002) or it can also compute statistics on the exploration. A hypothesis able to build the search graph of any combination of model and search strategy is available in MOOD.

All the concepts described so far are assembled in a *search strategy* to form a decoder. With MOOD, a search strategy is an object that offers a *decode* method that takes as a parameter the initial hypothesis and a transformation generator. The responsibility of the search strategy is the management of $\mathcal{H}_{active}$ and $\mathcal{H}_{complete}$, that is, to determine the order in which the hypotheses are explored and to establish a pruning policy (lines 4 and 5).

As shown in Figure 1, the hypothesis and the search strategy are independent of the model because they are templatized. They can thus be reused with any other model implementing the concepts described in the previous section.

The current distribution of MOOD implements a multi-stack beam search strategy (see Section 4).

3.3. Object-oriented architecture at work

One of the key advantage of object-oriented architectures is an improved reusability of the different modules of the program. In order to illustrate this point, let us take a look at the problems an engineer faces when adapting an existing decoder to a new problem.

The first modification that must often be made during the lifetime of a decoder is changing the cost function. Because with MOOD the different functionalities have been separated, the task only consists of passing a different cost object when creating the starting hypotheses. No existing code needs to be changed or duplicated. It is noteworthy that the cost function can be anything. It can even encapsulate an estimation of the future cost of a translation.

Another useful modification of an existing decoder consists in adding new search strategies. Within MOOD, the inner workings of the search algorithm is cleanly disjoint from that of the other modules, so one only needs to write a new search strategy class that defines a *decode* function that takes as a parameter a starting hypothesis and a transformation generator. The search strategy does not have to be aware of the implementation of the hypotheses or of the transformations, thus it can be used with any model (word-based, phrase-based, hierarchical, etc.).

Creating a decoder for a new model representation demands more effort. Handling a new model representation means that new partial translation and transformation classes must be created. Because the cost and the transformation generator depend on the translations and the transformations representation, new classes for these modules will have to be created as well. Currently, MOOD only supports phrase-based models, but other models will soon be supported.

3.4. Practical considerations

In practice, many different transformation sequences can lead to equivalent hypotheses; when this is the case, we say that the hypotheses are in the same state. When two or more hypotheses are in the same state, we can safely ignore them all but the one having the highest cost value. To handle this strategy, MOOD expects each partial translation, cost and hypothesis to provide a *compareState* function that imposes a total ordering on their state.

3.5. Implementation

MOOD is implemented with the C++ programming language and is licensed under the Gnu General Public License (GPL).[1] This license grants the right to anybody to use, modify and distribute the program and its source code. The only restriction is that if a modified version is distributed, it must also be licensed under the GPL. As explained in (Walker, 2005), this kind of license stimulates new ideas and research. MOOD is currently available at `http://smtmood.sourceforge.net`.

It is our hope that the availability of the source code and the clean design of MOOD will make it a useful platform to implement and distribute new decoders.

4. Creating a phrase-based decoder

As a proof of concept that our framework is viable, we reproduced the most popular phrase-based decoder: PHARAOH (Koehn, 2004). In this section, we describe how we implemented RAMSES, our clone, based on the comprehensive user manual of PHARAOH. We followed this manual as faithfully as possible; the command-line syntax RAMSES recognizes mimics that of PHARAOH. The output produced by both decoders are compatible and RAMSES can also output its *n*-best lists in the same format as PHARAOH does, that is, a format that the CARMEL toolkit can parse (Knight and Al-Onaizan, 1999). Therefore, switching from one decoder to another should be easy.

4.1. Model representation

When a PBM decoder is launched, it takes as parameters a set of weighted rules and a language model. Each weighted rule consists of a sequence of source words, a sequence of target words and a probability. Before a source sentence is translated, all the rules that can apply to it are indexed and stored as *transformations*, which are tuples containing a rule and the position where it applies in the source.

As we already said in Section 3.1, a *partial translation* corresponds to a source sentence, a target sentence and a progress indicator. The source and the target sentences are sequences of words. The progress indicator contains a mask indicating which source words have been translated and the position of the word after the last translated one (needed by the transformation generator). When a transformation is applied, the target words of its rule are appended at the end of the target of the partial translation and the progress indicator is updated. An example of how a transformation is applied to a partial transformation can be found in Figure 2.

When a transformation is applied to a partial translation, the hypothesis ensures that its *cost* is updated. Each partial translation's cost is implemented using the following costs:

Language model The log-probability of the target created (evaluated by a trigram model).

[1] `http://www.gnu.org/copyleft/gpl.html`

Initial partial translation	
source	*quel monde merveilleux*
progress	`000`, next=1
target	
Transformation 1	
rule	*quel* → *what a* with probability 0.3
position	1
Partial translation 1	
source	*quel monde merveilleux*
progress	`100`, next=2
target	*what a*
Transformation 2	
rule	*merveilleux* → *wonderful* with probability 0.8
position	3
Partial translation 2	
source	*quel monde merveilleux*
progress	`101`, next=4
target	*what a wonderful*
Transformation 3	
rule	"monde" → "world" with probability 0.6
position	2
Completed translation	
source	*quel monde merveilleux*
progress	`111`, next=3
target	*what a wonderful world*

Figure 2: A possible transformation sequence that translates a French sentence into an English sentence using a PBM model.

Translation table The sum of the log-probabilities of the rules that have been applied so far.

Distortion The number of source words that were skipped between two consecutive rules. For example, in Figure 2, transformation 2 incurs a distortion cost of 1 because the French word *monde* is skipped.

Word penalty The number of words in the target sentence.

Heuristic An estimation of the cost to complete the partial translation. This estimation is the sum of the costs of translating each contiguous untranslated sequence as if it were a stand-alone sentence.

All these costs are weighted and combined in an exponential:

$$cost(h) = \exp \sum_{c \in \mathcal{C}} w_c value_c \qquad (2)$$

where h is the partial translation to evaluate, $\mathcal{C}$ contains the costs to consider, $value_c$ is the value of a cost and w_c is its weight. Even though equation 2 is a mix of other costs, it can be implemented in a cost object; in MOOD a cost can be arbitrarily complex.

The only remaining part of the model representation is the *transformation generator*. It consults the indexed transformations and the progress indicator to return the transformations that only translate untranslated words. Like PHARAOH, it is possible to restrict the transformations returned to the ones for which the starting position is not more than *dl* words away from the position right after the last translated word, where *dl* is a distortion limit.

4.2. Search space exploration

The last building block of the decoder is the *search strategy*. Like in PHARAOH, we implemented a beam search strategy. Instead of treating all the hypotheses together, a beam search proceeds level by level. At each level, the hypotheses are pruned regardless of the hypotheses in the other levels. The ones that are kept are expanded and stored for treatment in a further level. The advantage of using different levels is that we can group and prune together comparable hypotheses.

In PHARAOH and RAMSES, the level of a hypothesis corresponds to the number of source words that are translated in its partial translation. This is so because without a very good heuristic, it is difficult to compare a hypothesis where one word is translated with another where, for example, twenty words are translated.

The way it is designed, this search strategy can therefore be used by any model, as long as the latter provides a policy to establish the level of any given hypothesis.

5. Evaluation

We evaluated whether the MOOD framework can be use to develop a state-of-the-art decoder. We thus compared RAMSES against PHARAOH.

Because there are some places in the PHARAOH manual that left us with implementation choices, we ended up with a clone which differs slightly from the original. To measure this difference, we evaluated the translation quality of the two decoders.

A good design is often a compromise between modularity, which increases code reuse, and execution speed. To measure this compromise, we also compared the speed of the two decoders.

5.1. Corpora

We used the parallel corpora that were prepared for the shared task of the WPT'05 ACL workshop (Koehn and Monz, 2005). They have been extracted from the proceedings of the European Parliament. We focussed our attention on two different language pairs: *French-to-English* and *German-to-English*. If the former is a well studied language pair, the latter presents more challenge to a phrase-based decoder, because German has a substantially different word order than English has (Collins et al., 2005).

We trained (on the TRAIN section of the corpora) the phrase-based models with one of our programs relying on a bidirectional word alignment produced by GIZA++ (Och and Ney, 2000). Each PBM parameter was scored with its relative frequency as well as with an IBM1-like score ($P(s|t)$ and $P(t|s)$). The only preprocessing we applied consisted in putting all the texts in lowercase.

The two decoders were tuned (on the DEV sections) to optimize the BLEU metric using the algorithm described in (Och, 2003). We used the implementation available in the training toolkit graciously made available by the WPT'06 shared task organizers (see `http://www.statmt.org/wmt06/shared-task/baseline.html`).
The main characteristics of the corpus and the models we used in this study are reported in Table 1. Note that although the TEST sections we used here correspond to the test material that the participating systems had to translate, a direct comparison between our results and the ones reported during the workshop is not possible since, for some reason, we did not use the same evaluation toolkits (see next section).

pair	corpus	sentences	parameters
fr2en	TRAIN	688 031	13 190 285
	DEV	200	203 760
	TEST	2 000	663 042
de2en	TRAIN	751 088	11 852 331
	DEV	200	116 095
	TEST	2 000	415 507

Table 1: Main characteristics of the corpora and models we used in this study. `parameters` indicates the number of parameters of the PBM that match the source material.

5.2. Metrics

We used the BLEU (Papineni et al., 2002) and NIST (Doddington, 2002) metrics to automatically rate the translation quality of the engines we compared, both computed by the script `mteval`. We used the version `v11a` of the script which we downloaded from `http://www.nist.gov/speech/tests/mt/resources/scoring.htm`.
To compare the decoding speed of the two decoders, we decided to compute the number of states they explored per second. It is indeed a better indicator of the potential of the decoder than the total number of states or the total running time, because those are mainly affected by pruning and tuning. Different tunings yield different scores, thus affecting the number of hypotheses that are pruned. We also report the total number of states considered during each translation session. Note that because each decoder was tuned separately (on the same DEV sections with the same tool), they eventually ended up with a different scoring function, thus affecting the number of hypotheses that were pruned.

5.3. Results

The tests were conducted on computers equipped with a 2 GHz AMD Opteron 246 processor and 8 gigabytes of RAM. The results of the French-to-English experiment are reported in Table 2. Expectedly, the translation quality of the two decoders is comparable. We observe that RAMSES explores less states per second than PHARAOH. However, for this setting, RAMSES explored 20% more hypotheses than PHARAOH.
The performance of RAMSES and PHARAOH for the German-to-English task are reported in Table 3. As we already mentioned, German is a lot more difficult to translate than French for a PBM decoder (Collins et al., 2005), and this results into lower scores. For BLEU, PHARAOH ended up with a significantly higher score, even if RAMSES explored 40% more hypotheses than PHARAOH for this setting. The fact that we tuned both decoders on a rather small corpus (200 sentences) might explain the difference in BLEU scores we observe. However, note that the difference in NIST scores is in favor of RAMSES.

	BLEU	NIST	states / sec	states
RAMSES	0.2908	7.2672	375 708	1.39E+10
PHARAOH	0.2897	7.2969	587 704	1.16E+10

Table 2: Translation quality metrics and decoding speeds when translating from French to English.

	BLEU	NIST	states / sec	states
RAMSES	0.1994	6.4659	410 363	8.16E+9
PHARAOH	0.2262	6.4361	660 743	5.78E+9

Table 3: Translation quality metrics and decoding speeds when translating from German to English.

When analyzing these results, we must keep in mind that RAMSES was designed for two different reasons. First, to offer the community an Open Source, phrase-based decoder and second, to assess the usefulness of MOOD when creating state-of-the-art decoders.
There is no doubt that the first goal was reached. By releasing MOOD and RAMSES source codes, new SMT practitioners do not need to start from scratch anymore when they wish to test a new idea for the decoding process. Another advantage of Open Source software is that since more people have the chance to read the source code, they are more likely to detect bugs and improve the code.
We also consider that the second goal was reached. We matched PHARAOH's translation quality when translating from French to English, and we further think that the differences observed in the results when translating from German are due to implementation details or maybe to overfitting during tuning.
Currently, PHARAOH is roughly twice as fast as RAMSES. If only the features of PHARAOH are needed, then PHARAOH should be preferred to RAMSES, but for researchers who want to experiment, RAMSES is a solid contender.

6. Conclusion

Our main contribution has been to provide a general framework in order to implement a decoder. Our C++ implementation of this framework is called MOOD and it is released under an Open Source license.[2] Anyone can thus view, use and modify its source code freely. As a proof of concept that this framework can be used to build a full-fledged decoder, we implemented RAMSES, a PBM decoder based on PHARAOH's manual.

[2] `http://smtmood.sourceforge.net`

In its architecture, MOOD cleanly separates the model from the search space exploration strategy. It thus maximizes code reusability by allowing the same exploration strategy to be used with many different models. Code reusability has two major advantages over a complete rewrite: it is faster to develop a new decoder and reduces the risk of introducing bugs.

Even though RAMSES is not as fast as PHARAOH, it is a solid tool for research projects: the code is Open Source and the architecture is modular, making it easier for researchers to experiment with SMT. Also, the command-line interface of RAMSES is very close to that of PHARAOH, so switching from one to the other should be easy.

Currently, MOOD only supports PBM and the beam search strategy. We plan to support at least word-based models and hierarchical models. We will also add a greedy and a dynamic programming search space exploration strategy.

Often, SMT is said to be language independant, but sometimes, specific problems need specific solutions. We therefore intend to add to MOOD specialized modules for some languages. For instance, we would like to create a transformation generator that returns transformations that respect the structure of a sentence as well as specialized costs. Because MOOD is modular, these modules will be easy to switch on or off, so the overall decoder will not lose its generality.

7. References

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Philip Clarkson and Ronald Rosenfeld. 1997. Statistical language modeling using the CMU-cambridge toolkit. In *Proc. Eurospeech '97*, pages 2707–2710, Rhodes, Greece.

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause restructuring for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Ann Arbor, Michigan, June.

G. Doddington. 2002. Automatic evaluation of machine translation quality using n-gram cooccurrence statistics. In *Proceedings of the HLT*, pages 257–258, San Diego, USA.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 228–235.

K. Knight and Y. Al-Onaizan, 1999. *A Primer on Finite-State Software for Natural Language Processing*, August. http://www.isi.edu/licensed-sw/carmel/carmel-tutorial2.pdf.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistic*, 25(4):607–615.

Philipp Koehn and Christof Monz. 2005. Shared task: Statistical machine translation between european languages. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pages 119–124, Ann Arbor, Michigan, June.

P. Koehn, F.J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference (HLT)*, pages 127–133.

P. Koehn. 2004. Pharaoh: a beam search decoder for phrase-based SMT. In *Proceedings of AMTA*, pages 115–124.

Daniel Marcu and William Wong. 2002. A phrase-based, joint probability model for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

F.J. Och and H. Ney. 2000. Improved statistical alignment models. In *Conference of the Association for Computational Linguistic (ACL)*, pages 440–447, Hongkong, China.

F.J. Och and H. Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30:417–449.

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

Daniel Ortiz-Martínez, Ismael García-Varea, and Francisco Casacuberta. 2005. Thot: a toolkit to train phrase-based statistical translation models. In *Tenth Machine Translation Summit*, pages 141–148, Phuket, Thailand.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the ACL*, pages 311–318, Philadelphia, Pennsylvania.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal SMT. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 271–279, Ann Arbor, Michigan, June.

A. Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of ICSLP*, Denver, Colorado, Sept.

C. Tillmann and H. Ney. 2003. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29:97–133.

Nicola Ueffing, Franz Josef Och, and Hermann Ney. 2002. Generation of word graphs in statistical machine translation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, Philadephia, PA, July 6-7.

Daniel J. Walker. 2005. The open "a.i." kit^{TM}: General machine learning modules from statistical machine translation. In *Workshop on MT Summit X, "Open-Source Machine Translation"*, Phuket, Thailand.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, pages 523–530, Toulouse, France.