Statistical Machine Translation: Rapid Development with Limited Resources

George Foster, Simona Gandrabur, Philippe Langlais, Pierre Plamondon, Graham Russell and Michel Simard

RALI/DIRO, Université de Montréal
Montreal, Canada
www-rali.iro.umontreal.ca

Abstract

We describe an experiment in rapid development of a statistical machine translation (SMT) system from scratch, using limited resources: under this heading we include not only training data, but also computing power, linguistic knowledge, programming effort, and absolute time.

1 Introduction

A subject of current interest in the machine translation (MT) world, in the United States in particular, is the rapid development of systems for novel language pairs. This trend is marked by a number of assumptions. Often, at least one of the languages in question is taken to be previously unknown to system developers, who must either acquire the necessary knowledge and technology or devise methods that will mitigate the effects of their absence. Similarly, for many of the candidate languages there exists relatively little in the way of suitable training material, thus restricting the scope of purely statistical approaches.

This is quite uncontroversial, and may indeed be viewed as a welcome counter to excessive reliance on exceptionally large and 'clean' parallel corpora. However, other limiting factors are less frequently discussed in this context: the process of training and testing sophisticated translation models is extremely expensive, consuming important amounts of cleaning efforts, computing power, programming effort, and time. This paper reports an experiment one of whose aims was to answer the following question: "What progress can a small team of developers expect to achieve in creating a statistical MT system for an unfamiliar language, using only data and technology readily available in-house, or at short notice from external sources?" Options were constrained by the fact that the source language was Chinese, of which the development team had no knowledge, that no in-house Chinese-specific tools existed, and that available computing resources consisted of Pentium-4 class PCs with a maximum of 1Gb RAM.

This work was conducted within the NIST 2003 machine translation evaluation task,[1] specifically the Chinese-to-English task using the large but limited data set.[2] Advantages included ready access to sufficient data, an automatic scoring mechanism, and the possibility of comparison with other systems engaged on the same task.

2 The NIST Experience

At the outset, what motivated us to participate in the NIST evaluation was a desire to learn what was really involved in putting together a working statistical MT system. Deadlines appeared reassuringly distant and we had a plan to achieve good performance despite the limitations mentioned in the previous section. We put our hope into a rescoring approach built on top of a roughly state-of-the-art translation model such as IBM Model 4. Rescoring has been extensively used in automatic speech recognition (ASR) on n-best lists or word-graphs (Ortmanns et al., 1997; Rose and Riccardi, 1999), and has more recently been proposed for use in SMT (Och and Ney, 2002; Soricut et al., 2002; Ueffing et al., 2002).

The first step was to install the necessary packages, train translation and language models, and begin work on decoders for IBM Model 4. We also designed a rescoring infrastructure that could host any component able to return a score on a source-target sentence pair. Up to this point we were on familiar territory, since the work was based on a clean and well aligned bitext derived in-house from the Canadian Hansard.

By the time the training corpora were made available, a good deal of code had been written, if not fully tested. We were confronted by an obvious corollary of the statistical MT paradigm: a word-based approach to SMT can only be neutral with regard to the languages under consideration to the extent that adequately tokenized training and test corpora exist. Section 3 describes the difficulties we experienced in this area.

1 http://www.nist.gov/speech/tests/mt/. This and other URLs cited here are valid as of late July 2003.

2 Details of the corpora can be found via http://www.nist.gov/speech/tests/mt/resources/.

As time quickly passed we rapidly learned the second lesson of this exercise: building a statistical MT engine, even largely from pre-existing components, is by no means straightforward. Several unexpected problems arose which in retrospect are quite interesting. First, training very large models with publicly available packages is feasible only to the extent that the corresponding memory demands can be met (see section 4.1). Second, dealing with multiple decoders inevitably complicates development. However, thanks to the multiple decoder strategy suggested by Germann et al. (2001) we managed to get a decent IBM 4 decoder. Third, we observed that decoding in itself involves a compromise between quality of results and the length of the delay before results are emitted: tuning for performance is a delicate and time-consuming process.

The immediate aim of the NIST exercise we participated in was to translate 919 Chinese sentences (length varying from 5 to 101 words, with an average of 29) into English. Additionally, the rescoring strategy required the translation of some 20,000 sentences for the purpose of training the rescoring layer. In the event, slower than expected progress here meant that rescoring could not in fact be applied before the NIST submission deadline arrived, and the 919 test sentences were submitted in their raw state.

In the hope that the lessons learned during this experience may benefit other teams tempted to engage in the exercise of building a statistical MT system from scratch, the following sections are an attempt to describe what to expect at each stage in the development of an SMT engine. More precisely, section 3 discusses the non-rewarding but vital task of preprocessing the available corpora. The different decoders we implemented are presented in section 4 and their performance compared in section 5. In the calmer post-NIST environment a second experiment was conducted in order to measure the impact of the training corpus on translation performance. We report the result of this study (referred in the following as the `take2` experiment) in section 6. Finally, section 7 summarizes the major lessons learned during this exercise.

3 Corpus Preprocessing

Rather than the simple formality we had hoped for, corpus preparation turned out to demand almost one man-month of effort. This excludes time spent on the several training runs which had to be aborted when preprocessing errors were discovered. The lesson here is that this phase of work requires realism concerning the initial state of the data, a judicious mixture of automatic and semi-automatic approaches, and either close attention to quality control or being prepared to repeat the process until it is correct. Major difficulties are briefly outlined below.

3.1 Collection

Training data for the NIST task, in the form of a range of Chinese-English corpora, was distributed by the Linguistic Data Consortium (LDC).[3] Nevertheless, the collection process was not entirely trivial, since a certain amount of interaction with the LDC was necessary before a definitive list of texts could be obtained.

Individual corpora were supplied in a surprising variety of formats; this had the effect of multiplying the effort expended on conversion. Similarly, since the original Chinese texts were produced in Hong Kong, Taiwan and the PRC, the variations in character set and encoding employed in the different language communities also posed some problems.

3.2 Segmentation and Tokenization

The treatment of the English texts was accomplished with the help of existing in-house facilities. As expected, most of the Chinese texts lacked annotations indicating sentence and word boundaries, and so mechanisms were devised to add these. Sentence boundaries were identified by a table-driven scanner sensitive to sentence-final punctuation.

Word boundaries were inserted by means of a revised version of the `mansegment` program supplied by the LDC. Its basic idea is to employ a unigram word model within a dynamic programming framework to produce the most probable token segmentation over a given sequence of characters. It was found that performance in terms of both speed and accuracy could be improved significantly by presegmenting the initial character sequence in order to isolate punctuation characters, and applying the probabilistic component only between these positions.

3.3 Sentence Alignment

For the NIST exercise (`take1`), only pre-aligned texts were used. Some regions acknowledged by the supplier to be potentially unreliable were omitted. No systematic manual checking of alignment quality was performed, and the few instances of obvious errors (largely due to faulty segmentation) which came to our attention were ignored. We call this corpus the `take1` corpus.

When time permitted, following the NIST exercise, we conducted a second experiment in which non-aligned corpora were automatically aligned at the sentence level by an in-house aligner. In the hope of improving the quality of the bitext, we provided the aligner with a bilingual dictionary extracted from the translation model trained on the `take1` corpus, frequent and likely associations ac-

3 `http://www.ldc.upenn.edu/`

cording to the model being retained as entries in the lexicon. Certain alignments were filtered out: those (n-0, 0-n) with empty source or target, and those in which the disparity between source and target lengths exceeded an ad hoc threshold.

3.4 Sentence Length

The program selected for training translation models (see section 4) places a limit of 40 tokens on the length of a source-language sentence. Tokenization of the training corpus produced a high proportion of sentences longer than this, and a number of solutions were considered. Note that the 40-token limit is present for good reasons; while it could in principle have been raised or removed altogether, this would have led to increased memory usage, an area in which performance was already borderline.

Rather than arbitrarily discarding or truncating the affected data it was decided for the `take1` exercise to split each long source or target sentence into a sequence of 'pseudo-sentences', none of which would exceed 40 tokens in length, to be presented to the training component individually.

Consistent with the knowledge-poor context of this work, no Chinese parser or chunker was available to support this process. Instead, a heuristic approach was adopted, in which input sentences were split either immediately before or immediately after certain classes of punctuation; the process also took into account different 'strengths' of punctuation, and aimed to minimize variation in the length of the resulting pseudo-sentences. In the rare cases where no suitable punctuation existed, sentences were split at an arbitrary token boundary.

For the `take2` experiments, however, we decided to simplify this process by discarding segment pairs for which the source sentence was longer than 40 words.

4 Translation Engines

The problem of statistical translation can be viewed as an optimization problem where, given a source string $F = \langle f_1, \ldots, f_j \rangle$, a translation model $P_{\text{tm}}(\cdot)$ and a language model $P_{\text{lm}}(\cdot)$, we try to find a target string $E = \langle e_1, \ldots, e_i \rangle$ that maximizes the joint probability

$$P_{\text{lm}}(E) * P_{\text{tm}}(F \mid E).$$

As such, it is an instance of the noisy channel approach (Brown et al., 1993), in which an output signal is 'decoded' in order to recover the original input; algorithms which perform this task are known as decoders.

The probabilities $P_{\text{lm}}(\cdot)$ and $P_{\text{tm}}(\cdot)$ are derived by training on monolingual and bilingual corpora. The language model used in this work was a trigram model trained by an existing in-house package, and the translation models considered were the IBM model 2, also trained using an in-house tool, and IBM model 4, trained using the GIZA++ package (Och and Ney, 2000).[4]

One important choice for rescoring is how to represent the sets of translations that are output from the base model. There are at least two possibilities: n-best lists—explicit enumerations of the candidate translations—and word graphs (Ueffing et al., 2002), which are capable of storing much larger sets of translations implicitly. Word graphs are potentially more powerful than n-best lists, but they are also more complex to implement. Furthermore, they constrain the rescoring layer to respect the factorization inherent in the graph. Because of these problems, we chose a simple n-best list representation.

4.1 Crucial Details

It is certainly naïve to assume that any package as complex as GIZA++ (which is nevertheless exceptionally well written and documented) can be ready for full-scale use immediately upon installation. In retrospect, we estimate that around 3 weeks were required to master input/output formats, including the time to write wrappers for the model data structures. Establishing the limits of the package (maximum input size, etc.) took at least another week of monitoring, excluding computation time.

We soon found that using GIZA++ to train a translation model on a corpus of more than one million sentence pairs was impractical. Beyond this point, memory is saturated and the system is forced to swap, drastically increasing the time required.

IBM model 4 conditions the distortion probabilities on the class of the centroid source and target words. To acquire these classes, we used the program `mkcls`.[5] In contrast to the bilingual procedure described by Och (1999), for which no ready-made solution was immediately available, this permits only the creation of monolingual classes. Using this program, source and target vocabulary were processed into 50 classes; this required around ten hours of computation.

In total, the full training of an IBM model 4 on a corpus of around one million sentence pairs requires two to three days of computation on a 1GB memory Pentium 4 desk computer.

4.2 Multiple Decoders

Three different decoders, all previously described in the statistical MT literature, were implemented. One advantage of having several decoders at one's disposal is that certain bugs may be detected by comparing the trans-

4 `http://www-i6.informatik.rwth-aachen.de/Colleagues/och/software/GIZA++.html`. GIZA++ also provides IBM model 2, but it was felt that faster progress could be made with the more familiar alternative.

5 `http://www-i6.informatik.rwth-aachen.de/Colleagues/och/software/mkcls.html`

lations they produce (and this did in fact occur). More central to the rescoring strategy adopted for this work is the fact that differences in translation quality arising from the individual characteristics of the decoders can be exploited, as described in section 5.2.

4.2.1 Greedy Decoder

The first decoder we tried (referred to here as GREEDY) was the greedy decoder described in Germann et al. (2001). This approach starts with an initial guess E_0 which it iteratively tries to enhance (i.e., transform to a better scoring string) by applying a series of operators, mainly insertions, deletions or replacements in source words. After each modification is made the resulting string is scored, and the best result at iteration i becomes the next source E_{i+1}. Iteration stops when no improvement is made, i.e. when $E_{i+1} = E_i$. Since we are working with n-best lists, we keep the N best generated hypotheses (including intermediate results).

The algorithm was implemented almost exactly as described, with only a slightly different way of choosing the fertility-0 list (the list of words that can be 'spontaneously' inserted into the source). In addition to using fertility 0 probability, word frequency was also used as a criterion to choose the words in the list: these were selected exclusively from the 10% most common words in English.

Although the ISI REWRITE DECODER[6] was available, using it would have implied training a new language model with the CMU-Cambridge Statistical Language Modeling Toolkit (Clarkson and Rosenfeld, 1997), rather than taking advantage of an existing model: since the algorithm is simple, it was re-implemented.

4.2.2 Stack-based Decoder

The second decoder, referred to as FST, used a stack-based approach (Germann et al., 2001) in which a translation is seen as a path in a probabilistic FST, with states being 'translation contexts' and transitions being simple alignments with their scores.

The translation context associated with a state contains all information needed to obtain the list of alignments that are possible from that state, that is, the alignments that can be added to the path ending at that state. It consists mainly of the two last target words of the path (needed by the trigram language model), the quantities c_{ρ_i} and $class(e_{\rho_i})$ of Germann et al. (2001) (needed for the distortion component of Model 4), and the set of source words already covered.

The initial state is the empty alignment, using none of the source or target words. Then, from any given state there are transitions for every possible single alignment from that state, each transition leading to a new state translation context. We reach a final state when the alignment is complete. The score of the alignment is the sum of the transition weights.

The FST is traversed in a best-first manner. Unfortunately, the search problem is NP-complete, so the space must be reduced. For that, we used a n-dimensional beam table for comparing and pruning hypotheses. The n dimensions of the table divide the paths into subsets of paths (each table cell corresponding to a distinct subset) expected to be directly comparable. While expanding a current path (adding all possible transitions to that path) the out paths formed are directly compared with their subset's current best, and dropped if their score falls outside the beam. An alternative to using a beam is to set a maximum number of paths.

In our case the table had three dimensions, the source path length (number of source words covered by the path), the target path length (number of target words used), and the sum of the logarithm of the frequency of the source words already used (a crude approximation to the joint probability of those words). This last heuristic was introduced as a normalization in order to moderate the strong influence that frequent words in the training corpus tend to have at translation time. This means that we only compare paths using the same number of source and target words, and whose source words have similar a priori probabilities.

4.2.3 Inverted Alignment Decoder

In an attempt to gauge the influence of both decoders and models on the overall translation quality, we also implemented a version of the inverted alignment search algorithm described by Nießen et al. (1998) for IBM 2 models. The basic idea of this method is to expand hypotheses along the positions of the target string while progressively covering the source. The algorithm allows any target word to be aligned to none, one or several consecutive source words (up to a maximum that was set to 3 in the reported experiments); thus, this search accounts for the notion of fertility which is not explicitly captured by IBM 2 models.

A hypothesis is fully determined by four parameters: the source and target positions, the source coverage in words and the target word found at the target position. Therefore, the search space can be represented as a 4-dimensional table, each item in the table containing backtracking information and the hypothesis score. We extended the algorithm in a straightforward manner to account for a trigram language model instead of the bigram language model originally proposed. We experimented with several pruning strategies and report (in table 2) results from two configurations illustrating the usual compromise between decoding speed and quality: IBM2-FAST applies stringent pruning filters and trans-

6 http://www.isi.edu/licensed-sw/rewrite-decoder.

lates a 20-word sentence within a few seconds; IBM2-SLOW performs almost no filtering and, with luck, translates a 20-word sentence in approximately half an hour on our standard Pentium-4 computer.

4.3 Costs of Diversity

While there are advantages in using several decoders (Germann et al., 2001), there are also obvious drawbacks where time is concerned; these are summarized in table 1. There is an element of arbitrariness in these estimates, since other tasks were being carried out concurrently. Nevertheless, we believe it is worth trying to distinguish coding time from tuning time. The former is the time required to produce a decoder that runs on several benchmarks with no obvious bugs either in the code or in the output produced. The latter is the time required to optimize the code and to devise a general filtering strategy that does not exclude too many good hypotheses. In practice, of course, the two are interrelated, optimization of the code sometimes coming at the price of revisions to data structures, and so on.

A working but unfinished decoder can be had in two to three weeks of fairly pleasant work. A final, stable, version tends to take as much time again, giving a total of something over one month. Note that in order to accommodate a different language pair, it would probably be necessary to adjust certain tuning parameters; part of the work in the second phase of decoder development would therefore have to be repeated.

As one might expect, development of the greedy decoder was the most straightforward, and creating a decoder for an IBM Model 2 is slightly faster than doing so for the more complex Model 4. In total, preparation of the decoders required over three man-months of effort.

decoder	coding	tuning
GREEDY	2	1
FST	3	3
IBM2	2	3
total	7	7

Table 1: Approximate number of man weeks of development (coding + tuning) of the different decoders used in this study.

5 Comparing Decoders

In order to compare the decoders, we ran 4 translation sessions; each of these consisted in translating 100 sentences (of at most 20 words, and naturally not seen at training time) randomly selected from the texts provided for the NIST exercise.

Table 2 shows the results in terms of word error rate (WER) and NIST scores as returned by the `mteval` program.[7] Two values returned by this package are actually reported: the NIST score of the candidate, and its normalization by the NIST score obtained by the reference itself (NIST%). These three measures are reported for both the best translation (column 2) and the 100 best translations (column 3) in which case the best value of each score over the 100-best translations is the one reported.

5.1 The Bigger the Better?

Several patterns emerge from these figures. First, we see a great difference in performance over the kind of sentences being translated. Reasons for the notably bad results obtained on *sinorama* and *xinhua* are addressed in section 5.3.

Decoder performance also varies greatly. The greedy decoder seems to be the weakest; this was to be expected since the search space considered by this method is fairly small and since the operations described by Germann et al. (2001) were designed for a different pair of languages (time did not permit us to investigate operations other than those described by the authors). Interestingly, the fast version of the IBM Model 2 decoder yields better results than the greedy decoder relying on Model 4. On the other hand, if time is not a concern better translations can be obtained with the FST method: extending the search space is obviously worth the effort. This is also confirmed by the fact that the IBM2-SLOW method outperforms its IBM2-FAST counterpart. It is also worth noting that, in this setting, Model 4 provides better results than Model 2.

Finally, significantly better results can be expected if we consider even a fairly small number of alternative translations, in this case one hundred. For example, a potential gain of 17% was observed using IBM2-SLOW on the *hansard* corpus.

5.2 Benefits of Diversity

We mentioned in section 4.3 that diversity in approaches involves certain costs, but fortunately it also yields benefits. Since each decoder has its own characteristics, something may be gained by combining their outputs. The following experiment demonstrates this: we merged the 25 best translations produced by each decoder for a given sentence, thus producing a 100-best translation comparable in size to those produced individually by each decoder (although several translation might be identical). The WER measured on this merged session is systematically lower for each corpus than the lowest of the WERs measured for each decoder on each corpus. The absolute improvement in WER (over the minimum observed) is 4·7 for *hansard* (48·30 instead of 53·03), 4·3 for *un*, 1·8 for

[7] `http://www.nist.gov/speech/tests/mt/resources/scoring.htm`

decoder	1-best WER	1-best NIST	1-best NIST%	100-best WER	100-best NIST	100-best NIST%
hansard						
GREEDY	68·93	2·41448	24·20	61·71	3·68806	37·00
IBM2-FAST	65·87	3·22954	32·30	59·22	4·42125	44·20
FST	62·86	4·19043	41·90	55·24	5·10464	51·00
IBM2-SLOW	63·85	3·85769	38·50	53·03	5·28764	52·80
un						
GREEDY	70·35	2·76181	26·10	62·97	3·98415	37·70
IBM2-FAST	69·80	3·19254	30·20	63·04	4·38660	41·50
FST	65·57	4·56739	43·20	57·18	5·80536	54·90
IBM2-SLOW	68·77	4·39036	41·50	58·65	5·77882	54·60
sinorama						
GREEDY	86·89	0·79860	7·80	82·16	1·37465	13·40
IBM2-FAST	87·55	1·09399	10·30	82·45	1·68875	15·80
FST	88·97	1·72001	16·10	85·40	2·35273	22·00
IBM2-SLOW	87·56	1·46096	13·70	81·55	2·44893	23·00
xinhua						
GREEDY	89·64	1·30970	12·70	85·10	2·00496	19·40
IBM2-FAST	91·09	1·08899	10·30	85·90	1·86932	17·70
FST	90·82	1·08510	10·30	87·98	1·56167	14·80
IBM2-SLOW	89·13	1·34132	12·70	83·86	2·29718	21·80

Table 2: Decoder performance on four types of corpus.

sinorama and 0·15 for *xinhua*.

Note that combining the outputs of several translation engines is a natural idea that was used some years ago in the work of Frederking et al. (1994).

5.3 Results Explained

The disappointingly mixed results shown in table 2 raise the question of how the pattern seen there is to be explained. Is the wide divergence between corpora due to some hidden defect in the process by which models were trained or applied, or does it rather reflect an unsuspected variation in some properties of the corpora themselves?

In view of the multiple engine strategy adopted for this work, training of IBM models 2 and 4 being performed by two different packages and decoders being written by two different co-authors, we felt confident in rejecting the hypothesis of major bugs in the process.

In order to investigate other potential reasons for the cross-corpus results, we analysed the corpora further. The poor performance on the *sinorama* and *xinhua* corpora may be related to the fact that, as table 3 illustrates, the train/test data mismatch is more significant on these corpora than on the others: ignoring entries on the diagonal, it can be seen from the perplexities that models trained on the 'good' *un* and *hansard* corpora tend to perform better than those trained on the 'bad' *sinorama* or *xinhua*. Conversely, all models tend to perform better on the 'good' corpora than the 'bad'.

Another possible explanation for the *sinorama* and *xinhua* results is their poor target vocabulary coverage, as illustrated in table 4. The target vocabulary used for decoding is obtained by grouping target words coming from two sources: those directly predicted from source words through the translation table of the model, and the fertility 0 words (similar to the fertility-0 list of GREEDY). A translation can only be formed with these words.

train	test *hansard*	test *sinorama*	test *un*	test *xinhua*
hansard	20·14	451·68	296·10	1005·51
sinorama	769·53	33·85	1058·98	1970·16
un	500·14	860·34	25·01	1007·62
xinhua	1393·69	1269·27	856·07	15·44

Table 3: Language model perplexities obtained by training and testing on various corpora

corpus	*used* OOV	*used* rank	*complete* OOV	*complete* rank
hansard	11·29	1·4	0·29	44·9
sinorama	29·80	1·8	3·81	143·5
un	8·21	1·3	0·53	25·4
xinhua	43·31	2·1	4·06	162·9

Table 4: Target vocabulary coverage.

Table 4 gives the average rank in the translation table of the reference words (expected translation) and the percentage of reference words that are not covered by the

target vocabulary (out-of-vocabulary or OOV). The *used* section refers to the vocabulary that was actually used for decoding. The *complete* section corresponds to a complete vocabulary, which is much too large to be used in practice. We see that both *sinorama* and *xinhua* had a significant portion of OOV words in the *used* target vocabulary. However, the *complete* results show that even when these words are covered by the complete vocabulary, their rank in the translation table is so low that they most probably would not have been picked anyway. These results seem to indicate that it is much more difficult to build a correct target vocabulary for *xinhua* and *sinorama*.

6 Influence of the Training Corpus

After the end of the NIST exercise, we conducted a second experiment in order to analyse the influence of the training corpus. Since practical considerations precluded training a single large model on all the available corpora, we trained several moderately-sized translation models on each of them. Their main characteristics are summarized in table 5, along with their performance on the NIST translation task. We also conducted some experiments involving extracts from the corpora, the details of which are described below.

`take1` is the corpus used for the NIST exercise, containing a preponderance of sentences from the *un* corpus: 934,508, plus 205,368 from the *hansard* corpus, 50,378 from *sinorama*, and 35,234 from *xinhua*.

MIX is another experiment where we gathered a total of 166,529 sentence pairs consisting of those of the *fbis*, the *xinhua* and the *sinorama* corpora, plus the first 50,000 pairs of the *hansard* corpus.

HIGH is an experiment where we used the same training corpus as the MIX experiment, but where the decoder was allowed to explore a larger space.

We used a refined version of the FST decoder, with tunings adjusted so that translating a sentence took 3 minutes on a 1Gb Pentium 4 computer, regardless of its length. Results are shown in table 5; only the NIST score is given, as the evaluation was performed by the automated procedure that NIST maintained after the closing date of the official evaluation in order to allow further development and investigation by participants.

Several interesting observations can be made from table 5. First, it is clear that the choice of the corpus has a clear impact on measured performance: the worst corpus to train on was *hansard* while the best was *fbis*.

Second, a small corpus of within-domain training text is preferable to a larger out-domain one. This can be seen by comparing the size of the *fbis* and *hansard* corpora, the latter producing a much lower score than the former even though it is 6 times bigger. Here of course we assume that the test sentences resembled *fbis* more closely than *hansard*.

corpus	type	$\|lm\|$	$\|tm\|$	NIST
fbis	reports	68,848	36,586	5·0288
sinorama	magazine	103,250	53,877	4·2273
un	reports	720,000	662,360	4·0975
xinhua	news	65,000	39,364	3·7751
hansard	hansards	351,514	335,910	3·5069
MIX	mixed	166,529	166,529	5·6875
HIGH	mixed	166,529	166,529	5·5938
`take1`	mixed	818,937	1,225,488	4·3437

Table 5: NIST score of the translation produced as a function of the training corpus. The $|lm|$ column indicates the number of English sentences from each corpus used for training the language model; $|tm|$ indicates the number of pair of sentences of each corpus.

A third remark can be made concerning the mixed experiments. Clearly, mixing several training corpora (at least for this task, and for in the proportion used) is fruitful. We measured an absolute increase of the NIST score of 0·6 compared to the best score obtained for an individual type of corpus (*fbis*). This being said, we do not know exactly how much could be gained by training a translation model on a much larger corpus.

Finally, from the several experiments carried out on mixed corpora, it is worth pointing out that the better model probabilities achieved by increasing the number of hypotheses considered are not reflected in higher NIST scores; we observed a decrease of 0·1 from MIX to HIGH. To the extent that the NIST metric is accurate, this suggests that the models used are somehow inadequate.

7 Conclusions

In recent years there has been significant progress in numerous fields of natural language processing (NLP), such as automatic speech recognition, machine translation, information retrieval, etc. This progress is in part due to the introduction of statistical methods and machine learning techniques, combined with greater accessibility of data and low-cost computational power. Moreover, a growing number of out-of-the-box, often open-source, technology kits, such as statistical machine learning libraries, have become available. In this context, limits on time and resources represent smaller obstacles to building reliable and useful NLP applications than they have in the past.

However, as our current work suggests, the process of building these applications is not yet as painless and effortless as we could hope. In our attempt to build a SMT system from scratch for an unfamiliar language, using widely available corpora, toolkit for training translation models, and published decoding algorithms, we encoun-

tered numerous unexpected difficulties. The main lesson we learned is that essential practical aspects of the process, time consuming although of little inherent scientific interest, often become the bottleneck.

First, training corpora, although available in large quantities, often require a significant amount of work even when they are ostensibly clean: inconsistent and incorrect annotations and corrupt or incomplete data imposed some frustrating delays. Second, generally known decoding techniques and heuristics cannot be simply applied without extensive experimentations with parameters for accuracy/runtime trade-offs.

References

Brown, P. F., S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer (1993). The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics 19*(2), 263–311.

Clarkson, P. and R. Rosenfeld (1997). Statistical Language Modeling using the CMU-Cambridge Toolkit. In *Eurospeech'97*, Volume 5, pp. 2707–2710.

Frederking, R., S. Nirenburg, D. Farwell, S. Helmreich, E. Hovy, K. Knight, S. Beale, C. Domashnev, D. Attardo, D. Grannes, and R. Brown (1994). Integrating Translations from Multiple Sources within the Pangloss Mark III Machine Translation System. In *Proceedings of the First Conference of the Association for Machine Translation in the Americas (AMTA)*, Columbia, pp. 73–80.

Germann, U., M. Jahr, K. Knight, D. Marcu, and K. Yamada (2001). Fast Decoding and Optimal Decoding for Machine Translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, pp. 228–235.

Nießen, S., S. Vogel, H. Ney, and C. Tillmann (1998). A DP based Search Algorithm for Statistical Machine Translation. In *Proceedings of the 36th Annual Meetings of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, Volume 2, Montreal, pp. 960–967.

Och, F. J. (1999). An Efficient Method for Determining Bilingual Word Classes. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Bergen, Norway, pp. 71–76.

Och, F. J. and H. Ney (2000). Improved Statistical Alignment Models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, pp. 440–447.

Och, F. J. and H. Ney (2002). Discriminative Training and Maximum Entropy Models for Statistical Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, pp. 295–302.

Ortmanns, S., H. Ney, and X. Aubert (1997). A Word Graph Algorithm for Large Vocabulary Continuous Speech Recognition. *Computer Speech and Language 11*(1), 43–72.

Rose, R. C. and G. Riccardi (1999). Automatic Speech Recognition using Acoustic Confidence Conditioned Language Models. In *Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH)*, pp. 303–306.

Soricut, R., K. Knight, and D. Marcu (2002). Using a large monolingual corpus to improve translation accuracy. In *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA-2002)*, Tiburon, CA.

Ueffing, N., F. Och, and H. Ney (2002). Generation of Word Graphs in Statistical Machine Translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 156–163.