Design of the Moses Decoder for Statistical Machine Translation

Hieu Hoang
University of Edinburgh
h.hoang@sms.ed.ac.uk

Philipp Koehn
University of Edinburgh
pkoehn@inf.ed.ac.uk

Abstract

We present a description of the implementation of the open source decoder for statistical machine translation which has become popular with many researchers in SMT research. The goal of the project is to create an open, high quality phrase-based decoder which can reduce the time and barrier to entry for researchers wishing to do SMT research. We discuss the major design objective for the Moses decoder, its performance relative to other SMT decoders, and the steps we are taking to ensure that its success will continue.

1 Motivation

Phrase-based translation has been one of the major advances in statistical machine translation (Brown et al. 1990) in recent years and is currently one of the techniques which can claim to be state-of-the-art in machine translation. Phrase-based models are a development of the word based models as exemplified by the (Brown et al. 1990). In phrase-based translation, contiguous segments of words in the input sentence are mapped to contiguous segments of words in the output sentence.

In SMT, we are given a source language sentence, s, which is to be translated into a target language sentence, t. The goal of machine translation is to find the translation, $\hat{t}$, which is defined as:

$$\hat{t} = \arg\max_t \, p(t \mid s)$$

where $p(t \mid s)$ is the probability model. The argmax implies a search for the best translation $\hat{t}$ in the space of possible translations t. This search is the task of the decoder, which we will concentrate on in this paper.

There have been numerous implementations of phrase-based decoders for SMT prior to our work. Early systems such as the Alignment Template System (ATS) (Och and Ney 2004) and Pharaoh (Koehn 2004) were widely used and accepted by the research community. ATS is perhaps the cross-over system, in that word classes were translated as phrases but the surface words were translated word by word. Pharaoh substituted the word classes with surface words, thereby discarding the use of word classes in decoding altogether.

There has been other phrase-based decoders such as PORTAGE (Sadat et al. 2005), Phramer (Olteanu et al. 2006), the MITLL/AFRL system (Shen et al. 2005), ITC-irst (Bertoldi et al. 2004), Ramses/Mood (Patry et al. 2006) to name but a few. Other researchers such as (Kumar and Byrne 2003) have also used weighted finite state transducers but they have more difficulty modeling reordering.

Many early systems came with restrictive licenses; ATS has never been publicly released, Pharaoh was released in 2003 as a pre-compiled binary with documentation. This severely limited the extent to which other researchers can study and enhance the decoder. Without access to the decoder source code research was generally restricted to altering the input, augmenting it with extra information, or modifying the output or re-ranking the n-best list output.

The main contribution of this paper is to show how we have created an extensible decoder, has acceptable run time performance compared to similar systems, and the ease of use and development that has made it the preferred choice for researchers looking for a phrase-based SMT decoder.

As an indication of the take-up of the Moses toolkit, out of over 20 competing teams at the recent IWSLT 2007 conference[1], half used Moses.

As an indication of the extensibility of the decoder, there are currently four language model implementations which has been integrated with the decoder by various researchers. In addition, the framework exists to integrate language models, such as those described in (Bilmes and Kirchhoff 2003), which takes advantage of the factored representation within Moses.

It is noted that Mood/Ramses also supports multiple LM implementations, an internally developed language model, in additional to SRILM, to overcome the latter's licensing restrictions.

In addition, there are two built-in phrase table implementations, one which loads all data into memory for fast decoding, and a binary phrase table as described in (Zens and Ney 2007) which loads on demand to conserve memory usage.

The Moses decoder has the ability to accept simple sentence input, confusion network or lattice networks, in common with SMT decoders such as the MITLL/AFRL or ITC-irst systems. The decoder also produces diverse types of output, ranging from 1-best, n-best lists and word lattices.

2 Comparison with other projects

The Moses decoder is designed within a strict modular and object-oriented framework for easy maintainability and extensibility.

In designing the decoder, we modeled the software design methodology and aims on some research-oriented software libraries outside of the SMT and NLP field which is open source, written in C++, have a large and diverse user-base, have succeeded in becoming the industry norm in their field.

Specifically, we modeled the software on the CGAL library (Fabri et al. 2000), used in computational geometry, and DCMTK (Eichelberg et al. 2004) library used in medical imaging. We believe they set good examples of the standards that we should follow.

However, there are differences between our project and CGAL or DCMTK.

The first difference is project size, for example, whereas CGAL consists of over 500,000 lines of code and multiple libraries and example program, the Moses decoder consists of 20,000 lines in 2 libraries. The difference is scale makes implementing some steps in the development life cycle impractical or unnecessary. For example, functionality specification before implementation was described for CGAL and is typical of large projects but would have been cumbersome for Moses.

Secondly, the aims of Moses and these projects are different. The goal of the CGAL project is to *'make…computational geometry available for industrial application'*[2].

Both CGAL and DCMTK are used extensively in commercial applications. Therefore, issues such robustness, cross-platform compatibility and ease-of-use are predominant for these projects.

Commercialization is not an aim of the Moses project but we believe these issues are still as important as they affect the usability and uptake of the system. Therefore, the Moses decoder was built to address these issues without compromising the academic priorities of the project.

Thirdly, the correct implementation is easier to decide in libraries such as CGAL as the algorithms are closely specified by the mathematical specification, therefore, testing and specification writing is more prevalent and easier than in Moses. For DCMTK, the medical imaging standards and protocols offers a clear guide for implementation. By contrast, the function of an SMT decoder is search for which there are no correct implementation, we can only measure its performance relative to previous versions and other similar decoders.

These differences are minor compared to the similarities Moses has to CGAL and DCMTK, and indeed, to any well developed software project. Design goals such as robustness, flexibility, ease of use and efficiency are commonality that we share and which we will discuss in more detail in the next section.

As a contrast to CGAL and DCMTK whose design we would like to emulate, we also looked at a project within the NLP field which contains certain aspect in the design we would like to avoid.

GIZA++ (Och and Ney 2003) is a very popular system within SMT for creating word alignment from parallel corpus, in fact, the Moses training scripts uses it. The system was release under the GPL open source license. However, its lack of

[1] http://iwslt07.oitc.it/menu/program.html

[2] http://cordis.europa.eu/esprit/src/21957.htm

clear design, documentation and obscure coding style makes it difficult for other researcher to contribute or extend the system. For a long time, it couldn't even be compiled on modern GCC compilers. Other systems which seeks to improve word alignment and segmentation, such as MTTK (Deng et al. 2006), have been created to replace GIZA++.

3 Design Goals

We decided to develop the Moses decoder as a C++ library.

We steered clear of scripting languages for performance reasons and the fact they often offer even less in the way of cross-platform compatibility. Java was also avoided for performance reasons but it's rich library and multi-platform support would have been useful.

We note that Hiero (Chiang 2005) is written in a scripting language with performance critical components rewritten in a compiled language. This is not the approach we considered as we believed it would have raised the complexity and reduce reliability of the project having to develop (and debug) in two languages and managing the interface between them. We also note that the LinearB and Phramer decoders are implemented in Java and have reported significantly worse run time speeds, (Olteanu et al. 2006).

C++ can be inelegant and difficult for inexperienced developers but using other object oriented language such as Smalltalk or C# was out of the question as they lack acceptance within the MT research community.

3.1 Comparable Performance

The Pharaoh decoder (Koehn 2004) represented the state-of-the-art in phrase-based decoders prior to the introduction of Moses. Moses was designed to supersede Pharaoh in performance and functionality. Moses was used as the basis for the JHU Workshop (Koehn et al. 2006) on Factored Machine Translation where it was extensively enhanced; we capitalized on the experience of colleagues at the workshop and used Pharaoh as the baseline during development to ensure that we obtain comparable performance. Table 1 shows the comparison of the translation performance of Pharaoh and Moses for a typical decoding of 2000 sentence trained on the news-commentary corpus[3]. We also include Phramer as an example of a Java-based decoder. Due to improvements in the search algorithm, Moses can slightly outperform Pharaoh on most tasks, which was confirmed by (Shen et al. 2007).

Table 1 Comparison with pharaoh & Phramer for a typical fr-en translation of 2000 sentences

	Time taken	Peak memory usage	BLEU
Pharaoh	99min	46MB	19.57
Moses	69min	154MB	19.57
Moses, with load on-demand PT & LM	102min	239MB	19.57
Phramer	649min	1218MB	19.44

In addition, most of the functionality of Pharaoh has been replicated.

3.2 Integration of Word-Level Factors

The Moses decoder isn't purely a clone of Pharaoh, it was created to conduct research into word-level factors in phrase-base MT. Whereas traditional, non-factored SMT typically deals only with the surface form of words, factored translation models augments different factors, such as POS tags or lemma, into source and target sentences to improve translation. This transforms the representation of a word from a string to a vector of strings, and a phrase or sentence from a sequence of words to a sequence of vectors. Such a change to the basic data structure of a decoder propagated throughout the rest of the system, therefore, it was simpler to build the Moses decoder from scratch rather than extend an existing decoder such as Pharaoh.

Some research into factored machine translation has been published by (Koehn and Hoang 2007).

3.3 Flexibility

Flexibility is an important software design goal which will enable researchers to extend the use of the Moses decoders to tasks that were not originally envisioned.

Following (Fabri et al. 2000), we identify four sub-issues which affects flexibility:

i. Modularity

[3] http://www.statmt.org/wmt07/shared-task.html

ii. Adaptability
iii. Extensibility
iv. Openness

3.4 Modularity

Firstly, software modularity enables developers to work on one component of the decoder without affecting other components. A modular design reduces the learning curve for developers by shielding them from having to understand the entire system if they are only developing a specific part.

Modularity also assists in the re-using of components by separating the implementation details from the module interface.

Moses takes advantage of C++ support for object-oriented and generic programming to enable modularity.

In keeping with the extensible design of CGAL and DCMTK, the core of the decoder is compiled as a static library which can interact with other components through a well-defined API. The simple application which currently comes with the decoder enables users to use the system via the command line and also provides an example of the API.

Therefore, the current typical compilation of the decoder would combine the libraries from IRSTLM, SRILM, Moses, and moses-cmd to create a binary executable.

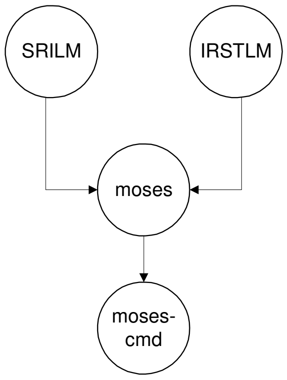

Figure 1 Project Dependencies

Any of these libraries can be dropped or replaced with other components with the same API.

We detail some examples of the object-oriented design of Moses below.

The input into the decoder can be one of three types: a simple string (sentence), a confusion network or a lattice network, Figure 2.

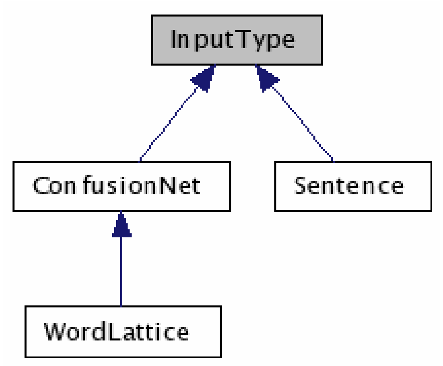

Figure 2 Input Types

Language models are abstracted to enable different implementations to be used and provide a framework for more complex models such as factored LM and the Bloom filter language model (Talbot and Osborne 2007). Similarly, phrase tables are abstracted to provide support for multiple implementations.

Each component model which contributes to the log-linear hypothesis score inherits from the ScoreProducer base class, Figure 3.

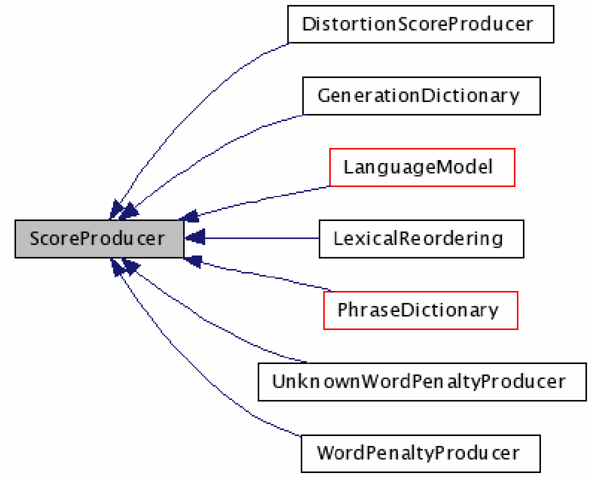

Figure 3 Score Producer

The Moses library provide a simple API whose main entry point is the class

```
Manager
```

This class is instantiated in the client application, moses-cmd in our case. Each input is decoded by calling the class method below:

```
ProcessSentence()
```

3.5 Adaptability

Phrase-based SMT is a fast moving research field where virtually all aspects of the theory are still being explored and implementations can be improved. The Moses decoder has to be amenable to researchers to adapt any component of the decoder in ways that perhaps wasn't foreseen in the original implementation.

Certainly, modularity plays an important part in this but it can also have the opposite effect of allowing obtuse or badly written implementation to hide behind the API, reducing the ability for researchers to question, investigate or extend. As a voluntary project, there is limited power to enforce good implementation and it would be difficult not to accept added functionality.

However, we use coding standards and designs during the development of the decoder that we hope makes the task of working with Moses easier for developers, and that they will continue to use those standards to uphold the clarity of the code.

These coding standards include:

i. strict object-oriented design
ii. descriptive variable, class, object and function names
iii. consistent indentation
iv. use of STL containers
v. implementation of STL-compatible iterators for internal container classes.

The source code for the Moses decoder has contributions from a number of developers in the last two years, Figure 4, including four developers who have made significant contributions but were not in the original JHU Workshop. However, code clarity has, by-and-large, remained intact.

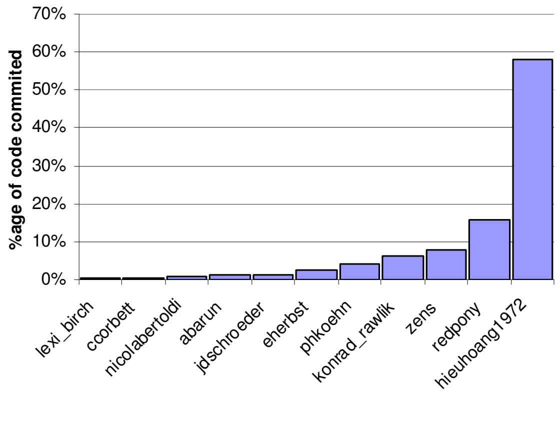

Figure 4 Code committed

We do not know how the decoder will be changed in future, nor do we know where and by whom it will be used. Moses is first and foremost an academic project but that doesn't exclude its use in commercial applications.

We also believe that it will be useful as a teaching tool for computational linguists, machine translation researchers or general computer science students. It is important with such a diverse potential user base, with widely varying degrees of C++ and programming experience, that we make the development and use of Moses as easy as possible, without imposing a significant burden on advanced users.

We would like to lower the learning curve by letting users use Moses in an environment and tools where they are most comfortable with. Therefore, the Moses decoder is operating system and compiler neutral. It is known to run on Windows (natively, or with Cygwin), Linux 32 and 64 bits, Mac OSX and OpenBSD. It is known to be compileable with modern gcc compilers, Visual Studio.net, Intel C++ for both Linux and Windows. We encourage the use of modern graphical integrated development environments (IDE) for Moses and include project files for Visual Studio, Eclipse and XCode, in addition to conventional makefiles.

We note that almost half of the source code downloads for the Moses toolkit from Sourceforge are for the non-Unix version, and that 58% of the visitors to the Moses website uses Windows, Figure 5.

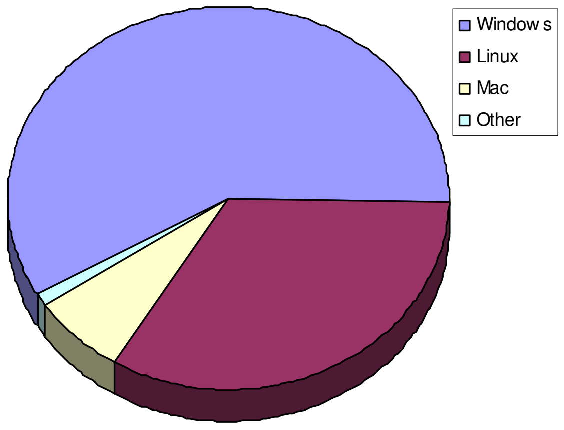

Figure 5 OS of Moses website visitors

This heterogeneous approach allows developers who have previously been excluded to participate within the SMT community and strengthens the decoder by allowing people of different backgrounds to apply their skills. This is of particular concern to us as we are attempting to integrate linguistic information into machine translation with factored decoding.

It also enables best-of-breed tools to be bought to the development of the decoder, regardless of platform. For example, we use both open source and commercial tools on Linux and Windows to track down memory issues, as well as performance profilers. This greatly enhances the efficiency of development and the reliability of the decoder.

Other NLP libraries, such as SRILM (Stolcke 2002) can be compiled and executed under multiple platforms but its development are very much Unix-centric so requires porting tools for non-Unix platforms. We believe the platform and compiler agnostic approach is unique for a major open source C++ project within recent NLP history.

3.6 Openness

An important reason for initiating the Moses project was the need to create a competitive decoder which could be extended with factors, as well as other advances in phrase-based machine translation. It is open source to enable other researchers to extend a state-of-the-art decoder without having to recreate what we have already built.

The decoder was improved at the JHU Workshop by a number of researchers so it needed to be flexible from the beginning. From this experience, we realize that releasing the source code is not enough. The decoder must be written and structured in a clear way to enable other researchers to contribute to the project.

Aside from the legalese of releasing the source code under an open source license, we believe that open source also means the source code is clear and accessible to allow others to examine, critique and contribute. Coding standards aimed at source code clarity and support for modern tools backs this goal.

Documentation of the algorithms used, and of the source code are also essential to allow others to understand the details of the decoder. Every class and function in the Moses decoder is commented in a Doxygen compatible format, HTML documents and figures, such as those in Figure 2 and Figure 3, are generated automatically from these comments and accessible via the Web[4].

Development is done through a source control system and all code changes are open to inspection. We encourage and enable all developers to use and extend Moses and feed back improvements. However, to ensure that the performance of the decoder is maintained and that changes to the decoder doesn't break existing setups, we maintain certain controls over the commit process.

There is a regression test suite which should be passed before any code can be committed to ensure that unintended divergence haven't crept in. A framework exists for creation of regression tests, developers who add new functionality to the decoder are encouraged to create additional tests to ensure that their functionality will work in future.

However, no amount of automated testing can be exhaustive. New committers are subject to peer review by a more experience contributor before the code is committed, and before the contributor is granted write access to the source control system. Also, code commits are monitored via email notifications to a public mailing list.

These measures add a little overhead to the development process this is necessary to maintain the quality of the system and assure to users and developers.

We have benefited from the examples of sound software engineering principles set by the CGAL and DCMTK project and hope that we will emulate their success by bringing these engineering principles into NLP. In contrast to the 'abandonware' status of GIZA++, both CGAL and DCMTK are still being developed.

4 Supporting Infrastructure

Other factors have contributed to the wide adoption of Moses.

4.1 'One-Stop Shop' for Phrase-Based SMT

The Moses project encompasses the decoder and many of the other components necessary to create a translation system which were previously available separately. These include scripts for creating alignments from a parallel corpus, creating phrase tables and language models, binarizing phrase tables, scripts for weight optimization using MERT (Och 2003), and testing scripts.

Steps such as MERT and testing which are CPU intensive have been re-engineered to run in parallel using Sun Grid Engine.

All scripts have also been extended for factored translation.

[4] http://www.statmt.org/moses/html/

4.2 Ongoing support

We assist in the adoption of Moses by offering ongoing support to users and developers through the support mailing list[5]. Questions relating to Moses, phrase-based translation or machine translation in general are often asked, and usually answered. The archived emails are publicly available and searchable, and have become an important knowledge source for the community.

The mailing list popularity has been steadily increasing since its inception, Figure 6, and is now the most popular mailing list for machine translation, based on volume.

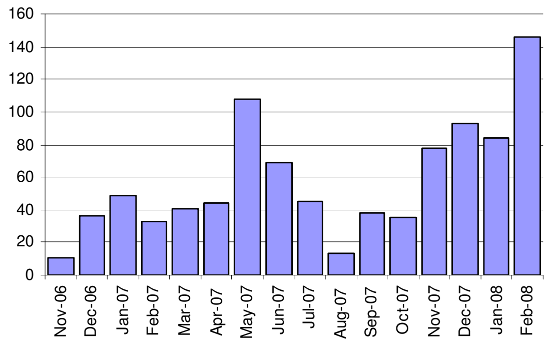

Figure 6 Emails to Moses support mailing list

5 Future Work

There has been some important developments in phrase-based translation in recent years, including the hierarchical phrase-based model as described in (Chiang 2005). Research have also been made into alternatives to the current log-linear scoring model such as discriminative models with millions of features (Liang et al. 2006), or kernel based models (Wang et al. 2007).

From a software engineering point of view, these improvements would require fundamental changes to the structure if they were to be implemented into Moses.

We are also interested in seeing the Moses decoder employed in search tasks outside of machine translation; Moses has been used for OCR correction, recasing, and transliteration.

Other improvements such as smaller, faster, more efficient phrase tables are also welcomed.

Lastly, we would like to see the training and tuning scripts re-engineered to the same modular design as the decoder. The future direction of the Moses decoder requires even more complex models which are already stretching the current script implementation to the limit of adaptability and reliability.

[5] moses-support@mit.edu

6 Conclusion

We have applied the sound software engineering principles and design to the implementation of the Moses decoder which has enabled other researchers to use and extend its functionality. We believe this has been a major factor for the widespread adoption of Moses within the SMT community. We hope that the design of the decoder will enable it to maintain it leading edge status into the future.

Acknowledgements

This work was supported in part under the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-C-0022 and in part under the EuroMatrix project funded by the European Commission (6th Framework Programme.

References

Bertoldi, N., R. Cattoni, et al. (2004). The ITC-irst Statistical Machine Translation System for IWSLT-2004. IWSLT, Kyoto, Japan.

Bilmes, J. A. and K. Kirchhoff (2003). Factored language models and Generalized Parallel Backoff. HLT/NACCL.

Brown, P. F., J. Cocke, et al. (1990). "A statistical approach to machine translation."

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. ACL.

Deng, Y., S. Kumar, et al. (2006). "Segmentation and alignment of parallel text for statistical machine translation." Natural Language Engineering.

Eichelberg, M., J. Riesmeier, et al. (2004). "Ten years of medical imaging standardization and prototypical implementation: the DICOM standard and the OFFIS DICOM toolkit (DCMTK)." Medical Imaging 2004: PACS and Imaging Informatics **5371**: 57-68 (2004).

Fabri, A., G.-J. Giezeman, et al. (2000). "On the Design of CGAL, a Computational Geometry Algorithms Library." Software—Practice & Experience **30**(11, Special issue on discrete algorithm engineering).

Koehn, P. (2004). Pharaoh: a Beam Search Decoder for Phrase-Based Statistical Machine Translation Models. AMTA.

Koehn, P., M. Federico, et al. (2006). Open Source Toolkit for Statistical Machine Translation. Report of the 2006 Summer Workshop at Johns Hopkins University.

Koehn, P. and H. Hoang (2007). Factored Translation Models. EMNLP.

Kumar, S. and W. Byrne (2003). A weighted finite state transducer implementation of the alignment template model for statistical machine translation. ACL, Edmonton, Canada.

Liang, P., A. Bouchard-Côté, et al. (2006). An End-to-End Discriminative Approach to Machine Translation. COLING/ACL.

Och, F. J. (2003). Minimum Error Rate Training for Statistical Machine Translation. ACL.

Och, F. J. and H. Ney (2003). "A Systematic Comparison of Various Statistical Alignment Models." Computational Linguistics **29**(1): 19-51.

Och, F. J. and H. Ney (2004). "The alignment template approach to statistical machine translation." Computational Linguistics.

Olteanu, M., C. Davis, et al. (2006). Phramer - An Open Source Statistical Phrase-Based Translator. ACL Workshop on Statistical Machine Translation.

Patry, A., F. Gotti, et al. (2006). Mood at work: Ramses versus Pharaoh. ACL, New York City, USA.

Sadat, F., H. Johnson, et al. (2005). PORTAGE: A Phrase-based Machine Translation System. ACL Workshop on Building and Using Parallel Texts: Data-Driven Machine Translation and Beyond, Ann Arbor, Michigan, USA.

Shen, W., B. Delaney, et al. (2005). The MITLL/AFRL MT System. IWSLT, Pittsburgh, PA, USA.

Shen, Y., C.-k. Lo, et al. (2007). HKUST Statistical Machine Translation Experiments for IWSLT 2007. IWSLT, Trento.

Stolcke, A. (2002). SRILM An Extensible Language Modeling Toolkit. Intl. Conf. on Spoken Language Processing.

Talbot, D. and M. Osborne (2007). Smoothed Bloom filter language models: Tera-Scale LMs on the Cheap. EMNLP, Prague, Czech Republic.

Wang, Z., J. Shawe-Taylor, et al. (2007). Kernel Regression Based Machine Translation. NAACL HLT.

Zens, R. and H. Ney (2007). Efficient phrase-table representation for machine translation with applications to online MT and speech recognition. HLT/NAACL.