PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 101 APRIL 2014 7–28

Dynamic Models in Moses for Online Adaptation

Nicola Bertoldi

Fondazione Bruno Kessler

Abstract

A very hot issue for research and industry is how to effectively integrate machine translation (MT) within computer assisted translation (CAT) software. This paper focuses on this issue, and more generally how to dynamically adapt phrase-based statistical machine translation (SMT) by exploiting external knowledge, like the post-editions from professional translators.

We present an enhancement of the Moses SMT toolkit dynamically adaptable to external information, which becomes available during the translation process, and which can depend on the previously translated text. We have equipped Moses with two new elements: a new phrase table implementation and a new LM-like feature. Both the phrase table and the LM-like feature can be dynamically modified by adding and removing entries and re-scoring them according to a time-decaying scoring function. The final goal of these two dynamically adaptable features is twofold: to create additional translation alternatives and to reward those which are composed of entries previously inserted therein.

The implemented dynamic system is highly configurable, flexible and applicable to many tasks, like for instance online MT adaptation, interactive MT, and context-aware MT. When exploited in a real-world CAT scenario where online adaptation is applied to repetitive texts, it has proven itself very effective in improving translation quality and reducing post-editing effort.

1. Introduction

One of the hot topic in Machine Translation (MT) research is the integration of MT technology into Computer Assisted Translation (CAT) tools, and more specifically the efficient exploitation of human translator feedback to improve the MT system.

The ultimate goal of incorporating an MT system in a CAT tool is to increase the productivity of a human translator by providing her/him translation suggestions, which are as accurate as possible, as close as possible to her/his preferences, and as

Corresponding author: bertoldi@fbk.eu
Cite as: Nicola Bertoldi. Dynamic Models in Moses for Online Adaptation. The Prague Bulletin of Mathematical Linguistics No. 101, 2014, pp. 7–28. doi: 10.2478/pralin-2014-0001.

coherent as possible throughout a whole document from a lexical, semantic and syntactic perspective. To this purpose, the continuous exploitation of the post-edits of a human translator is crucial for updating the MT system embedded in the CAT tool and improving its quality. The above-mentioned task is a special case of the more general topic of adapting an MT system using dynamic techniques, i.e. the scenario in which a MT system is applied to a stream of input text, while simultaneously receiving a stream of informative data to improve the behavior of the system over time For instance, MT technology to translate a stream of news content could benefit from the external knowledge about the current news domain, if properly fed into the MT models.

The current standard MT technology is not completely appropriate to this online adaptation scenario, due to some intrinsic shortcomings. For efficiency purposes, translation is performed independently sentence by sentence, and the context of the previous and next sentences is not taken into consideration; hence, the lexical, syntactic and semantic consistency cannot be assured at the document-level. Most MT models are estimated on a predefined corpus in the preliminary training phase, and they cannot be modified during the translation process; if any change is required the MT system needs to reload the models, which is an expensive operation due to their potential huge size. MT systems do not have learning capabilities either, and hence they cannot be adapted and become responsive to translator feedback.

In order to overcome some of these limitations, MT technology should employ modifiable models. Here is a brief and partial list of desiderata for such dynamic models: the insertion and deletion of entries and the modification of of their scores should be permitted; the changes inside the models should be immediately available to the decoder without reloading the models; according to the task, the changes could be either local to the current text to translate or persistent over time.

A recent implementation of MT models, discussed later in the paper, relying on the dynamic suffix array data structure does not suffer from the limitations of the static models, but still it does do not satisfy all requirements.

This paper presents an enhanced version of the well-known and world-wide used Moses SMT toolkit (Koehn et al., 2007), providing simple, modular and affordable solutions to make it dynamic. New implementation types for the phrase table and the language model are created, which give the required functions, and additionally can be modified in any time by means of command line instructions.

The paper is organized as follows. Section 2 overviews the phrase-based Moses system. In Section 3 a comprehensive description of our proposed enhanced dynamic version of Moses is given; more specifically, details are supplied about the employed data structures, the scoring of the entries in the translation and language models, the way of communicating to the decoder and updating the models. Section 4 introduces the standard generic framework where the proposed dynamic system can be applied. Section 5 compares our approach to tackle the online MT adaptation to other solutions at disposal in Moses or described in the literature. Section 6 reports on the ef-

ficiency and effectiveness of the proposed solution. Finally, Section 7 ends the paper by presenting some ideas to further improve the dynamic models, some of which are currently under development.

2. Description of the Static System

A high level description of the Moses SMT toolkit is provided aiming at highlighting the elements we modified to build its dynamic enhanced version as well as the decoding process. Among the many variants of Moses the phrase-based decoder is considered.

2.1. Data Structure

Phrase-based Moses relies on one or more *phrase tables*, which provides translation alternatives, and on several *feature functions*, which provide the partial scores for the overall computation of the final translation score.

The phrase table is essentially a set of *translation options*. A translation option pairs a source phrase with one of its target translations and assigns a series of scores from the feature functions. The phrase table is looked up for getting all translation options associated to a given source phrase. In the current version of Moses, several types of phrase tables are implemented; the two most used types are based on prefix trees and suffix arrays. Both data structures are very efficient to store a huge amount of entries and to reply to a search request as fast as possible. For efficiency, the scores of the phrase-based translation model are stored in the same table.

Feature functions can be classified into three groups according to the information they depend on: (i) those based on both source and target words (phrase pairs); (ii) those relying on monolingual –usually target– words (n-grams); and (iii) those depending only on the size of the output. The translation model and the lexicalized reordering model clearly belong to the first group, and both use the data structure of the phrase table. The target language model, which belongs to the second group, is usually provided by a third-party toolkit, like SRILM (Stolcke, 2002), IRSTLM (Federico et al., 2008), randLM (Talbot and Osborne, 2007), kenLM (Heafield et al., 2013); however, a basic LM implementation is available in Moses as well. The feature functions belonging to the third group, like word and phrase penalty, do not require lexical information, but rather they are mostly evaluated on-the-fly during translation.

2.2. Translation Process

The translation of an input sentence is performed into two steps.

In the first phase, called *pre-fetching*, the translation options for all possible *spans*[1] of the input sentence are collected from the phrase table(s). In this phase, pruning of

[1] A *span* is a sequence of consecutive words of the input sentence.

the options is also performed in order to keep the translation process computationally manageable. Options are first ranked according to their score in the phrase tables and an estimate of their expected utility within the final translation, and then the k-best [2] options are retained, while the others are discarded.

In the second phase, called *decoding*, a possibly large number of translation hypotheses are built by (i) selecting a span segmentation of the input sentence, (ii) reshuffling the spans, and (iii) concatenating all translation options of all spans gathered in the pre-fetching phase with the constraint that all words on the input sentence are covered exactly once. Each translation hypothesis is scored by log-linearly interpolating all feature functions.

The heavy computational effort of the decoding stage is made affordable by exploiting very efficient algorithms, like beam search, multi-stack, or cube-pruning, constraints to limit the word reordering and the translation options, and data structures to store the huge amount of translation alternatives.

3. Description of the Dynamic System

The data structures provided by Moses for the phrase table and the lexicalized reordering table, and by other software for the language model are optimized for huge amounts of data and for quick access. Unfortunately, these structures are static and do not admit any change once they have been built and loaded. The suffix-array implementation of the phrase and lexicalized reordering table could be easily adapted for dynamic modification. A comparison between this implementation and ours is given in Section 5.

In order to make Moses dynamically adaptable we propose a new type of implementation for the language model and for the phrase table which are illustrated in Sections 3.1 and 3.2, respectively. The dynamic language model and phrase table behave similarly to the standard language model and phrase table implementations with the additional feature that the set of its accumulated entries (either phrase pairs or n-grams) and their associated scores can change over time, by inserting or re-scoring entries. Furthermore, each entry is associated with an *age*, which is the time the entry was inserted in the dynamic language model and phrase table, and its scores are made dependent on it; a rationale for that is given in Section 3.3.

The dynamic models live in memory and currently the scores of each entry cannot be saved on disk; hence their content is lost in case of exit. Nevertheless, they can be easily re-created by collecting all entries inserted until the exit with the correct age, for instance from an activity log, and by loading them when Moses restarts and the models are initialized. The informing data to update the models are communicated to the system adding a simple xml-based annotation in the input text. Appendix A provides details about the annotation.

[2] k is set by means of the parameter "`-ttable-limit`".

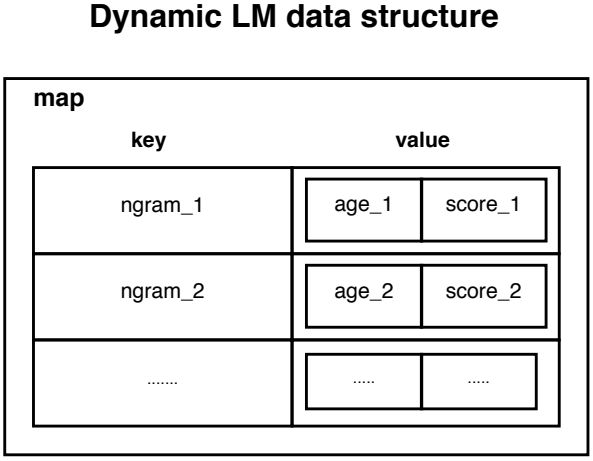

Figure 1. The data structure implementing the dynamic language. It relies on the map object provided by C++ Standard Template Library.

The dynamic system can be hence implemented by exploiting the dynamic phrase table and/or the dynamic language model in addition to or in substitution of the static models. Any method for combining the static and the dynamic phrase tables available in Moses can be ideally applied, namely fill-up, backoff, log-linear interpolation, etc. The dynamic language model must be used as an additional feature in Moses. No modification to the decoding phase is however required.

The source code of the proposed dynamic language model and phrase table is available in the branch "`dynamic-models`" under the official GitHub repository of Moses toolkit, directly accessible from this URL:
`https://github.com/moses-smt/mosesdecoder/tree/dynamic-models`

3.1. Dynamic Language Model

The dynamic language model, shown in Figure 1, is implemented by means of a simple data structure based on a C++ Standard Template Library[3] (STL) map. The map links a n-gram to its age and its age-dependent score. No restriction to the length of n-grams is given.

By means of this map the actions expected for the dynamic language model can be quickly performed: (i) inserting a new n-gram pair with an associated age; (ii) getting and updating the score of a given n-gram; (iii) deleting a specific n-gram; (iv) cleaning the dynamic language model.

[3] www.sgi.com/tech/stl

To insert an n-gram together with its age, first the age-dependent score is computed as described in Section 3.3, and then the data structure is updated. If the n-gram is new, it is added to the map with its age and score, otherwise its age and score are just updated. It is worth noting that the insertion of an n-gram, either existing or new, always implies the aging of all existing entries, and hence their re-scoring. The access and deletion as well the insertion of the n-grams are performed by means of the standard STL map functions. The computational cost is $\mathcal{O}(N)$ for insertion and $\mathcal{O}(\log(N))$ for access and deletion, where N is the total amount of stored n-grams. Insertion is more expensive because all existing n-grams must be aged.

Two modalities of lookup into the dynamic language model are implemented when Moses queries for the score of a target n-gram $(w_1, \ldots, w_l)$. In the first case (*AllSubStrings*), all its substrings of any length $(w_i, \ldots, w_j)$ are searched, their scores are averaged according to the following formula, which takes into account the number of substrings of a specific length, and the resulting score $avg\_score(w_1, \ldots, w_l)$ is actually returned.

$$avg\_score(w_1, \ldots, w_l) = \sum_{x=1}^{l} \frac{1}{x} \left(\sum_{i=1}^{l-x+1} score(w_i, \ldots, w_{i+x-1}) \right)$$

In the second case (*WholeString*) the whole string $(w_1, \ldots, w_l)$ is searched in the map receiving the stored $score(w_1, \ldots, w_l)$ is returned. The first modality exploits the dynamic language model as a simple lexicalized rewarding/penalty feature; instead, the second modality makes the dynamic language model more similar to a standard language model, since it simulates the summation of as many scores as the length of the n-grams. However, the dynamic language model is currently implemented as a stateless feature, and hence it does not support the computation of scores for n-grams across different translation options. This implementation choice is mainly justified by an efficiency reason: the lookup in the dynamic language model is performed only once and only for the n-grams included in the pre-fetched translation options; if we admitted the lookup of all possible n-grams created at translation time, like for a standard LM feature, the computational cost could become unaffordable.

3.2. Dynamic Phrase Table

The dynamic phrase table, which stores translation options and their associated ages and scores, is implemented by means of a two-tier data structure of maps and vectors, as sketched in Figure 2. Map and vector objects are supplied by the C++ STL. No restriction to the length of source and target phrases is given. The dynamic phrase table extends the Moses basic object *PhraseDictionary*.

The keys of the map consist of source phrases, and the corresponding values are pairs of pointers. The first pointer links to the current collection of translation options of the source phrase; for them, a specific data structure (*TranslationOptionCollection*)

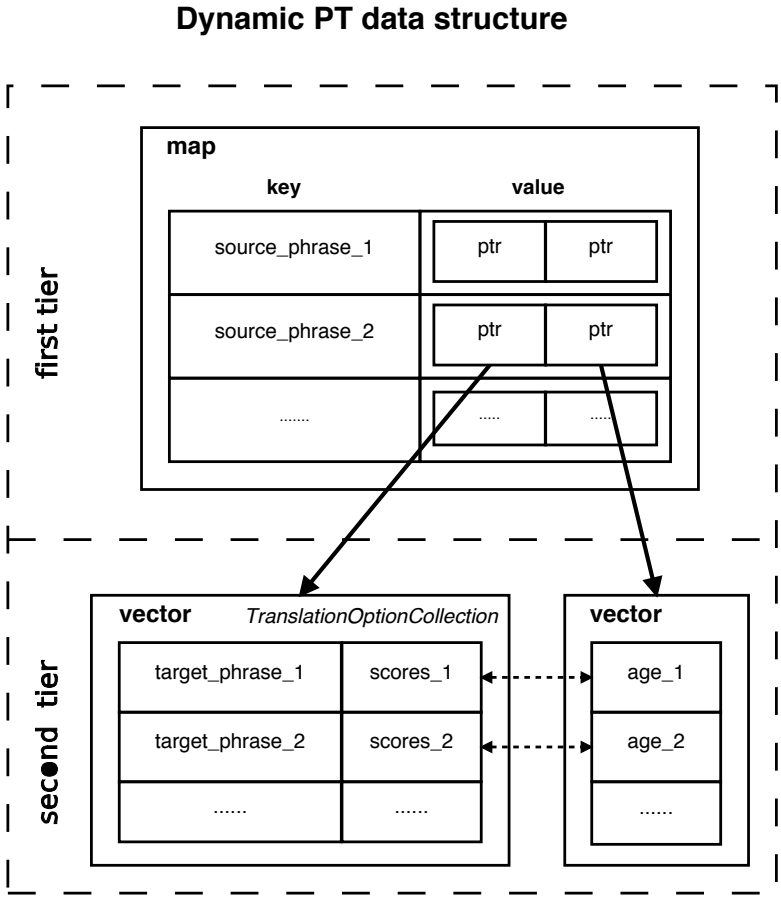

Figure 2. The two-tier data structure implementing the dynamic phrase table. It relies on map and vector objects provided by C++ Standard Template Library, and on the vector-based TranslationOptionCollection *object provided by Moses. Dashed lines are not real links; they just show that the entries of the two vectors are kept aligned.*

already available in Moses is exploited, which is essentially a vector of target phrases and associated scores. The score for each option is actually a vector of floating values. The second pointer links to a vector storing the ages associated to each translation options. The two vectors of target phrases and ages are kept aligned to simplify the access, modification, insertion and deletion of translation options.

The proposed data structure allows to perform the actions expected for the dynamic phrase table: (i) inserting a new phrase pair with an associated age; (ii) getting and updating the scores of a given phrase pair; (iii) returning the collection of translation options for a given source phrase; (iv) removing a specific phrase pair; (v) removing all translation options associated to a given source phrase; (vi) fully cleaning the dynamic phrase table.

To insert a phrase pair together with its age, first the new age-dependent score is computed as described in Section 3.3, and then the data structure is updated. Three cases are possible: (i) if a the source phrase does not exists a new entry in the map is inserted pointing to two newly created vectors, where the new target phrase and its age are inserted; (ii) if the source phrase exists but the target phrase does not, the new target phrase and its age are inserted just at the end of the two associated vectors, hence, keeping their alignment; (iii) if both source and target phrases exist, the corresponding entries in the two vectors are updated according to the new age and scores. It is worth noting that the insertion of a phrase pair, either existing or new, implies the aging of all existing entries, and hence their re-scoring.

To delete a phrase pair, first it is searched in the data structure, then the target phrase and the corresponding age are removed from both second-tier vectors, and finally the source phrase is removed from the first-tier map, if its corresponding translation option collection becomes empty. To delete all translation options associated to a given source phrase, first the two vectors linked to it are destroyed and then the source phrase is removed from the first-tier map. The cleaning of the whole dynamic model is done by recursively removing all source phrases.

To access the whole collection of the translation options of a specific source phrase, a quick lookup into the first-tier map is sufficient. The more expensive access to a specific phrase pair happens only when a new entry is added, but not during the pre-fetching and decoding phases.

The access to the whole collection of translation options for a given source phrase is performed very frequently in the pre-fetching phase of the decoding; hence, the data structure is optimized for minimizing the complexity of this action. More specifically, assuming that the dynamic phrase table stores S source phrases and T translation options for each source phrase on the average, and considered the complexity of the STL map- and vector-based functions employed, the global cost for inserting a phrase pair is $\mathcal{O}(T\,S)$, for deleting it is $\mathcal{O}(T\,\log(S))$, and for accessing the whole collection of translation options associated to a given source phrase is only $\mathcal{O}(\log(S))$. The insertion operation is expensive because all existing entries must be aged.

By default, the dynamic phrase table provides a vector of one score for each entry. However, it is possible to associate a vector of N values, by equally dividing the age-dependent score; this could be useful for instance if the dynamic phrase table is combined with a static phrase table, which has 4 scores by default.

3.3. Time-decaying Scoring Functions

The dynamic models presented in this paper aim at updating the SMT system by benefitting from information which becomes available during the translation of a document in an online framework. Such information can change over time according to the new text to translate, to the translations produced so far, and to the feedback received by an external knowledge source, like human post-editors and evaluators.

It is reasonable to give more importance to more recent update requests. Henceforth, we developed an approach, where each entry in the dynamic models is always associated with its age, which counts how many insertion operations have occurred after it was actually inserted. A score depending on such age is then associated to each entry according to the core policy that "the more recent, the worthier". This age-dependent score is actually exploited by the decoder during translation.

	hyperbola	power	exponential	cosine
$reward(age)$	$\frac{1}{age}$	$\sqrt[4]{\frac{1}{age}}$	$\exp\left(\frac{1}{age}-1\right)$	$\cos\left(\frac{\pi}{2}\frac{age-1}{MaxAge}\right)$
$penalty(age)$	$\frac{1}{age}-1$	$\sqrt[4]{\frac{1}{age}}-1$	$\exp\left(\frac{1}{age}-1\right)-1$	$\cos\left(\frac{\pi}{2}\frac{age-1}{MaxAge}\right)-1$

Table 1. Several types of reward and penalty scoring functions depending on the age of an entry.

Several scoring functions are defined as reported in Table 1. These functions operate like a decreasing reward, ranging from 1.0 to 0.0, or like an increasing penalty ranging from 0.0 to -1.0. During translation the Moses decoder could look for an entry which is not included in the dynamic model either because it was never inserted or because it became too old and hence it was removed. In this case, it receives a lower-bound no-match score fixed to 0.0 and to $penalty(MaxAge)$ for the reward and penalty scoring functions, respectively.

Distinct scoring functions and distinct values of $MaxAge$ can be configured for the dynamic phrase table and language model. Details how to configure the dynamic models are given in Appendix B.

4. Translation with a Dynamic SMT System

As mentioned in Section 1, a typical use of the proposed dynamic models is their exploitation in a SMT system, which adapts as soon as external (supervised) feedback is available, like in a CAT scenario.

1. initialize dynamic models M_d from files
2. estimate combined models M_c
3. initialize Suggestion Manager
4. for source sentence src_i, $i = 1 \ldots, N$
5. get annotation ann_i from Suggestion Manager
6. update M_d exploiting ann_i
7. update M_c
8. get translation tr_i of src_i exploiting M_c
9. (optionally) get post-edit pe_i of tr_i
10. update Suggestion Manager exploiting src_i, tr_i and pe_i

Figure 3. General procedure to translate with a dynamic SMT system. The informing data to update the dynamic models are generated by a Suggestion Manager, which optionally exploits human post-edits.

A static SMT system can translate segments of an input text in any order, and likely in parallel to save time; instead, a dynamic SMT system must proceed sequentially, and communicate synchronously with an external source to get the required feedback. The general procedure is sketched in Figure 3.

Let us assume to translate a document $D = \{src_1, \ldots, src_N\}$ composed of N segments. In the initialization phase, the dynamic models M_d are created by optionally loading entries from files (step 1), and the combined models M_c are estimated exploiting both M_d and standard static models (step 2).

An external module (*Suggestion Manager*) is supposed to be available, which provides the informing data to update the dynamic language model and phrase table in the format shown in Figures 2 and 3 in Appendix A. The Suggestion Manager potentially relies on the full input document, and on the portion of the translation produced so far. Hence, at the beginning it is initialized exploiting only D (step 3).

The translation of D proceeds sequentially sentence after sentence (step 4) as follows: first the informing data ann_i are requested from the Suggestion Manager (step 5); the dynamic models M_d are updated by exploiting ann_i (step 6) and combined into M_c (step 7); then, Moses produces the translation tr_i of the source sentence src_i exploiting the current combined model (step 8); optionally, the post-edit pe_i is collected (step 9), if human intervention is considered; finally, the Suggestion Manager is updated by exploiting src_i, tr_i, and pe_i (step 10).

It is worth noting that one instance of Moses runs continuously from the beginning, and it is not restarted for each input sentence; to achieve this goal Moses should be run in a client/server modality, no matter on which communication channel (standard I/O, socket, or other) it relies.

The dynamic system is efficient in an online MT adaptation task, if creating the informing data (step 5) and updating the combined model M_c and the Suggestion

Manager (steps 7 and 10, respectively) are fast enough. The update of M_d (step 6) is efficient per se thanks to our proposed implementation.

The log-linear interpolation provided by Moses is a good choice for an high-speed combination of static and dynamic models; in this case, the dynamic models are used as additional features which are configured as described in Appendix B. Combining dynamic and static phrase tables by means of fill-up, backoff, or linear interpolation, is also possible; however, in this case the dynamic table must be configured to store the same amount of scores as the static table. In principle, the combined model could be composed by the dynamic model only, and in this case no a-priori background knowledge is used. Vice-versa, the dynamic model could be completely disregarded, and in this case the system loses his adaptation capability.

Concerning the effectiveness of the proposed dynamic models, it is important to stress that the translation performance of the dynamic SMT system strongly depends on the quality of the informing data (phrase pairs and n-grams) generated by the Suggestion Module. Although the development and the description of a high-performing Suggestion Manager is out of the scope of the paper, it is worth mentioning main desiderata: (i) the informing data returned by the Manager consist of a possibly empty set of phrase pairs and n-grams from which the annotations are built; (ii) the informing data must depend on only the source segments, translations and post-edits available so far, but not necessarily on all of them; (iii) computation of the informing data must be reasonably fast.

A basic Python-based software `"onlineSMT.py"` was written implementing the online translation procedure described in Figure 3 and based on the dynamic SMT system; it is available in the directory `"scripts/online"` of the Github Moses branch `"dynamic-models"` (see Section 3 for its URL). With respect to the standard version of Moses, this software enables the sequential translation of the source sentences, the collection of the informing data from the Suggestion Manager, and the update of the dynamic models. Two Suggestion Managers were developed which provide the informing data (phrase pairs or n-grams): one relies on the Constrained Search algorithm (Cettolo et al., 2010); the other on a modified online version of mgiza++, and Moses phrase extractor. A third basic Suggestion Manager which does not create any informing data was also developed to mimic a static system within an online framework. Moreover, the dynamic wrapper optionally creates annotations not only for the dynamic models but also for the hard-coded ”xml-input” function of Moses. Usage instructions are available.

The software for tuning the Moses by means of Minimum Error Rate training procedure (Och, 2003) was also slightly modified, in order to properly handle the dynamic system; the new version `"mert_dynamic_moses.py"`, available in the same location, essentially substitutes the batch translation by means of the standard Moses with the online translation by means of the dynamic wrapper `"onlineSMT.py"`. The software allows the optimization of the additional weights for the dynamic models in a real online framework.

5. Comparison of Alternative Approaches for Online MT Adaptation

Moses includes several implementations of the phrase table, like OnDisk, Binary, Compress.[4] All of them aim at optimizing either memory consumption, access time, or disk usage; nevertheless they are static and cannot be modified once trained; hence they are not suitable in an online adaptation framework.

Moses is already able to modify its behavior at run-time by means of the "xml-input" function. The decoder can be fed with new phrase pairs and scores, which are used as exclusive or additional options for the sake of the translation. A few shortcomings of this approach are briefly reported: (i) the suggested options refer to a specific input span; (ii) it is not possible to provide options for overlapping spans; (iii) the suggested options are at disposal only for the current sentence, and then discarded; (iv) the LM is not modified at all.

Moses includes an implementation of the phrase table based on a suffix-array data structure. The phrase table is not created and scores are not computed in the training phase; the translation options are instead collected and scored on-the-fly at translation time, by means of an extremely efficient method of storing and searching the training corpus; word-alignment information for all sentence pairs should be also included in the training data. The suffix-array phrase table can be made suitable for online adaptation by extending the training corpus itself with the suggested options. Levenberg et al. (2010) presented stream-based translation models, based on a dynamic suffix array which allow to add and delete parallel sentences, and maintain the memory space bounds. To me, this phrase table implementation has a few weaknesses: (i) suggested options are merged into the whole training corpus; hence, it is not trivial rewarding them with respect to the others; (ii) the changes are persistent over time, because the new informing data are essentially fused into the training data; (iii) still, the LM is not modified at all.

The research about language model adaptation, conducted not only in MT but also in the speech processing area (Bellegarda, 2004), does not fit very well the online scenario considered here. In fact, to our knowledge most approaches to LM adaptation aim at re-estimating and/or re-tuning the internal LM parameters (i.e. n-gram probabilities) when new data appear, but they usually do not allow adding or deleting new n-grams on-the-fly. Levenberg and Osborne (2009) investigated in this direction; they presented an enhancement of the randomized language model, which is adapted by means of an efficient (in space and time) incremental training algorithm as soon as a small bunch of data becomes available, and which performs comparably to the corresponding batch re-trained LM. In our opinion, their approach, although very efficient, becomes infeasible in our online scenario, because the frequency of the adaptation step is extremely high, one every new sentence. Moreover, the deletion

[4] See the official Moses documentation for details and credits to authors.

of n-gram in the model is not controlled by external feedback, but just regulated by memory bounds.

Our new dynamic implementation of language model and phrase table overcomes the drawbacks of the mentioned approaches. In particular, (i) the entries inserted in the dynamic models are persistently available for the translation of the future sentences; (ii) however, they can be removed from the models at any time; (iii) if the suggested options refer to overlapping spans, the choice of the best alternative is made in the decoding phase by avoiding any potentially dangerous greedy decision taken before; (iv) due to the time-decaying scoring function, it is possible to reward or penalize specific translation options; (v) a way to dynamically update a LM-like feature is provided.

Other works proposed solutions for dynamic adaptation of MT systems, but either they did not make the source code publicly available, or they imposed stricter constraints than ours. For instance, Nepveu et al. (2004) described a dynamic system built at the word-level only and relying on IBM Model 2 to get word alignment. The dynamic systems based on caches proposed by Tiedemann (2010) and Gong et al. (2011) are updated "with the translation options used in the best (final) translation hypothesis of previous sentences", and hence they cannot embed a stream of external knowledge.

We have been certainly inspired from their ideas to implement the dynamic system, but we have created a more flexible and extendible infrastructure, making it available for further improvements.

6. Performance

The dynamic MT system presented in this work has already been successfully and effectively employed in an online MT adaptation scenario (Wäschle et al., 2013; Bertoldi et al., 2013). In these experimental investigations, we tested a variety of methods to generate informing data from the human post-edits, as well as several combinations of the dynamic models. We tried various settings on translation tasks in domains such as English-Italian Information Technology and English-German Patents. Relative improvements in terms of BLEU score up to 10% are reported over a state-of-the-art static system.

Here a brief summary of their analytic outcomes is reported.

- Best improvements are achieved when both dynamic phrase table and dynamic language model are employed. This is not surprising because they support each other in rewarding best or possibly new translation options suggested by the external user feedback. Moreover, each dynamic model is less effective if used alone, but still it is.
- The dynamic phrase table outperforms the hard-coded solution provided by Moses to insert new translation options via "-xml-input" feature. Through our novel approach Moses explores a larger search space and hence has an higher

level of flexibility to build the final translation; in fact, the cached translation options can be exploited at decoding time without any constraint. In the "xml-input" method they are instead strictly joint to a given source segment in a preliminary phase.
- The dynamic models are very effective with repetitive texts, but do not hurt with less repetitive texts; to a certain extent, accuracy gains are correlated with the document's level of repetitiveness.
- The dynamic models are absolutely agnostic about the informing data, which are provided from an external module (Suggestion Manager) and added into their caches without any quality check. Hence, the dynamic system strongly depends on the capability of this module to extract good, reliable and useful information from the user post-edits. In our ongoing further investigation, we observed much higher BLEU improvement, up to 20% relative, when a high-quality Suggestion Manager is employed.

The dynamic models were designed to store a limited amount of high-quality informing data (either phrase pairs or n-grams) and to make them available to the decoder in order to translate subsequent input sentences as coherently and consistently with the user feedback and the recent context as possible. By using time-decaying reward and penalty functions it is unnecessary to keep very old information in the cache, because their contribution in the overall translation process is negligible. In our experimentation, we usually stored no more than a thousand of entries. Indeed, in the considered online adaptation scenario, the recent user feedback is quite limited, hence the useful informing data which can be extracted from the feedback itself is small. If, by any chance, large amounts of informing data were available in advance, they could simply be exploited by other efficient but static data structures. Consequently, an extremely efficient data structure is not required at all; experimental evidence shows that the additional memory consumption is totally negligible in our standard configuration. By choice we preferred to have an easily adaptable and extendable implementation for the dynamic models.

We evaluated the impact of the dynamic models on the efficiency of the MT system by comparing the translation speed of the static and dynamic systems under computationally controlled conditions. We translated from English to Italian 10 documents from an Information Technology domain, for a total amount of 2.5K sentences and 39K running words, with the `"onlineSMT.py"` software introduced in Section 4. We employed the state-of-the-art static system[5] and the derived dynamic system equipped with a Suggestion Manager, which provides informing data by means of the Constrained Search algorithm.[6] With this setting, on the average approximately 7 phrase pairs and 53 n-grams are extracted from each post-edit and added to the dynamic

[5]For a fair comparison the static system also translates the text sequentially.

[6]Both systems are detailed in (Bertoldi et al., 2013).

phrase table and language model, respectively. The dynamic system is about 14% slower than the static system; approximately half of the additional computational time is due to the creation of the informing data by the Suggestion Manager. The dynamic system outperforms the static system by 13% relative BLEU.

7. Conclusion and Future Work

We have presented an enhanced version of Moses which provides a simple, modular and affordable solution to make it dynamic, in the sense that its models can be modified on-the-fly without reloading. The dynamic Moses system is hence able to quickly adapt to external feedback recommending new or better translation alternatives. This capability is achieved by means of a novel phrase table implementation and a new LM-like feature which dynamically store the external informing data and reward them according to a time-decaying scoring function.

These enhancements can be applied to the CAT scenario where the dynamic SMT system benefits from the translator's post-edits and s(he) benefits from the improved MT suggestions; hence, a virtuous cycle is established to diminish the overall post-editing effort. Recent papers presented experimental evidence of the effectiveness of the dynamic models described in this work; in an appropriate online MT adaptation framework the exploitation of the dynamic system significantly outperforms a static system up to 20% relative BLEU. In our experiments we found compatible results: the dynamic system outperforms the corresponding static system by 13% relative BLEU on the average, while its average translation speed decreases of approximately 14%.

However, these outcomes must be confirmed by employing the dynamic system in other translation tasks to further verify its versatility and its effectiveness. Additional experimental investigation is required to deeply understand the interaction between the components of the dynamic system, the effects of external factors, like the quality and the quantity of the informing data, the characteristics of the input text, and concurrent or contradictory feedbacks, and the impact of these variables on the overall performance.

The dynamic enhanced Moses version proposed in this work is under continuous development in order to further improve functions, efficiency and effectiveness and augment the field of applicability. Possible extensions are: (i) extending the dynamic language model to handle with cross-phrase n-grams; (ii) defining new scoring functions not related with recency, but to other variables, like human post-editors' confidence; (iii) enabling the contemporary usage of several dynamic phrase tables and language models; (iv) implementing a dynamic lexicalized reordering model mimicking the dynamic phrase table data structure. Some of these ideas are already under development, and they will be made available into Moses toolkit.

Acknowledgements

This research has been supported as part of the MosesCore project[7] (European Commission Grant Number 288487 under the 7th Framework Programme).

Special thanks got to Robert Grabowski, Liane Guillou, Michal Novák, Sorin Slavescu, José Camargo de Souza, with whom I worked during the MT Marathon 2012 to create the first implementation of the dynamic language model, and to Amin M. Farajian, who contributed to the development of the software supporting the dynamic system.

Bibliography

Bellegarda, Jerome R. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93 – 108, 2004.

Bertoldi, Nicola, Mauro Cettolo, and Marcello Federico. Cache-based Online Adaptation for Machine Translation Enhanced Computer Assisted Translation. In *Proceedings of the MT Summit XIV*, pages 35–42, Nice, France, September 2013.

Cettolo, Mauro, Marcello Federico, and Nicola Bertoldi. Mining parallel fragments from comparable texts. In *Proceedings of the International Workshop on Spoken Language Translation*, Paris, France, 2010.

Federico, Marcello, Nicola Bertoldi, and Mauro Cettolo. IRSTLM: an Open Source Toolkit for Handling Large Scale Language Models. In *Proceedings of Interspeech*, pages 1618–1621, Brisbane, Australia, 2008.

Gong, Zhengxian, Min Zhang, and Guodong Zhou. Cache-based document-level statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 909–919, Uppsala, Sweden, 2011. Association for Computational Linguistics.

Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria, 2013. Association for Computational Linguistics.

Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 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, 2007. Association for Computational Linguistics.

Levenberg, Abby and Miles Osborne. Stream-based randomised language models for smt. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2*, pages 756–764, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[7] `http://www.mosescore.eu`

Levenberg, Abby, Chris Callison-Burch, and Miles Osborne. Stream-based translation models for statistical machine translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 394–402, Los Angeles, California, June 2010. Association for Computational Linguistics.

Nepveu, Laurent, Guy Lapalme, Philippe Langlais, and George Foster. Adaptive language and translation models for interactive machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 190–197, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Och, Franz Josef. 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, 2003. Association for Computational Linguistics.

Stolcke, Andreas. Srilm - an extensible language modeling toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing*, Denver, Colorado, 2002.

Talbot, David and Miles Osborne. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, volume 7, pages 512–519, Prague, Czech Republic, 2007. Association for Computational Linguistics.

Tiedemann, Jörg. Context adaptation in statistical machine translation using models with exponentially decaying cache. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 8–15, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Wäschle, Katharina, Patrick Simianer, Nicola Bertoldi, Stefan Riezler, and Marcello Federico. Generative and discriminative methods for online adaptation in smt. In *Proceedings of the MT Summit XIV*, pages 11–18, Nice, France, September 2013.

Appendix A: Communication to the Dynamic System

A xml-based annotation[8] detailed in Tables 2 and 3 has been constructed to communicate with Moses to insert entries into the dynamic language model and phrase table. Note that the Moses parameter "-xml-input" should be set to "inclusive" to enable the parsing of the tag.

a_1	`<dlt cblm="Le visage rustre"/>`
a_2	`<dlt cblm="Le visage rustre de la domination || de la domination || visage"/>`
a_3	`<dlt cblm="Le visage rustre de la domination"/> <dlt cblm="de la domination"/> <dlt cblm="visage"/>`
b_1	`<dlt cblm-file="filename"/>`
b_2	`<dlt cblm-file="filename-1 || filename-2"/>`
c_1	`<dlt cblm-clear-entry="de la domination"/>`
c_2	`<dlt cblm-clear-entry ="de la domination || Le visage rustre || visage"/>`

c_3	`<dlt cblm-clear-entry ="de la domination"/>`
	`<dlt cblm-clear-entry ="Le visage rustre"/>`
	`<dlt cblm-clear-entry ="visage"/>`

d_1	`<dlt cblm-clear-all=""/>`
d_2	`<dlt cblm-command="clear"/>`

Table 2. Annotation to communicate with the dynamic language model. (a_1) insertion of one n-gram; (a_2) contemporary insertion of multiple n-grams; (a_3) sequential insertion of multiple n-grams; (b_1)-(b_2) insertion of n-grams from file(s); (c_1)-(c_3) deletion of single or multiple n-gram; (d_1)-(d_2) full cleanup of the dynamic language model. Double vertical lines separate multiple n-grams and filenames.

The tags can be included anywhere in the text to translate. Nevertheless, Moses first applies the required update to the dynamic language model and phrase table, then translates the actual input text with the updated models. If multiple tags are provided, they are exploited sequentially from left to right. If no text to translate is provided, Moses returns the translation of an empty string, in order to remain synchronized with the input.

The insertion of single or multiple entries are controlled by annotation in examples (a). If multiple entries are inserted with the annotation in examples (a_2), they are inserted contemporarily in the dynamic models and hence they will be associated to

[8] `dlt` stands for Document Level Translation because originally the dynamic models were intended for that task; `cblm` and `cbtm` stand for cache-based language model and translation model, respectively.

a_1	<dlt cbtm="The crude face ||| Le visage rustre"/>
a_2	<dlt cbtm="The crude face of domination ||| Le visage rustre de la domination |||| of domination ||| de la domination |||| face ||| visage"/>
a_3	<dlt cbtm="The crude face of domination ||| Le visage rustre de la domination"/> <dlt cbtm="of domination ||| de la domination"/> <dlt cbtm="face|||visage"/>
b_1	<dlt cbtm-file="filename"/>
b_2	<dlt cbtm-file="filename-1 || filename-2"/>
c_1	<dlt cbtm-clear-option="of domination ||| de la domination"/>
c_2	<dlt cbtm-clear-option="of domination ||| de la domination |||| The crude face ||| Le visage rustre |||| face ||| visage"/>
c_3	<dlt cbtm-clear-option="of domination ||| de la domination"/> <dlt cbtm-clear-option="The crude face ||| Le visage rustre"/> <dlt cbtm-clear-option="face |||| visage"/>
c_4	<dlt cbtm-clear-source="The crude face"/>
c_5	<dlt cbtm-clear-source="The crude face |||| of domination"/>
c_6	<dlt cbtm-clear-source"The crude face"/> <dlt cbtm-clear-source="of domination"/>
d_1	<dlt cbtm-clear-all=""/>
d_2	<dlt cbtm-command="clear"/>

Table 3. Annotation to communicate with the dynamic phrase table. (a_1) insertion of one translation option; (a_2) contemporary insertion of multiple translation options; (a_3) sequential insertion of multiple translation options; (b_1)-(b_2) insertion of translation options from file(s); (c_1)-(c_3) deletion of single or multiple translation options (c_4)-(c_6) deletion of all translation options associated to one or multiple source phrases; (d_1)-(d_2) full cleanup of the dynamic phrase table. Quadruple vertical lines separate phrase pairs; triple vertical lines separate source and target sides of a phrase pair. Double vertical lines separate multiple filenames.

a common age. Instead if separate tags for each entry are used, like in examples (a_3), the entries are inserted sequentially from left to right, and hence they receive different ages, with the right-most entry being the most recent.

The dynamic models can be also updated by loading entries and corresponding ages from one or more files, either in the initialization phase or with the annotation shown in examples (b). File formats for the dynamic language model and phrase table are shown in Table 4.

The deletion of single or multiple entries are controlled by annotations (c); while the overall cleanup of the models is done using the equivalent annotations in examples (d).

```
Le visage rustre ||| 1
la domination ||| 3
de la domination ||| 2
...|||...|||...
```

```
The crude face ||| Le visage rustre ||| 1
domination ||| la domination ||| 3
of domination ||| de la domination ||| 2
of domination ||| la domination ||| 3
...|||...|||...
```

Table 4. Format of the file containing informing data for updating the dynamic language model (above) and phrase table (below). The age is specified for each translation option and for each n-gram. In case of duplicates the last entry is valid.

Appendix B: Configuration of the Dynamic System

Assuming that the dynamic models are used as additional features in Moses, they must be specified in the configuration file, as explained below.

Moreover, the Moses parameter "-xml-input" should be set to "inclusive" to enable the parsing of the tags for updating the dynamic models. The Moses parameter "-no-cache" should be set to disable the caching of translation options; in fact, they and their scores can change over time as the dynamic models change.

Configuration of the Dynamic Language Model

The *feature* and *weights* sections must be modified to properly configure the dynamic language model and to set the corresponding weight.

```
[feature]
DynamicCacheBasedLanguageModel [parameters]

[weights]
DynamicCacheBasedLanguageModel0= <value>
```

The dynamic language model is configured in the *feature* section. The following parameters are currently configurable:

- `cblm-score-type`: scoring function type (see below); default is 0
- `cblm-query-type`: query type; default is 0
- `cblm-max-age`: maximum allowed age of the n-grams; default is 1000

The `cblm-score-type` parameter selects one of the age-dependent functions to score the n-gram stored in the cache of the dynamic language model. Table 5 shows the available scoring functions, which are described in Section 3.3. The `cblm-query-type` parameter selects one of the two modalities of lookup an n-gram in the cache of the dynamic language model described in Section 3.3 either 0 for *AllSubStrings* or 1 for *WholeString*. The `cblm-max-age` parameter sets the maximum age beyond which an n-gram is removed from the cache.

	scoring function type		
0	hyperbola-based reward	10	hyperbola-based penalty
1	power-based reward	11	power-based penalty
2	exponential-based reward	12	exponential-based penalty
3	cosine-based reward		

Table 5. Reward and penalty scoring functions implemented for the dynamic language model and phrase table.

The weight for the dynamic language model is set in the *weights* section.

Configuration of the Dynamic Phrase Table

The *feature*, *weights*, and *mapping* sections must be modified to properly configure the dynamic phrase table and the decoder and to set the corresponding weights.

```
[feature]
PhraseDictionaryDynamicCacheBased [parameters]

[weights]
PhraseDictionaryDynamicCacheBased0= <value>

[mapping]
0 T 0
1 T 1
```

The dynamic phrase table is configured in the *feature* section. As the dynamic phrase table extends the Moses basic object *PhraseDictionary*, all its parameters are still available. Moreover, few parameters are specific to the dynamic phrase table:

- `cbtm-score-type`: select the scoring function (see belov); default is 0
- `cbtm-max-age`: maximum allowed age of the phrase pairs; default is 1000

The `cbtm-score-type` parameter selects one of the scoring functions reported in Table 5 to score the phrase pairs stored in the cache of the dynamic phrase table. The `cbtm-max-age` parameter sets the maximum age beyond which a phrase pair is removed from the cache.

The weight(s) for the score(s) provided by the dynamic phrase table are set in the *weights* section. The number of values set in the *weights* section must coincide with the specified `num-features`.

Finally, in the *mapping* section the decoder must be informed about how many phrase tables are actually exploited. Assuming that the dynamic phrase table is used in addition to one standard phrase table, and that the translation options are fetched from either two, the section is set as shown above.

Address for correspondence:
Nicola Bertoldi
`bertoldi@fbk.eu`
Fondazione Bruno Kessler
via Sommarive 18, Povo, 38123 Trento, Italy