Phrase-Based Statistical Machine Translation as a Traveling Salesman Problem

Mikhail Zaslavskiy* Mines ParisTech, Institut Curie, 77305 Fontainebleau, France, mikhail.zaslavskiy@ensmp.fr

Marc Dymetman Xerox Research Centre Europe, 38240 Meylan, France, {marc.dymetman,nicola.cancedda}@xrce.xerox.com

Nicola Cancedda Xerox Research Centre Europe, 38240 Meylan, France, {marc.dymetman,nicola.cancedda}@xrce.xerox.com

Abstract

An efficient decoding algorithm is a crucial element of any statistical machine translation system. Some researchers have noted certain similarities between SMT decoding and the famous Traveling Salesman Problem; in particular (Knight, 1999) has shown that any TSP instance can be mapped to a sub-case of a word-based SMT model, demonstrating NP-hardness of the decoding task. In this paper, we focus on the reverse mapping, showing that any phrase-based SMT decoding problem can be directly reformulated as a TSP. The transformation is very natural, deepens our understanding of the decoding problem, and allows direct use of any of the powerful existing TSP solvers for SMT decoding. We test our approach on three datasets, and compare a TSP-based decoder to the popular beam-search algorithm. In all cases, our method provides competitive or better performance.

1 Introduction

Phrase-based systems (Koehn et al., 2003) are probably the most widespread class of Statistical Machine Translation systems, and arguably one of the most successful. They use aligned sequences of words, called biphrases, as building blocks for translations, and score alternative candidate translations for the same source sentence based on a log-linear model of the conditional probability of target sentences given the source sentence:

$$p(T, a|S) = \frac{1}{Z_S} \exp \sum_k \lambda_k h_k(S, a, T) \qquad (1)$$

where the h_k are features, that is, functions of the source string S, of the target string T, and of the alignment a, where the alignment is a representation of the sequence of biphrases that where used in order to build T from S; The λ_k's are weights and Z_S is a normalization factor that guarantees that p is a proper conditional probability distribution over the pairs (T, A). Some features are *local*, i.e. decompose over biphrases and can be precomputed and stored in advance. These typically include forward and reverse phrase conditional probability features $\log p(\tilde{t}|\tilde{s})$ as well as $\log p(\tilde{s}|\tilde{t})$, where $\tilde{s}$ is the source side of the biphrase and $\tilde{t}$ the target side, and the so-called "phrase penalty" and "word penalty" features, which count the number of phrases and words in the alignment. Other features are *non-local*, i.e. depend on the order in which biphrases appear in the alignment. Typical non-local features include one or more n-gram language models as well as a distortion feature, measuring by how much the order of biphrases in the candidate translation deviates from their order in the source sentence.

Given such a model, where the λ_i's have been tuned on a development set in order to minimize some error rate (see e.g. (Lopez, 2008)), together with a library of biphrases extracted from some large training corpus, a *decoder* implements the actual search among alternative translations:

$$(a^*, T^*) = \arg\max_{(a,T)} P(T, a|S). \qquad (2)$$

The decoding problem (2) is a discrete optimization problem. Usually, it is very hard to find the exact optimum and, therefore, an approximate solution is used. Currently, most decoders are based on some variant of a heuristic left-to-right search, that is, they attempt to build a candidate translation (a, T) incrementally, from left to right, extending the current partial translation at each step with a new biphrase, and computing a score composed of two contributions: one for the known elements of the partial translation so far, and one a heuristic

* This work was conducted during an internship at XRCE.

Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 333–341, Suntec, Singapore, 2-7 August 2009. ©2009 ACL and AFNLP

estimate of the remaining cost for completing the translation. The variant which is mostly used is a form of *beam-search*, where several partial candidates are maintained in parallel, and candidates for which the current score is too low are pruned in favor of candidates that are more promising.

We will see in the next section that some characteristics of beam-search make it a suboptimal choice for phrase-based decoding, and we will propose an alternative. This alternative is based on the observation that phrase-based decoding can be very naturally cast as a Traveling Salesman Problem (TSP), one of the best studied problems in combinatorial optimization. We will show that this formulation is not only a powerful conceptual device for reasoning on decoding, but is also practically convenient: in the same amount of time, off-the-shelf TSP solvers can find higher scoring solutions than the state-of-the art beam-search decoder implemented in *Moses* (Hoang and Koehn, 2008).

2 Related work

Beam-search decoding

In beam-search decoding, candidate translation prefixes are iteratively extended with new phrases. In its most widespread variant, *stack decoding*, prefixes obtained by consuming the same number of source words, no matter which, are grouped together in the same *stack*[1] and compete against one another. *Threshold* and *histogram* pruning are applied: the former consists in dropping all prefixes having a score lesser than the best score by more than some fixed amount (a parameter of the algorithm), the latter consists in dropping all prefixes below a certain rank.

While quite successful in practice, stack decoding presents some shortcomings. A first one is that prefixes obtained by translating different subsets of source words compete against one another. In one early formulation of stack decoding for SMT (Germann et al., 2001), the authors indeed proposed to lazily create one stack for each subset of source words, but acknowledged issues with the potential combinatorial explosion in the number of stacks. This problem is reduced by the use of heuristics for estimating the cost of translating the remaining part of the source sentence. However, this solution is only partially satisfactory. On the one hand, heuristics should be computationally light, much lighter than computing the actual best score itself, while, on the other hand, the heuristics should be tight, as otherwise pruning errors will ensue. There is no clear criterion to guide in this trade-off. Even when good heuristics are available, the decoder will show a bias towards putting at the beginning the translation of a certain portion of the source, either because this portion is less ambiguous (i.e. its translation has larger conditional probability) or because the associated heuristics is less tight, hence more optimistic. Finally, since the translation is built left-to-right the decoder cannot optimize the search by taking advantage of highly unambiguous and informative portions that should be best translated far from the beginning. All these reasons motivate considering alternative decoding strategies.

Word-based SMT and the TSP

As already mentioned, the similarity between SMT decoding and TSP was recognized in (Knight, 1999), who focussed on showing that any TSP can be reformulated as a sub-class of the SMT decoding problem, proving that SMT decoding is NP-hard. Following this work, the existence of many efficient TSP algorithms then inspired certain adaptations of the underlying techniques to SMT decoding for word-based models. Thus, (Germann et al., 2001) adapt a TSP subtour elimination strategy to an IBM-4 model, using generic Integer Programming techniques. The paper comes close to a TSP formulation of decoding with IBM-4 models, but does not pursue this route to the end, stating that *"It is difficult to convert decoding into straight TSP, but a wide range of combinatorial optimization problems (including TSP) can be expressed in the more general framework of linear integer programming"*. By employing generic IP techniques, it is however impossible to rely on the variety of more efficient both exact and approximate approaches which have been designed specifically for the TSP. In (Tillmann and Ney, 2003) and (Tillmann, 2006), the authors modify a certain Dynamic Programming technique used for TSP for use with an IBM-4 word-based model and a phrase-based model respectively. However, to our knowledge, none of these works has proposed a direct reformulation of these SMT models as TSP instances. We believe we are the first to do so, working in our case

[1] While commonly adopted in the speech and SMT communities, this is a bit of a misnomer, since the used data structures are priority queues, not stacks.

with the mainstream phrase-based SMT models, and therefore making it possible to directly apply existing TSP solvers to SMT.

3 The Traveling Salesman Problem and its variants

In this paper the Traveling Salesman Problem appears in four variants:

STSP. The most standard, and most studied, variant is the *Symmetric TSP*: we are given a non-directed graph G on N nodes, where the edges carry real-valued costs. The STSP problem consists in finding a tour of minimal total cost, where a tour (also called Hamiltonian Circuit) is a "circular" sequence of nodes visiting each node of the graph exactly once;

ATSP. The *Asymmetric TSP*, or ATSP, is a variant where the underlying graph G is directed and where, for i and j two nodes of the graph, the edges (i,j) and (j,i) may carry different costs.

SGTSP. The *Symmetric Generalized TSP*, or SGTSP: given a non-oriented graph G of $|G|$ nodes with edges carrying real-valued costs, given a partition of these $|G|$ nodes into m non-empty, disjoint, subsets (called clusters), find a circular sequence of m nodes of minimal total cost, where each cluster is visited exactly once.

AGTSP. The *Asymmetric Generalized TSP*, or AGTSP: similar to the SGTSP, but G is now a directed graph.

The STSP is often simply denoted TSP in the literature, and is known to be NP-hard (Applegate et al., 2007); however there has been enormous interest in developing efficient solvers for it, both exact and approximate.

Most of existing algorithms are designed for *STSP*, but *ATSP*, *SGTSP* and *AGTSP* may be reduced to *STSP*, and therefore solved by *STSP* algorithms.

3.1 Reductions AGTSP→ATSP→STSP

The transformation of the AGTSP into the ATSP, introduced by (Noon and Bean, 1993)), is illustrated in Figure (1). In this diagram, we assume that $Y_1, \ldots, Y_K$ are the nodes of a given cluster, while X and Z are arbitrary nodes belonging to other clusters. In the transformed graph, we introduce edges between the Y_i's in order to form a cycle as shown in the figure, where each edge has a large negative cost $-K$. We leave alone the incoming edge to Y_i from X, but the outgoing edge

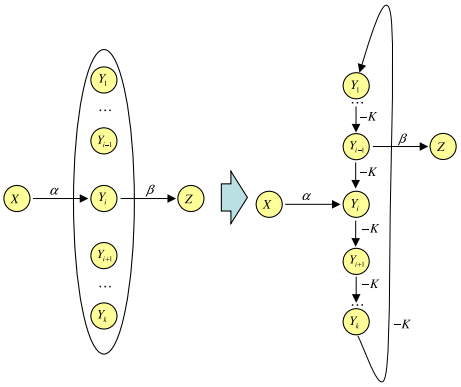


Figure 1: AGTSP→ATSP.

from Y_i to X has its origin changed to Y_{i-1}. A feasible tour in the original AGTSP problem passing through X, Y_i, Z will then be "encoded" as a tour of the transformed graph that first traverses X , then traverses $Y_i, \ldots, Y_K, \ldots, Y_{i-1}$, then traverses Z (this encoding will have the same cost as the original cost, minus $(k-1)K$). Crucially, if K is large enough, then the solver for the transformed ATSP graph will tend to traverse as many K edges as possible, meaning that it will traverse exactly $k-1$ such edges in the cluster, that is, it will produce an encoding of some feasible tour of the AGTSP problem.

As for the transformation ATSP→STSP, several variants are described in the literature, e.g. (Applegate et al., 2007, p. 126); the one we use is from (Wikipedia, 2009) (not illustrated here for lack of space).

3.2 TSP algorithms

TSP is one of the most studied problems in combinatorial optimization, and even a brief review of existing approaches would take too much place. Interested readers may consult (Applegate et al., 2007; Gutin, 2003) for good introductions.

One of the best existing TSP solvers is implemented in the open source *Concorde* package (Applegate et al., 2005). *Concorde* includes the fastest exact algorithm and one of the most efficient implementations of the Lin-Kernighan (LK) heuristic for finding an approximate solution. LK works by generating an initial random feasible solution for the TSP problem, and then repeatedly identifying an ordered subset of k edges in the current tour and an ordered subset of k edges not included in the tour such that when they are swapped the objective function is improved. This is somewhat

reminiscent of the *Greedy decoding* of (Germann et al., 2001), but in LK several transformations can be applied simultaneously, so that the risk of being stuck in a local optimum is reduced (Applegate et al., 2007, chapter 15).

As will be shown in the next section, phrase-based SMT decoding can be directly reformulated as an AGTSP. Here we use *Concorde* through first transforming AGTSP into STSP, but it might also be interesting in the future to use algorithms specifically designed for AGTSP, which could improve efficiency further (see Conclusion).

4 Phrase-based Decoding as TSP

In this section we reformulate the SMT decoding problem as an *AGTSP*. We will illustrate the approach through a simple example: translating the French sentence *"cette traduction automatique est curieuse"* into English. We assume that the relevant biphrases for translating the sentence are as follows:

ID	source	target
h	*cette*	*this*
t	*traduction*	*translation*
ht	*cette traduction*	*this translation*
mt	*traduction automatique*	*machine translation*
a	*automatique*	*automatic*
m	*automatique*	*machine*
i	*est*	*is*
s	*curieuse*	*strange*
c	*curieuse*	*curious*

Under this model, we can produce, among others, the following translations:

$h \cdot mt \cdot i \cdot s$	*this machine translation is strange*
$h \cdot c \cdot t \cdot i \cdot a$	*this curious translation is automatic*
$ht \cdot s \cdot i \cdot a$	*this translation strange is automatic*

where we have indicated on the left the ordered sequence of biphrases that leads to each translation.

We now formulate decoding as an AGTSP, in the following way. The graph nodes are all the possible pairs (w, b), where w is a source word in the source sentence s and b is a biphrase containing this source word. The graph clusters are the subsets of the graph nodes that share a common source word w.

The costs of a transition between nodes M and N of the graph are defined as follows:
(a) If M is of the form (w, b) and N of the form (w', b), in which b is a single biphrase, and w and w' are consecutive words in b, then the transition cost is 0: once we commit to using the first word of b, there is no additional cost for traversing the other source words covered by b.
(b) If $M = (w, b)$, where w is the *rightmost source word* in the biphrase b, and $N = (w', b')$, where $w' \neq w$ is the *leftmost source word* in b', then the transition cost corresponds to the cost of selecting b' just after b; this will correspond to "consuming" the source side of b' after having consumed the source side of b (whatever their relative positions in the source sentence), and to producing the target side of b' directly after the target side of b; the transition cost is then the addition of several contributions (weighted by their respective λ (not shown), as in equation 1):

- The cost associated with the features local to b in the biphrase library;
- The "distortion" cost of consuming the source word w' just after the source word w: $|\text{pos}(w') - \text{pos}(w) - 1|$, where $\text{pos}(w)$ and $\text{pos}(w')$ are the positions of w and w' in the source sentence.
- The language model cost of producing the target words of b' right after the target words of b; with a bigram language model, this cost can be precomputed directly from b and b'. This restriction to bigram models will be removed in Section 4.1.

(c) In all other cases, the transition cost is infinite, or, in other words, there is no edge in the graph between M and N.

A special cluster containing a single node (denoted by $-$$ in the figures), and corresponding to special *beginning-of-sentence* symbols must also be included: the corresponding edges and weights can be worked out easily. Figures 2 and 3 give some illustrations of what we have just described.

4.1 From Bigram to N-gram LM

Successful phrase-based systems typically employ language models of order higher than two. However, our models so far have the following important "Markovian" property: the cost of a path is additive relative to the costs of transitions. For example, in the example of Figure 3, the cost of *this · machine translation · is · strange*, can only take into account the conditional probability of the word *strange* relative to the word *is*, but not relative to the words *translation* and *is*. If we want to extend the power of the model to general n-gram language models, and in particular to the 3-gram

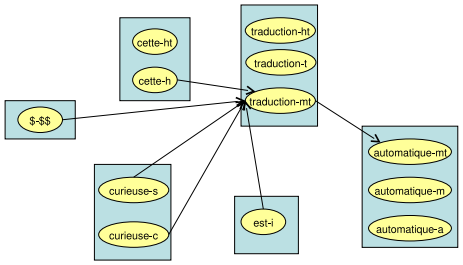


Figure 2: Transition graph for the source sentence *cette traduction automatique est curieuse*. Only edges entering or exiting the node traduction – mt are shown. The only successor to [traduction – mt] is [automatique – mt], and [cette – ht] is not a predecessor of [traduction – mt].

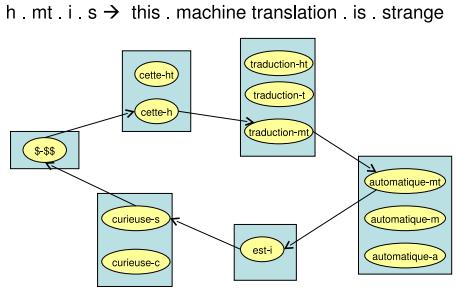


Figure 3: A GTSP tours is illustrated, corresponding to the displayed output.

case (on which we concentrate here, but the techniques can be easily extended to the general case), the following approach can be applied.

Compiling Out for Trigram models

This approach consists in "compiling out" all biphrases with a target side of only one word. We replace each biphrase b with single-word target side by "extended" biphrases $b_1, \ldots, b_r$, which are "concatenations" of b and some other biphrase b' in the library.[2] To give an example, consider that we: (1) remove from the biphrase library the biphrase i, which has a single word target, and (2) add to the library the extended biphrases mti, ti, $si, \ldots$, that is, all the extended biphrases consisting of the concatenation of a biphrase in the library with i, then it is clear that these extended biphrases will provide enough context to compute a trigram probability for the target word produced immediately next (in the examples, for the words *strange*, *automatic* and *automatic* respectively). If we do that exhaustively for all biphrases (relevant for the source sentence at hand) that, like i, have a single-word target, we will obtain a representation that allows a trigram language model to be computed at each point.

[2] In the figures, such "concatenations" are denoted by $[b' \cdot b]$; they are interpreted as encapsulations of first consuming the source side of b', *whether or not this source side precedes the source side of* b *in the source sentence*, producing the target side of b', consuming the source side of b, and producing the target side of b immediately after that of b'.

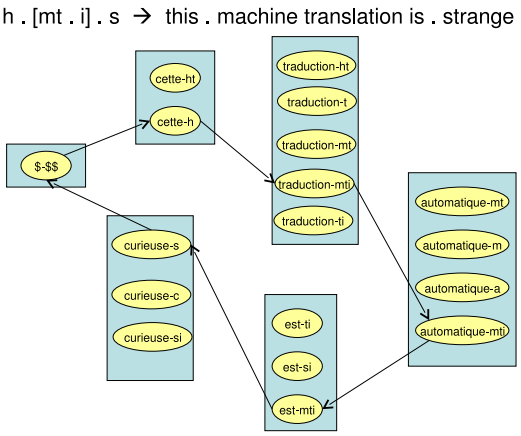


Figure 4: Compiling-out of biphrase i: (est,is).

The situation becomes clearer by looking at Figure 4, where we have only eliminated the biphrase i, and only shown some of the extended biphrases that now encapsulate i, and where we show one valid circuit. Note that we are now able to associate with the edge connecting the two nodes (est, mti) and $(\text{curieuse}, s)$ a trigram cost because mti provides a large enough target context.

While this exhaustive "compiling out" method works in principle, it has a serious defect: if for the sentence to be translated, there are m relevant biphrases, among which k have single-word targets, then we will create on the order of km extended biphrases, which may represent a significant overhead for the TSP solver, as soon as k is large relative to m, which is typically the case. The problem becomes even worse if we extend the compiling-out method to n-gram language models with $n > 3$. In the Future Work section below, we describe a powerful approach for circumventing this problem, but with which we have not experimented yet.

5 Experiments

5.1 Monolingual word re-ordering

In the first series of experiments we consider the artificial task of reconstructing the original word order of a given English sentence. First, we randomly permute words in the sentence, and then we try to reconstruct the original order by max-

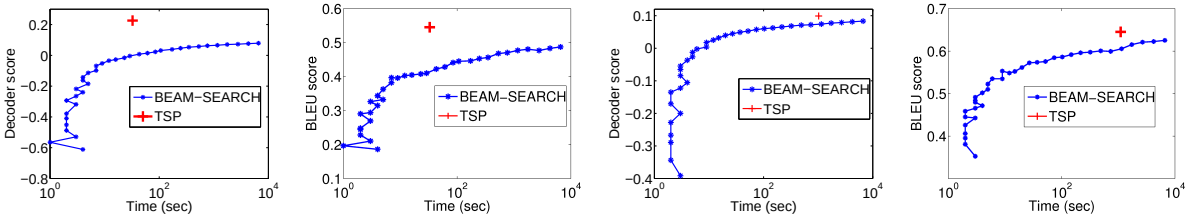


Figure 5: (a), (b): LM and BLEU scores as functions of time for a bigram LM; (c), (d): the same for a trigram LM. The x axis corresponds to the cumulative time for processing the test set; for (a) and (c), the y axis corresponds to the mean difference (over all sentences) between the lm score of the output and the lm score of the reference normalized by the sentence length N: (LM(ref)-LM(true))/N. The solid line with star marks corresponds to using beam-search with different pruning thresholds, which result in different processing times and performances. The cross corresponds to using the exact-TSP decoder (in this case the time to the optimal solution is not under the user's control).

imizing the LM score over all possible permutations. The reconstruction procedure may be seen as a translation problem from "Bad English" to "Good English". Usually the LM score is used as one component of a more complex decoder score which also includes biphrase and distortion scores. But in this particular "translation task" from bad to good English, we consider that all "biphrases" are of the form $e - e$, where e is an English word, and we do not take into account any distortion: we only consider the quality of the permutation as it is measured by the LM component. Since for each "source word" e, there is exactly one possible "biphrase" $e - e$ each cluster of the Generalized TSP representation of the decoding problem contains exactly one node; in other terms, the Generalized TSP in this situation is simply a standard TSP. Since the decoding phase is then equivalent to a word reordering, the LM score may be used to compare the performance of different decoding algorithms. Here, we compare three different algorithms: classical beam-search (*Moses*); a decoder based on an exact TSP solver (*Concorde*); a decoder based on an approximate TSP solver (Lin-Kernighan as implemented in the Concorde solver) [3]. In the Beam-search and the LK-based TSP solver we can control the trade-off between approximation quality and running time. To measure re-ordering quality, we use two scores. The first one is just the "internal" LM score; since all three algorithms attempt to maximize this score, a natural evaluation procedure is to plot its value versus the elapsed time. The second score is BLEU (Papineni et al., 2001), computed between the reconstructed and the original sentences, which allows us to check how well the quality of reconstruction correlates with the internal score. The training dataset for learning the LM consists of 50000 sentences from NewsCommentary corpus (Callison-Burch et al., 2008), the test dataset for word reordering consists of 170 sentences, the average length of test sentences is equal to 17 words.

Bigram based reordering. First we consider a bigram Language Model and the algorithms try to find the re-ordering that maximizes the LM score. The TSP solver used here is exact, that is, it actually finds the optimal tour. Figures 5(a,b) present the performance of the TSP and Beam-search based methods.

Trigram based reordering. Then we consider a trigram based Language Model and the algorithms again try to maximize the LM score. The trigram model used is a variant of the exhaustive compiling-out procedure described in Section 4.1. Again, we use an exact TSP solver.

Looking at Figure 5a, we see a somewhat surprising fact: the cross and some star points have positive y coordinates! This means that, when using a bigram language model, it is often possible to reorder the words of a randomly permuted reference sentence in such a way that the LM score of the reordered sentence is larger than the LM of the reference. A second notable point is that the increase in the LM-score of the beam-search with time is steady but very slow, and never reaches the level of performance obtained with the exact-TSP procedure, even when increasing the time by sev-

[3] Both TSP decoders may be used with/or without a *distortion limit*; in our experiments we do not use this parameter.

eral orders of magnitude. Also to be noted is that the solution obtained by the exact-TSP is provably the optimum, which is almost never the case of the beam-search procedure. In Figure 5b, we report the BLEU score of the reordered sentences in the test set relative to the original reference sentences. Here we see that the exact-TSP outputs are closer to the references in terms of BLEU than the beam-search solutions. Although the TSP output does not recover the reference sentences (it produces sentences with a slightly higher LM score than the references), it does reconstruct the references better than the beam-search. The experiments with trigram language models (Figures 5(c,d)) show similar trends to those with bigrams.

5.2 Translation experiments with a bigram language model

In this section we consider two real translation tasks, namely, translation from English to French, trained on Europarl (Koehn et al., 2003) and translation from German to Spanish training on the NewsCommentary corpus. For Europarl, the training set includes 2.81 million sentences, and the test set 500. For NewsCommentary the training set is smaller: around 63k sentences, with a test set of 500 sentences. Figure 6 presents Decoder and Bleu scores as functions of time for the two corpuses.

Since in the real translation task, the size of the TSP graph is much larger than in the artificial reordering task (in our experiments the median size of the TSP graph was around 400 nodes, sometimes growing up to 2000 nodes), directly applying the exact TSP solver would take too long; instead we use the approximate LK algorithm and compare it to Beam-Search. The efficiency of the LK algorithm can be significantly increased by using a good initialization. To compare the quality of the LK and Beam-Search methods we take a rough initial solution produced by the Beam-Search algorithm using a small value for the stack size and then use it as initial point, both for the LK algorithm and for further Beam-Search optimization (where as before we vary the Beam-Search thresholds in order to trade quality for time).

In the case of the Europarl corpus, we observe that LK outperforms Beam-Search in terms of the Decoder score as well as in terms of the BLEU score. Note that the difference between the two algorithms increases steeply at the beginning, which means that we can significantly increase the quality of the Beam-Search solution by using the LK algorithm at a very small price. In addition, it is important to note that the BLEU scores obtained in these experiments correspond to feature weights, in the log-linear model (1), that have been optimized for the Moses decoder, but not for the TSP decoder: optimizing these parameters relatively to the TSP decoder could improve its BLEU scores still further.

On the News corpus, again, LK outperforms Beam-Search in terms of the Decoder score. The situation with the BLEU score is more confuse. Both algorithms do not show any clear score improvement with increasing running time which suggests that the decoder's objective function is not very well correlated with the BLEU score on this corpus.

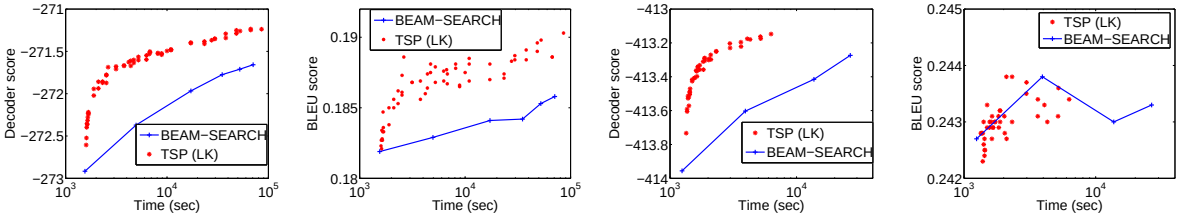
6 Future Work

In section 4.1, we described a general "compiling out" method for extending our TSP representation to handling trigram and N-gram language models, but we noted that the method may lead to combinatorial explosion of the TSP graph. While this problem was manageable for the artificial monolingual word re-ordering (which had only one possible translation for each source word), it becomes unwieldy for the real translation experiments, which is why in this paper we only considered bigram LMs for these experiments. However, we know how to handle this problem in principle, and we now describe a method that we plan to experiment with in the future.

To avoid the large number of artificial biphrases as in 4.1, we perform an *adaptive selection*. Let us suppose that (w, b) is a SMT decoding graph node, where b is a biphrase containing only one word on the target side. On the first step, when we evaluate the traveling cost from (w, b) to (w', b'), we take the language model component equal to

$$\min_{b'' \neq b', b} -\log p(b'.v | b.e, b''.e),$$

where $b'.v$ represents the first word of the b' target side, $b.e$ is the only word of the b target side, and $b''.e$ is the last word of the b'' target size. This procedure underestimates the total cost of tour passing through biphrases that have a single-word target. Therefore if the optimal tour passes only through biphrases with more than one



(a) (b) (c) (d)

Figure 6: (a), (b): Europarl corpus, translation from English to French; (c),(d): NewsCommentary corpus, translation from German to Spanish. Average value of the decoder and the BLEU scores (over 500 test sentences) as a function of time. The trade-off quality/time in the case of LK is controlled by the number of iterations, and each point corresponds to a particular number of iterations, in our experiments LK was run with a number of iterations varying between 2k and 170k. The same trade-off in the case of Beam-Search is controlled by varying the beam thresholds.

word on their target side, then we are sure that this tour is also optimal in terms of the tri-gram language model. Otherwise, if the optimal tour passes through (w, b), where b is a biphrase having a single-word target, we add only the extended biphrases related to b as we described in section 4.1, and then we recompute the optimal tour. Iterating this procedure provably converges to an optimal solution.

This powerful method, which was proposed in (Kam and Kopec, 1996; Popat et al., 2001) in the context of a finite-state model (but not of TSP), can be easily extended to N-gram situations, and typically converges in a small number of iterations.

7 Conclusion

The main contribution of this paper has been to propose a transformation for an arbitrary phrase-based SMT decoding instance into a TSP instance. While certain similarities of SMT decoding and TSP were already pointed out in (Knight, 1999), where it was shown that any Traveling Salesman Problem may be reformulated as an instance of a (simplistic) SMT decoding task, and while certain techniques used for TSP were then adapted to word-based SMT decoding (Germann et al., 2001; Tillmann and Ney, 2003; Tillmann, 2006), we are not aware of any previous work that shows that SMT decoding can be directly reformulated as a TSP. Beside the general interest of this transformation for understanding decoding, it also opens the door to direct application of the variety of existing TSP algorithms to SMT. Our experiments on synthetic and real data show that fast TSP algorithms can handle selection and reordering in SMT comparably or better than the state-of-the-art beam-search strategy, converging on solutions with higher objective function in a shorter time.

The proposed method proceeds by first constructing an AGTSP instance from the decoding problem, and then converting this instance first into ATSP and finally into STSP. At this point, a direct application of the well known STSP solver *Concorde* (with Lin-Kernighan heuristic) already gives good results. We believe however that there might exist even more efficient alternatives. Instead of converting the AGTSP instance into a STSP instance, it might prove better to use directly algorithms expressly designed for ATSP or AGTSP. For instance, some of the algorithms tested in the context of the *DIMACS* implementation challenge for ATSP (Johnson et al., 2002) might well prove superior. There is also active research around AGTSP algorithms. Recently new effective methods based on a "memetic" strategy (Buriol et al., 2004; Gutin et al., 2008) have been put forward. These methods combined with our proposed formulation provide ready-to-use SMT decoders, which it will be interesting to compare.

Acknowledgments

Thanks to Vassilina Nikoulina for her advice about running Moses on the test datasets.

References

David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. 2005. Concorde tsp solver. `http://www.tsp.gatech.edu/concorde.html`.

David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. 2007. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, January.

Luciana Buriol, Paulo M. França, and Pablo Moscato. 2004. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10(5):483–506.

Chris Callison-Burch, Philipp Koehn, Christof Monz, Josh Schroeder, and Cameron Shaw Fordyce, editors. 2008. *Proceedings of the Third Workshop on SMT*. ACL, Columbus, Ohio, June.

Ulrich Germann, Michael Jahr, Kevin Knight, and Daniel Marcu. 2001. Fast decoding and optimal decoding for machine translation. In *In Proceedings of ACL 39*, pages 228–235.

Gregory Gutin, Daniel Karapetyan, and Krasnogor Natalio. 2008. Memetic algorithm for the generalized asymmetric traveling salesman problem. In *NICSO 2007*, pages 199–210. Springer Berlin.

G. Gutin. 2003. Travelling salesman and related problems. In *Handbook of Graph Theory*.

Hieu Hoang and Philipp Koehn. 2008. Design of the Moses decoder for statistical machine translation. In *ACL 2008 Software workshop*, pages 58–65, Columbus, Ohio, June. ACL.

D.S. Johnson, G. Gutin, L.A. McGeoch, A. Yeo, W. Zhang, and A. Zverovich. 2002. Experimental analysis of heuristics for the atsp. In *The Travelling Salesman Problem and Its Variations*, pages 445–487.

Anthony C. Kam and Gary E. Kopec. 1996. Document image decoding by heuristic search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:945–950.

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

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *NAACL 2003*, pages 48–54, Morristown, NJ, USA. Association for Computational Linguistics.

Adam Lopez. 2008. Statistical machine translation. *ACM Comput. Surv.*, 40(3):1–49.

C. Noon and J.C. Bean. 1993. An efficient transformation of the generalized traveling salesman problem. *INFOR*, pages 39–44.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei J. Zhu. 2001. BLEU: a Method for Automatic Evaluation of Machine Translation. *IBM Research Report*, RC22176.

Kris Popat, Daniel H. Greene, Justin K. Romberg, and Dan S. Bloomberg. 2001. Adding linguistic constraints to document image decoding: Comparing the iterated complete path and stack algorithms.

Christoph Tillmann and Hermann Ney. 2003. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Comput. Linguist.*, 29(1):97–133.

Christoph Tillmann. 2006. Efficient Dynamic Programming Search Algorithms For Phrase-Based SMT. In *Workshop On Computationally Hard Problems And Joint Inference In Speech And Language Processing*.

Wikipedia. 2009. Travelling Salesman Problem — Wikipedia, The Free Encyclopedia. [Online; accessed 5-May-2009].