Efficient Handling of N-gram Language Models for Statistical Machine Translation

Marcello Federico
Fondazione Bruno Kessler - IRST
I-38050 Trento, Italy
federico@itc.it

Mauro Cettolo
Fondazione Bruno Kessler - IRST
I-38050 Trento, Italy
cettolo@itc.it

Abstract

Statistical machine translation, as well as other areas of human language processing, have recently pushed toward the use of large scale n-gram language models. This paper presents efficient algorithmic and architectural solutions which have been tested within the Moses decoder, an open source toolkit for statistical machine translation. Experiments are reported with a high performing baseline, trained on the Chinese-English NIST 2006 Evaluation task and running on a standard Linux 64-bit PC architecture. Comparative tests show that our representation halves the memory required by SRI LM Toolkit, at the cost of 44% slower translation speed. However, as it can take advantage of memory mapping on disk, the proposed implementation seems to scale-up much better to very large language models: decoding with a 289-million 5-gram language model runs in 2.1Gb of RAM.

1 Introduction

In recent years, we have seen an increasing interest toward the application of n-gram Language Models (LMs) in several areas of computational linguistics (Lapata and Keller, 2006), such as machine translation, word sense disambiguation, text tagging, named entity recognition, etc. The original framework of n-gram LMs was principally automatic speech recognition, under which most of the standard LM estimation techniques (Chen and Goodman, 1999) were developed. Nowadays, the availability of larger and larger text corpora is stressing the need for efficient data structures and algorithms to estimate, store and access LMs. Unfortunately, the rate of progress in computer technology seems for the moment below the space requirements of such huge LMs, at least by considering standard lab equipment.

Statistical machine translation (SMT) is today one of the research areas that, together with speech recognition, is pushing mostly toward the use of huge n-gram LMs. In the 2006 NIST Machine Translation Workshop (NIST, 2006), best performing systems employed 5-grams LMs estimated on at least 1.3 billion-word texts. In particular, Google Inc. presented SMT results with LMs trained on 8 trillion-word texts, and announced the availability of n-gram statistics extracted from one trillion of words. The n-gram Google collection is now publicly available through LDC, but their effective use requires either to significantly expand computer memory, in order to use existing tools (Stolcke, 2002), or to develop new ones.

This work presents novel algorithms and data structures suitable to estimate, store, and access very large LMs. The software has been integrated into a popular open source SMT decoder called Moses.[1] Experimental results are reported on the Chinese-English NIST task, starting from a quite well-performing baseline, that exploits a large 5-gram LM.

This paper is organized as follows. Section 2 presents techniques for the estimation and represen-

[1] http://www.statmt.org/moses/

Proceedings of the Second Workshop on Statistical Machine Translation, pages 88–95,
Prague, June 2007.

tation in memory of n-gram LMs that try to optimize space requirements. Section 3 describes methods implemented in order to efficiently access the LM at run time, namely by the `Moses` SMT decoder. Section 4 presents a list of experiments addressing specific questions on the presented implementation.

2 Language Model Estimation

LM estimation starts with the collection of n-grams and their frequency counters. Then, smoothing parameters are estimated (Chen and Goodman, 1999) for each n-gram level; infrequent n-grams are possibly pruned and, finally, a LM file is created containing n-grams with probabilities and back-off weights.

2.1 N-gram Collection

Clearly, a first bottleneck of the process might occur if all n-grams have to be loaded in memory. This problem is overcome by splitting the collection of n-grams statistics into independent steps and by making use of an efficient data-structure to collect and store n-grams. Hence, first the dictionary of the corpus is extracted and split into K word lists, balanced with respect to the frequency of the words. Then, for each list, only n-grams whose first word belongs to the list are extracted from the corpus. The value of K is determined empirically and should be sufficiently large to permit to fit the partial n-grams into memory. The collection of each subset of n-grams exploits a dynamic prefix-tree data structure shown in Figure 1. It features a table with all collected 1-grams, each of which points to its 2-gram successors, namely the 2-grams sharing the same 1-gram prefix. All 2-gram entries point to all their 3-gram successors, and so on. Successor lists are stored in memory blocks allocated on demand through a memory pool. Blocks might contain different number of entries and use 1 to 6 bytes to encode frequencies. In this way, a minimal encoding is used in order to represent the highest frequency entry of each block. This strategy permits to cope well with the high sparseness of n-grams and with the presence of relatively few highly-frequent n-grams, that require counters encoded with 6 bytes.

The proposed data structure differs from other implementations mainly in the use of dynamic allocation of memory required to store frequencies of n-grams. In the structure proposed by (Wessel et al., 1997) counters of n-grams occurring more than once are stored into 4-byte integers, while singleton n-grams are stored in a special table with no counters. This solution permits to save memory at the cost of computational overhead during the collection of n-grams. Moreover, for historical reasons, this work ignores the issue with huge counts. In the SRILM toolkit (Stolcke, 2002), n-gram counts are accessed through a special class type. Counts are all represented as 4-byte integers by applying the following trick: counts below a given threshold are represented as unsigned integers, while those above the threshold, which are typically very sparse, correspond indeed to indexes of a table storing their actual value. To our opinion, this solution is ingenious but less general than ours, which does not make any assumption about the number of different high order counts.

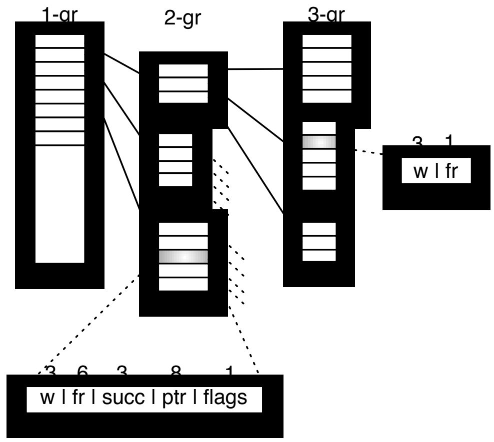

Figure 1: Dynamic data structure for storing n-grams. Blocks of successors are allocated on demand and might vary in the number of entries (depth) and bytes used to store counters (width). Size in bytes is shown to encode words (w), frequencies (fr), and number of (succ), pointer to (ptr) and table type of (flags) successors.

2.2 LM Smoothing

For the estimation of the LM, a standard interpolation scheme (Chen and Goodman, 1999) is applied in combination with a well-established and simple smoothing technique, namely the Witten-Bell linear discounting method (Witten and Bell, 1991). Smoothing of probabilities up from 2-grams is performed separately on each subset of n-grams.

For example, smoothing statistics for a 5-gram (v, w, x, y, z) are computed by means of statistics that are local to the subset of n-grams starting with v. Namely, they are the counters $N(v, w, x, y, z)$, $N(v, w, x, y)$, and the number $D(v, w, x, y)$ of different words observed in context (v, w, x, y).

Finally, K LM files are created, by just reading through the n-gram files, which are indeed not loaded in memory. During this phase pruning of infrequent n-grams is also permitted. Finally, all LM files are joined, global 1-gram probabilities are computed and added, and a single large LM file, in the standard ARPA format (Stolcke, 2002), is generated.

We are well aware that the implemented smoothing method is below the state-of-the-art. However, from one side, experience tells that the gap in performance between simple and sophisticated smoothing techniques shrinks when very large corpora are used; from the other, the chosen smoothing method is very suited to the kind of decomposition we are applying to the n-gram statistics. In the future, we will nevertheless address the impact of more sophisticated LM smoothing on translation performance.

2.3 LM Compilation

The final textual LM can be compiled into a binary format to be efficiently loaded and accessed at runtime. Our implementation follows the one adopted by the CMU-Cambridge LM Toolkit (Clarkson and Rosenfeld, 1997) and well analyzed in (Whittaker and Raj, 2001). Briefly, n-grams are stored in a data structure which privileges memory saving rather than access time. In particular, single components of each n-gram are searched, via binary search, into blocks of successors stored contiguously (Figure 2). Further improvements in memory savings are obtained by quantizing both back-off weights and probabilities.

2.4 LM Quantization

Quantization provides an effective way of reducing the number of bits needed to store floating point variables. (Federico and Bertoldi, 2006) showed that best results were achieved with the so-called *binning method*. This method partitions data points into uniformly populated intervals or bins. Bins are filled in in a greedy manner, starting from the lowest value. The center of each bin corresponds to the mean value

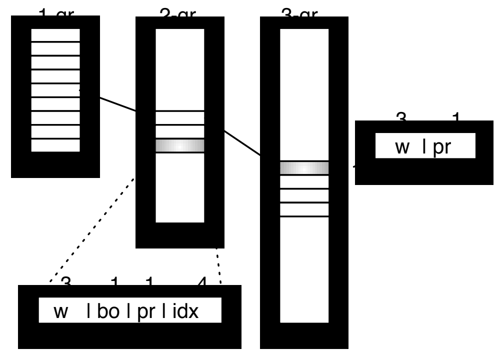

Figure 2: Static data structure for LMs. Number of bytes are shown used to encode single words (w), quantized back-off weights (bo) and probabilities (pr), and start index of successors (idx).

of all its points. Quantization is applied separately at each n-gram level and distinctly to probabilities or back-off weights. The chosen level of quantization is 8 bits (1 byte), that experimentally showed to introduce negligible loss in translation performance.

The quantization algorithm can be applied to any LM represented with the ARPA format. Quantized LMs can also be converted into a binary format that can be efficiently uploaded at decoding time.

3 Language Model Access

One motivation of this work is the assumption that efficiency, both in time and space, can be gained by exploiting peculiarities of the way the LM is used by the hosting program, i.e. the SMT decoder. An analysis of the interaction between the decoder and the LM was carried out, that revealed some important properties. The main result is shown in Figure 3, which plots all calls to a 3-gram LM by `Moses` during the translation from German to English of the following text, taken from the Europarl task:

```
ich bin kein christdemokrat und
glaube daher nicht an wunder .
doch ich möchte dem europäischen
parlament , so wie es gegenwürtig
beschaffen ist , für seinen
grossen beitrag zu diesen arbeiten
danken.
```

Translation of the above text requires about 1.7 million calls of LM probabilities, that however involve only 120,000 different 3-grams. The plot shows typical locality phenomena, that is the decoder tends to

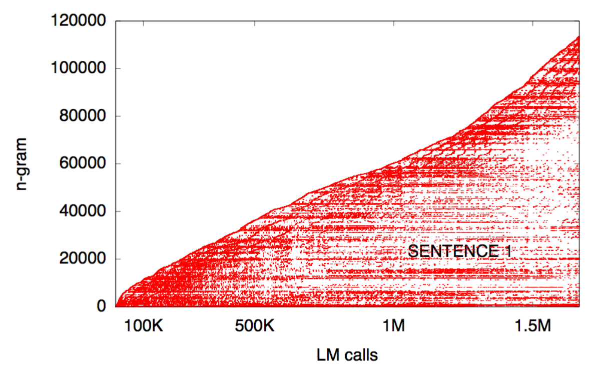

Figure 3: LM calls during translation of a German text: each point corresponds to a specific 3-gram.

access the LM n-grams in nonuniform, highly localized patterns. Locality is mainly temporal, namely the first call of an n-gram is easily followed by other calls of the same n-gram. This property suggests that gains in access speed can be achieved by exploiting a cache memory in which to store already called n-grams. Moreover, the relatively small amount of involved n-grams makes viable the access of the LM from disk on demand. Both techniques are briefly described.

3.1 Caching of probabilities

In order to speed-up access time of LM probabilities different cache memories have been implemented through the use of hash tables. Cache memories are used to store all final n-gram probabilities requested by the decoder, LM states used to recombine theories, as well as all partial n-gram statistics computed by accessing the LM structure. In this way, the need of performing binary searches, at every level of the LM tables, is reduced at a minimum.

All cache memories are reset before decoding each single input set.

3.2 Memory Mapping

Since a limited collection of all n-grams is needed to decode an input sentence, the LM is loaded on demand from disk. The data structure shown in Figure 2 permits indeed to efficiently exploit the so-called *memory mapped* file access.[2] Memory mapping basically permits to include a file in the address space of a process, whose access is managed as virtual memory (see Figure 4).

[2]POSIX-compliant operating systems and Windows support some form of memory-mapped file access.

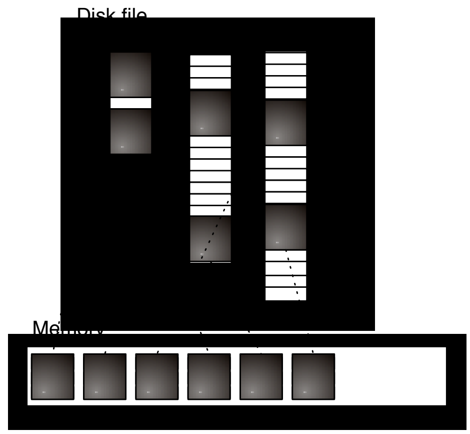

Figure 4: Memory mapping of the LM on disk. Only the memory pages (grey blocks) of the LM that are accessed while decoding the input sentence are loaded in memory.

During decoding of a sentence, only those n-grams, or better memory pages, of the LM that are actually accessed are loaded into memory, which results in a significant reduction of the resident memory space required by the process. Once the decoding of the input sentence is completed, all loaded pages are released, so that resident memory is available for the n-gram probabilities of the following sentence. A remarkable feature is that memory-mapping also permits to share the same address space among multiple processes, so that the same LM can be accessed by several decoding processes (running on the same machine).

4 Experiments

In order to assess the quality of our implementation, henceforth named IRSTLM, we have designed a suite of experiments with a twofold goal: from one side the comparison of IRSTLM against a popular LM library, namely the SRILM toolkit (Stolcke, 2002); from the other, to measure the actual impact of the implementation solution discussed in previous sections. Experiments were performed on a common statistical MT platform, namely `Moses`, in which both the IRSTLM and SRILM toolkits have been integrated.

The following subsection lists the questions

set	type	|W|	
		source	target
large	parallel	83.1M	87.6M
giga	monolingual	-	1.76G
NIST 02	dev	23.7K	26.4K
NIST 03	test	25.6K	28.5K
NIST 04	test	51.0K	58.9K
NIST 05	test	31.2K	34.6K
NIST 06 nw	test	18.5K	22.8K
NIST 06 ng	test	9.4K	11.1K
NIST 06 bn	test	12.0K	13.3K

Table 1: Statistics of training, dev. and test sets. Evaluation sets of NIST campaigns include 4 references: in table, average lenghts are provided.

which our experiments aim to answer.

Assessing Questions

1. Is LM estimation feasible for large amounts of data?
2. How does IRSTLM compare with SRILM w.r.t.:
   (a) decoding speed?
   (b) memory requirements?
   (c) translation performance?
3. How does LM quantization impact in terms of
   (a) memory consumption?
   (b) decoding speed?
   (c) translation performance?
   (d) tuning of decoding parameters?
4. What is the impact of caching on decoding speed?
5. What are the advantages of memory mapping?

Task and Experimental Setup

The task chosen for our experiments is the translation of news from Chinese to English, as proposed by the NIST MT Evaluation Workshop of 2006.[3] A translation system was trained according to the *large-data* condition. In particular, all the allowed bilingual corpora have been used for estimating the phrase-table. The target side of these texts was also employed for the estimation of three 5-gram LMs, henceforth named *large*. In particular, two LMs were estimated with the SRILM toolkit by pruning singletons events and by employing the Witten-Bell and the absolute discounting (Kneser and Ney, 1995) smoothing methods; the shorthand for these two LMs will be “lrg-sri-wb” and “lrg-sri-kn”, respectively. Another large LM was estimated with the IRSTLM toolkit, by employing the only smoothing method available in the package (Witten-Bell) and by pruning singletons n-grams; its shorthand will be “lrg”. An additional, much larger, 5-gram LM was instead trained with the IRSTLM toolkit on the so-called English Gigaword corpus, one of the allowed monolingual resources for this task.

Automatic translation was performed by means of `Moses` which, among other things, permits the contemporary use of more LMs, feature we exploited in our experiments as specified later.

Optimal interpolation weights for the log-linear model were estimated by running a minimum error training algorithm, available in the `Moses` toolkit, on the evaluation set of the NIST 2002 campaign. Tests were performed on the evaluation sets of the successive campaigns (2003 to 2006). Concerning the NIST 2006 evaluation set, results are given separately for three different types of texts, namely newswire (nw) and newsgroup (ng) texts, and broadcast news transcripts (bn).

Table 1 gives figures about training, development and test corpora, while Table 2 provides main statistics of the estimated LMs.

LM	millions of				
	1-gr	2-gr	3-gr	4-gr	5-gr
lrg-sri-kn	0.3	5.2	5.9	7.1	6.8
lrg-sri-wb	0.3	5.2	6.4	7.8	6.8
lrg	0.3	5.3	6.6	8.4	8.0
giga	4.5	64.4	127.5	228.8	288.6

Table 2: Statistics of LMs.

MT performance are provided in terms of case-insensitive BLEU and NIST scores, as computed with the NIST scoring tool. For time reasons, the decoder run with monotone search; preliminary experiments showed that this choice does not affect comparison of LMs. Reported decoding speed is the elapsed real time measured with the Linux/UNIX `time` command divided by the number of source words to be translated. dual Intel/Xeon

[3] `www.nist.gov/speech/tests/mt/`

CPU 3.20GHz with 8Gb RAM. Experiments run on dual Intel/Xeon CPUs 3.20GHz/8Gb RAM.

4.1 LM estimation

First of all, let us answer the question (number 1) on the feasibility of the procedure for the estimation of huge LMs. Given the amount of training data employed, it is worth to provide some details about the estimation process of the "giga" LM. According to the steps listed in Section 2.1, the whole dictionary was split into $K = 14$ frequency balanced lists. Then, 5-grams beginning with words from each list were extracted and stored. Table 3 shows some figures about these dictionaries and 5-gram collections. Note that the dictionary size increases with the list index: this means only that more frequent words were used first. This stage run in few hours with 1-2Gb parallel processes.

list index	dictionary size	number of 5-grams: observed	distinct	non-singletons
0	4	217M	44.9M	16.2M
1	11	164M	65.4M	20.7M
2	8	208M	85.1M	27.0M
3	44	191M	83.0M	26.0M
4	64	143M	56.6M	17.8M
5	137	142M	62.3M	19.1M
6	190	142M	64.0M	19.5M
7	548	142M	66.0M	20.1M
8	783	142M	63.3M	19.2M
9	1.3K	141M	67.4M	20.2M
10	2.5K	141M	69.7M	20.5M
11	6.1K	141M	71.8M	20.8M
12	25.4K	141M	74.5M	20.9M
13	4.51M	141M	77.4M	20.6M
total	4.55M	2.2G	951M	289M

Table 3: Estimation of the "giga" LM: dictionary and 5-gram statistics ($K = 14$).

The actual estimation of the LM was performed with the scheme presented in Section 2.2. For each collection of non-singletons 5-grams, a sub-LM was built by computing smoothed n-gram ($n = 1 \cdots 5$) probabilities and interpolation parameters. Again, by exploiting parallel processing, this phase took only few hours on standard HW resources. Finally, sub-LMs were joined in a single LM, which can be stored in two formats: (i) the standard textual ARPA format, and (ii) the binary format of Section 2.3. In addition, LM probabilities can be quantized according to the procedure of Section 2.4.

LM	format	quantization	file size
lrg-sri-kn	textual	n	893Mb
lrg-sri-wb	textual	n	952Mb
lrg	textual	n	1088Mb
		y	789Mb
	binary	n	368Mb
		y	220Mb
giga	textual	n	28.0Gb
		y	21.0Gb
	binary	n	8.5Gb
		y	5.1Gb

Table 4: Figures of LM files.

The estimation of the "lrg-sri" LMs, performed by means of the SRILM toolkit, took about 15 minutes requiring 5Gb of memory. The "lrg" LM was estimated as the "giga" LM in about half an hour demanding only few hundreds of Mb of memory.

Table 4 lists the size of files storing various versions of the "large" and "giga" LMs which differ in format and/or type.

4.2 LM run-time usage

Tables 5 and 6 shows BLEU and NIST scores, respectively, measured on test sets for each specific LM configuration. The first two rows of the two tables regards runs of `Moses` with the SRILM, that uses "lrg-sri" LMs. The other rows refer to runs of `Moses` with IRSTLM, either using LM "lrg" only, or both LMs, "lrg" and "giga". LM quantization is marked by a "q".

Finally, in Table 7 figures about the decoding processes are recorded. For each LM configuration, the process size, both virtual and resident, is provided together with the average time required for translating a source word with/without the activation of the caching mechanism described in Section 3.1. It is to worth noticing that the "giga" LM (both original and quantized) is loaded through the memory mapping service presented in Section 3.2.

Table 7 includes most of the answers to question number 2:

2.a Under the same conditions, `Moses` running with SRILM permits almost double faster

LM	NIST test set					
	03	04	05	06 nw	06 ng	06 bn
lrg-sri-kn	28.74	30.52	26.99	29.28	23.47	27.27
lrg-sri-wb	28.05	29.86	26.52	28.37	23.13	26.37
lrg	28.49	29.84	26.97	28.69	23.28	26.70
q-lrg	28.05	29.66	26.48	28.58	22.64	26.05
lrg+giga	30.77	31.93	29.09	29.74	24.39	28.50
q-lrg+q-giga	30.42	31.47	28.62	29.76	24.28	28.23

Table 5: BLEU scores on NIST evaluation sets for different LM configurations.

LM	NIST test set					
	03	04	05	06 nw	06 ng	06 bn
lrg-sri-kn	8.73	9.29	8.47	8.98	7.81	8.52
lrg-sri-wb	8.52	9.14	8.27	8.96	7.90	8.34
lrg	8.73	9.21	8.45	8.95	7.82	8.47
q-lrg	8.60	9.11	8.32	8.88	7.73	8.31
lrg+giga	9.08	9.49	8.80	8.92	7.86	8.66
q-lrg+q-giga	8.93	9.38	8.65	9.05	7.99	8.60

Table 6: NIST scores on NIST evaluation sets for different LM configurations.

translation than IRSTLM (13.33 vs. 6.80 words/s). Anyway, IRSTLM can be sped-up to 7.52 words/s by applying caching.

2.b IRSTLM requires about half memory than SRILM for storing an equivalent LM during decoding. If the LM is quantized, the gain is even larger. Concerning file sizes (Table 4), the size of IRSTLM binary files is about 30% of the corresponding textual versions. Quantization further reduces the size to 20% of the original textual format.

2.c Performance of IRSTLM and SRILM on the large LMs smoothed with the same method are comparable, as expected (see entries “lrg-sri-wb” and “lrg” of Tables 5 and 6). The small differences are due to different probability values assigned by the two libraries to out-of-vocabulary words.

Concerning quantization, gains in terms of memory space (question 3.a) have already been highlighted (see answer 2.b). For the remaining points:

3.b comparing “lrg” vs. “q-lrg” rows and “lrg+giga” vs. “q-lrg+q-giga” rows of Table 7, it results that quantization allows only a marginal decoding time reduction (1-3%)

3.c comparing the same rows of Tables 5 and 6, it can be claimed that quantization doesn’t affect translation performance in a significant way

3.d no specific training of decoder weights is required since the original LM and its quantized version are equivalent. For example, by translating the NIST 05 test set with the weights estimated on the “lrg+giga” configuration, the following BLEU/NIST scores are got: 28.99/8.79 with the “q-lrg+q-giga” LMs, 29.09/8.80 with the “lrg+giga” LMs (the latter scores are also given in Tables 5 and 6). Employing weights estimated on “q-lrg+q-giga” scores are: 28.58/8.66 with “lrg+giga” LMs, 28.62/8.65 with “q-lrg+q-giga” LMs (again also in Tables 5 and 6). Also on other test sets differences are negligible.

LM	process size virtual	process size resident	caching	dec. speed (src w/s)
lrg-sri-kn/wb	1.2Gb	1.2Gb	-	13.33
lrg	750Mb	690Mb	n	6.80
			y	7.42
q-lrg	600Mb	540Mb	n	6.99
			y	7.52
lrg+giga	9.9Gb	2.1Gb	n	3.52
			y	4.28
q-lrg+q-giga	6.8Gb	2.1Gb	n	3.64
			y	4.35

Table 7: Process size and decoding speed with/wo caching for different LM configurations.

Table 7 answers the question number 4 on caching, by reporting the decoding speed-up due to this mechanism: a gain of 8-9% is observed on “lrg” and “q-lrg” configurations, of 20-21% in case also “giga/q-giga” LMs are employed.

The answer to the last question is that thanks to the memory mapping mechanism it is possible run `Moses` with huge LMs, which is expected to improve performance. Tables 5 and 6 provide quantitative support to the statement. In fact, a gain of 1-2 absolute BLEU was measured on different test sets when “giga” LM was employed in addition to

	NIST test set					
	03	04	05	06 nw	06 ng	06 bn
BLEU						
ci	33.62	35.04	31.92	32.74	26.18	32.43
cs	31.44	32.99	29.95	30.49	24.35	31.10
NIST						
ci	9.27	9.75	9.00	9.24	8.00	8.97
cs	8.88	9.40	8.64	8.82	7.69	8.77

Table 8: Case insensitive (ci) and sensitive (cs) scores of the best performing system.

"lrg" LM. The SRILM-based decoder would require a process of about 30Gb to load the "giga" LM; on the contrary, the virtual size of the IRSTLM-based decoder is 6.8Gb, while the actual resident memory is only 2.1Gb.

4.3 Best Performing System

Experimental results discussed so far are not the best we are able to get. In fact, the adopted setup fixed the monotone search and the use of no reordering model. Then, in order to allow a fair comparison of the IRSTLM-based `Moses` system with the ones participating to the NIST MT evaluation campaigns, we have (i) set the maximum reordering distance to 6 and (ii) estimated a lexicalized reordering model on the large parallel data by means of the training option "orientation-bidirectional-fe".

Table 8 shows BLEU/NIST scores measured on test sets by employing the IRSTLM-based `Moses` with this setting and employing "q-lrg+q-giga" LMs. It ranks at the top 5 systems (out of 24) with respect to the results of the NIST 06 evaluation campaign.

5 Conclusions

We have presented a method for efficiently estimating and handling large scale n-gram LMs for the sake of statistical machine translation. LM estimation is performed by splitting the task with respect to the initial word of the n-grams, and by merging the resulting sub-LMs. Estimated LMs can be quantized and compiled in a compact data structure. During the search, LM probabilities are cached and only the portion of effectively used LM n-grams is loaded in memory from disk. This method permits indeed to exploit locality phenomena shown by the search algorithm when accessing LM probabilities. Results show an halving of memory requirements, at the cost of 44% slower decoding speed. In addition, loading the LM on demand permits to keep the size of memory allocated to the decoder nicely under control.

Future work will investigate the way for including more sophisticated LM smoothing methods in our scheme and will compare IRSTLM and SRILM toolkits on increasing size training corpora.

6 Acknowledgments

This work has been funded by the European Union under the integrated project TC-STAR - Technology and Corpora for Speech-to-Speech Translation - (IST-2002-FP6-506738, http://www.tc-star.org).

References

S.F. Chen and J. Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 4(13):359–393.

P. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the CMU–cambridge toolkit. In *Proc. of Eurospeech*, pages 2707–2710, Rhodes, Greece.

M. Federico and N. Bertoldi. 2006. How many bits are needed to store probabilities for phrase-based translation? In *Proc. of the Workshop on Statistical Machine Translation*, pages 94–101, New York City, June. Association for Computational Linguistics.

R. Kneser and H. Ney. 1995. Improved backing-off for m-gram language modeling. In *Proc. of ICASSP*, volume 1, pages 181–184, Detroit, MI.

M. Lapata and F. Keller. 2006. Web-based models for natural language processing. *ACM Transactions on Speech and Language Processing*, 1(2):1–31.

NIST. 2006. Proc. of the NIST MT Workshop. Washington, DC. NIST.

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

F. Wessel, S. Ortmanns, and H. Ney. 1997. Implementation of word based statistical language models. In *Proc. SQEL Workshop on Multi-Lingual Information Retrieval Dialogs*, pages 55–59, Pilsen, Czech Republic.

E. W. D. Whittaker and B. Raj. 2001. Quantization-based Language Model Compression. In *Proc. of Eurospeech*, pages 33–36, Aalborg.

I. H. Witten and T. C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. Inform. Theory*, IT-37(4):1085–1094.