Fast and optimal decoding for machine translation

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Artificial Intelligence 154 (2004) 127–143
Artificial Intelligence
www.elsevier.com/locate/artint

Fast and optimal decoding for machine translation ☆

Ulrich Germann [a,*], Michael Jahr [b], Kevin Knight [a], Daniel Marcu [a], Kenji Yamada [a]

[a] *Information Sciences Institute and Department of Computer Science, University of Southern California, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA*
[b] *Department of Computer Science, Stanford University, Stanford, CA 94305, USA*

Received 25 June 2002; received in revised form 19 June 2003

Abstract

A good decoding algorithm is critical to the success of any statistical machine translation system. The decoder's job is to find the translation that is most likely according to a set of previously learned parameters (and a formula for combining them). Since the space of possible translations is extremely large, typical decoding algorithms are only able to examine a portion of it, thus risking to miss good solutions. Unfortunately, examining more of the space leads to unacceptably slow decodings.

In this paper, we compare the speed and output quality of a traditional stack-based decoding algorithm with two new decoders: a fast but non-optimal greedy decoder and a slow but optimal decoder that treats decoding as an integer-programming optimization problem.

Keywords: Statistical machine translation; Machine translation; Decoding; SMT; MT

1. Introduction

Statistical machine translation (SMT) in the tradition of Brown et al. (e.g. [1,3]), which is often referred to as the noisy channel approach to machine translation, restates the

☆ This is an extended version of our paper, "Fast and Optimal Decoding for Machine Translation", which received a Best Paper Award at the 39th Annual Meeting of the Association for Computational Linguistics (ACL-2001).

* Corresponding author.
E-mail addresses: germann@isi.edu (U. Germann), jahr@cs.stanford.edu (M. Jahr), knight@isi.edu (K. Knight), marcu@isi.edu (D. Marcu), kyamada@isi.edu (K. Yamada).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2003.06.001

problem of finding the optimal (say) English translation $\hat{\mathbf{e}}$ of a French sentence **f**, or $\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} P(\mathbf{e} \mid \mathbf{f})$, as finding

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} P(\mathbf{f} \mid \mathbf{e}) \cdot P(\mathbf{e}).^{1}$$

SMT systems within this framework typically consist of three components: (1) a language model (LM) that assigns a probability $P(\mathbf{e})$ to any given English string **e**; (2) a translation model (TM) that assigns a probability $P(\mathbf{f} \mid \mathbf{e})$ to any given pair of English and French strings **e** and **f**; and (3) a decoding algorithm (**decoder**) to perform the search.

If the source and target languages are constrained to have the same word order (by choice or through suitable preprocessing), then a linear Viterbi algorithm can be applied [12]. If word re-ordering is limited to rotations around nodes in a binary tree, it can be carried out by a high-polynomial algorithm [14]. For arbitrary word-reordering, the decoding problem is NP-hard [7].

It is a sensible, albeit still computationally intensive strategy to restrict the search to a large subset of likely decodings and choose just among them [2,13]. Obviously, it is possible to miss good translations this way. Whenever the decoder returns a sub-optimal solution, i.e., a string $\mathbf{e}'$ for which there exists some other string $\hat{\mathbf{e}}$ such that $P(\mathbf{f} \mid \hat{\mathbf{e}}) \cdot P(\hat{\mathbf{e}}) > P(\mathbf{f} \mid \mathbf{e}') \cdot P(\mathbf{e}')$, we call this a **search error**. As Wang and Waibel [13] remark, it is hard to determine search errors—the only way to show that a decoding is sub-optimal is to actually produce a higher-scoring one.

Thus, while decoding is a clear-cut optimization task in which every problem instance has a right answer, it is hard to come up with good answers quickly. This paper reports on measurements of speed, search errors, and translation quality in the context of a traditional stack decoder [2,6] and two new decoders. The first is a fast greedy decoder, and the second is a slow optimal decoder based on generic mathematical programming techniques.

We also consider how the three decoders can be combined, and how the output of one helps in the design of the others. Since all our experiments are cast in a framework that uses the IBM translation Model 4 [3], we begin with a description of this particular TM.

2. IBM Model 4

Among the models presented in [3], Model 4 is the most sophisticated and most suitable for decoding.[2] Like all the other IBM models, it revolves around the notion of **word alignments**. Given an English string **e** and a French string **f**, a word alignment is a many-to-one function that maps each word in **f** onto exactly one word in **e**, or onto the **NULL word**. The NULL word is a mechanism to account for French words that have no direct counterpart in the English string, such as the word "*-là*" in Fig. 1. The **fertility** of an

[1] According to Bayes' Law, $P(\mathbf{e} \mid \mathbf{f}) \cdot P(\mathbf{f}) = P(\mathbf{f} \mid \mathbf{e}) \cdot P(\mathbf{e})$. Since $P(\mathbf{f})$ is constant for any given **f**, $\arg\max_{\mathbf{e}}(P(\mathbf{f} \mid \mathbf{e}) \cdot P(\mathbf{e}))/P(\mathbf{f}) = \arg\max_{\mathbf{e}} P(\mathbf{f} \mid \mathbf{e}) \cdot P(\mathbf{e})$.

[2] Model 5 is aimed at removing a technical deficiency from Model 4 (see [3] and below), and possibly achieving better alignments in bilingual training data. However, it also removes useful conditioning variables. We therefore prefer Model 4 for decoding.

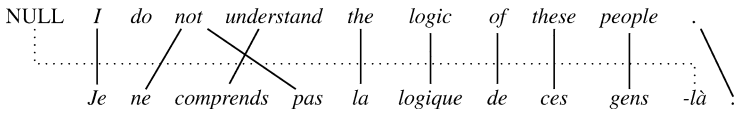

Fig. 1. A sample word alignment.

English word is the number of French words mapped onto it; English words with a fertility of zero are called **infertile**. In Fig. 1, the word "*not*" has a fertility of 2, and "*do*" is infertile.

Model 4 assumes the following stochastic process that creates a French string and an alignment function from a given English string.

First, every English word is assigned a fertility. These assignments are made stochastically according to the **fertility table** $n(\phi_i \mid e_i)$. We delete from the string any word with fertility zero, we duplicate any word with fertility two, and so forth. The NULL word originally has the fertility $\phi_0 = 0$. For each English word in the resulting string, we increment ϕ_0 by one with the probability p_1, which is typically about .02 for the language pair English/French.

Next, we perform a word-for-word replacement of the English words (and ϕ_0 'copies' of the NULL word) by French words, according to the **translation table** $t(f_{i,k} \mid e_i)$. The indices i and k indicate that the respective French word is the replacement of the kth copy of the original English word e_i.

Finally, we permute the French words by stochastically assigning a string position $\pi_{i,k}$ to each French word. With respect to these permutations, Model 4 distinguishes between **heads**, **non-heads**, and **NULL-generated** words. The **head** $f_{i,1}$ of an English word e_i, is the leftmost of the ϕ_i French words generated by e_i. All subsequent words $f_{i,k>1}$ aligned with e_i (if any) are **non-heads**. We can picture the process as follows.

Heads. For each English word e_i, the head word $f_{i,1}$ is assigned a French string position $\pi_{i,1}$. The probability of this assignment is determined by the **head distortion table** $d_1(\pi_{i,1} - \lceil \overline{\pi_{\rho_i}} \rceil \mid \mathit{class}(e_{\rho_i}), \mathit{class}(f_{i,1}))$ where *class* is a function that assigns automatically determined word classes to French and English vocabulary items, ρ_i the index of the first fertile English word to the left of e_i[3] and $\lceil \overline{\pi_{\rho_i}} \rceil$ the ceiling of the average of the positions of all French words aligned with e_{ρ_i}.

Non-heads. For English words with $\phi_i > 1$, the remaining words are all assigned a position $\pi_{i,k} > \pi_{i,k-1}$ with $1 < k \leqslant \phi_i$ according to the **non-head distortion table** $d_{>1}(\pi_{i,k} - \pi_{i,k-1} \mid \mathit{class}(f_{i,k}))$.

The use of relative offsets rather than absolute string positions encourages adjacent English words to translate into adjacent French words. In a monotone alignment (no word re-ordering), all probabilities will be of the form $d(+1 \mid \ldots)$, which is typically high for French and English.

[3] Brown et al. [3] do not provide a precise definition of the probability of the position of the head of e_1.

NULL-generated. After heads and non-heads are placed, NULL-generated words are permuted into the remaining vacant slots randomly. The aggregate distortion probability of these NULL-generated words is 1: There are $\phi_0!$ different placement schemes for ϕ_0 NULL-generated words,[4] each of which is chosen with probability $1/\phi_0!$. However, there are also $\phi_0!$ ways of generating these words (unlike 'regular' English words, NULL generates French words in no particular order). Thus, the number of ways to generate these words and the number of schemes to distribute them into the vacant slots cancel each other out.

The reader may have noticed that the distortion procedure just described neither a priori prevents French words from being piled on top of each other nor guarantees that the French string does not have unfilled positions. This technical deficiency of Model 4 has been recognized already by Brown et al. [3] and eliminated in Model 5. With respect to decoding, this deficiency does not cause any problems, as we will simply not consider any unreasonable alignment hypothesis of this sort, regardless of its probability.

The result of the stochastic generation process just described is a French string $\mathbf{f}$ and a word alignment a of $\mathbf{f}$ with $\mathbf{e}$. The probability $P(a, \mathbf{f} \mid \mathbf{e})$ is the product of all individual decisions in this process.

$$
\begin{aligned}
P(a, \mathbf{f} \mid \mathbf{e}) = {} & \prod_{i=1}^{l} n(\phi_i \mid e_i) \quad \text{fertilities of English words} \\
& \times \binom{m-\phi_0}{\phi_0} p_1^{\phi_0} (1-p_1)^{m-2\phi_0} \quad \text{NULL word fertility} \\
& \times \prod_{j=1}^{m} t(f_j \mid e_{a(j)}) \quad \text{word translations} \\
& \times \prod_{i=1,\phi_i>0}^{l} d_1\big(\pi_{i,1} - \lceil \overline{\pi_{\rho_i}} \rceil \mid \mathit{class}(e_{\rho_i}), \mathit{class}(f_{i,1})\big) \quad \text{head distortions} \\
& \times \prod_{i=1,\phi_i>1}^{l} \prod_{k=2}^{\phi_i} d_{>1}(\pi_{i,k} - \pi_{i,k-1} \mid \mathit{class}(f_{i,k})) \quad \text{non-head distortions.}
\end{aligned}
$$

In this formula, $\mathbf{e} = e_1, \ldots, e_l$ is the English string, e_0 the NULL word, ϕ_i the fertility of e_i, $\mathbf{f} = f_1, \ldots, f_m$ the French string, $a : \{1, \ldots, m\} \mapsto \{0, 1, \ldots, l\}$ the alignment function that maps each French index position word onto the index of the corresponding English word (or the NULL word), $f_{i,k}$ the kth French word produced by e_i, $\pi_{i,k}$ the position of $f_{i,k}$ in $\mathbf{f}$ (with $a(\pi_{i,k}) = i$), ρ_i the index of the first fertile English word to the left of e_i, and $\overline{\pi_{\rho_i}}$ the average position of all French words aligned with e_{ρ_i}.

[4] Assuming a "sane" placement of non-NULL-generated words, leaving exactly ϕ_0 gaps.

3. Definition of the search problem

The probability $P(\mathbf{f} \mid \mathbf{e})$ is the sum of the probabilities of all possible ways of aligning **f** with **e**:

$$P(\mathbf{f} \mid \mathbf{e}) = \sum_{a} P(a, \mathbf{f} \mid \mathbf{e}).$$

Calculating $\sum_a P(a, \mathbf{f} \mid \mathbf{e})$ is prohibitively expensive: there are m^{l+1} different alignments of m French with l English words. Therefore, it is common practice to approximate the solution by searching for the pair $\langle \hat{\mathbf{e}}, \hat{a} \rangle$ that maximizes the term $P(a, \mathbf{f} \mid \mathbf{e}) \cdot P(\mathbf{e})$.

Definition of the search problem. Given an input string **f**, find the string $\hat{\mathbf{e}}$ and the alignment $\hat{a}$ so that

$$\langle \hat{\mathbf{e}}, \hat{a} \rangle = \arg\max_{\mathbf{e},a} P(a, \mathbf{f} \mid \mathbf{e}) \cdot P(\mathbf{e}).$$

In Model 4 decoding, $P(a, \mathbf{f} \mid \mathbf{e})$ is calculated with the formula given above. $P(\mathbf{e})$ is typically estimated using a smoothed n-gram model of English.

4. Stack-based decoding

4.1. The basic algorithm

Stack decoding is a best-first search algorithm first introduced into the domain of speech recognition (SR) by Jelinek [6]. It is very similar to the **A*** algorithm originally presented by Hart et al. [5]. A stack decoder conducts an ordered search through the search space by building solutions incrementally and storing partial solutions (**hypotheses**) in a priority queue. Though technically misleading,[5] this priority queue is usually called the **stack**. In this article, we follow this terminological convention. Under ideal circumstances (unlimited stack size and exhaustive search time), a stack decoder is guaranteed to find an optimal solution [9]. Our hope is to do almost as well under real-world constraints of limited space and time.

The basic stack decoding algorithm works as follows.

(1) Initialize the stack with an **empty hypothesis**.
(2) Pop h, the most promising hypothesis, off the stack.
(3) If h is **complete** (defined below), output h and terminate.
(4) Extend h in each possible manner of incorporating the next input word, and insert the resulting hypotheses into the stack.
(5) Return to step (2).

[5] The search algorithm always expands the most promising (highest-scoring to some evaluation/prediction function) hypothesis, not the most recent one.

In the context of MT, a hypothesis is a partial word alignment: an English string and an alignment function that maps a subset of the French input words onto the words in this string (or onto the NULL word). A hypothesis is called **complete** if it accounts for all French input words. (Note that we are now in the business of decoding, so that the translation direction changes: The input is French and the output English.)

4.2. Stack decoding: machine translation versus speech recognition

There are two important differences between stack decoding for speech recognition and stack decoding for machine translation. The first is that in SR, the transcription always follows the input order. In other words, there is always a strict left-to-right correspondence between input and output. Consequently, the search can proceed in a monotone fashion, processing the input left to right. This is not the case for MT. Even for language pairs as similar as French and English, there is rarely a strict left-to-right correspondence between input and output. We address this problem by allowing the MT decoder to consume its input in any order. This allows us to build the solution from left to right, regardless of word order differences. On the downside, it significantly increases the decoding complexity: instead of just one input string, we must consider up to $n!$ permutations of an n-word input sentence.

The other difference concerns the **heuristic function** that estimates the cost of completing partial hypotheses. This estimate allows us to compare the value of different partial hypotheses, and thus to focus the search in the most promising direction. It is important for this heuristic function to be as accurate as possible, particularly for a single-stack decoder. If it severely underestimates the completion cost, short hypotheses (those that cover a smaller portion of the input) will usually score higher than longer ones, simply because they cover fewer of the input events, each of which adds to the cost. In this case, the search is inefficient at best (if the stack is large enough to hold *all* hypotheses), and might even never finish in the worst case (if longer, more complete hypotheses are consistently "pushed off" the stack by shorter ones under real-world limitations). If the heuristic function overestimates the completion cost, the best solution might be suppressed due to this overestimation (cf. [9]). A good heuristic function is therefore crucial to the success of a single-stack decoder.

In SR, where the input is processed strictly left-to-right, a simple yet reliable class of heuristics can be used that estimates the completion cost based on the amount of input left to decode. In contrast, such heuristics are much more difficult to develop for MT [9,13], partly because there is no strict left-to-right correspondence.

At the time we implemented our decoders and conducted our experiments, we were not aware of any good heuristic function for MT. Our stack decoder compensates for the lack of a good heuristic function by using multiple stacks, one for each subset of the set of input words. Thus, each hypothesis in each stack competes only against hypotheses that cover the same portion of the input. But how does the decoder decide which hypothesis to extend during each iteration? We address this issue by simply extending the top hypothesis from each stack. It is obvious that this approach is still very inefficient. A better solution would be to somehow compare hypotheses from different stacks and extend only the best ones.

Our multistack decoder is closely patterned on the Model 3 decoder described by Brown et al. [2]. We gradually build more and more complete solution hypotheses repeatedly executing one of the following four operations, until the entire input is accounted for.

- **Add:** add a new English word and align a single French word to it.
- **AddZfert:** add two new English words. The first has fertility zero, while the second is aligned to a single French word.
- **Extend:** align an additional French word to the most recent English word, increasing its fertility.
- **AddNull:** align a French word to the NULL word.

Of the operations, **Extend** and **AddNull** are the least expensive, depending only on the number of unaligned French words in the hypothesis. **Add** is more expensive, as we must consider not only each unaligned French word in the hypothesis, but all possible English translations of each one. In practice, we restrict the search to considering only the ten candidates with the highest $t(e \mid f)$. Consequently, **Add** is ten times as expensive as **Extend** and **AddNull**.

At first glance, **AddZfert** is by far the most expensive operation. Since we are considering ten candidates for non-zero-fertility insertion, we must form 10 **AddZfert** hypotheses for each candidate considered for zero-fertility insertion. Naturally, we want to keep the list of zero-fertility candidates as small as possible without forfeiting good zero-fertility insertions.

The first thing we can do is to consider only English words for zero-fertility insertion that both occur frequently and have a high probability of being infertile. Secondly, we need to consider only zero-fertility word insertions that will increase the probability of a hypothesis, namely those that increase $P(\mathbf{e})$ more than they decrease $P(a, \mathbf{f} \mid \mathbf{e})$.[6] In the case of a bigram language model, this is completely safe, as the entire context needed to determine the effect of the zero-fertility insertion (one word to the left and one to the right) is known at the time we make the decision. As one of our reviewers pointed out, the optimal insertion of (any number of) infertile words could even be precomputed for each pair of (fertile) English words when using a bigram LM, so that **Add** and **AddZfert** could be conflated into one operation that inserts the optimal (possibly empty) sequence of infertile words before it adds a fertile word. With a trigram model, things are not quite as trivial, since the second word of the right trigram context is not known at the time the operation is performed. We conjecture that using the bigram lookup to determine the optimal zero-fertility insertion would probably be a good approximation when using a trigram LM.

By only considering helpful zero-fertility insertions, we can avoid significant overhead in the **AddZfert** operation, in many cases eliminating all possibilities and reducing its cost to less than that of **AddNull**.

The advantage of the stack decoder is that it explores a much larger portion of the search space than the greedy decoder, while running faster than the optimal decoder (both

[6] The latter is necessarily true, since every infertile word introduces an additional factor $n(0 \mid e_i) < 1$ into the calculation of $P(a, \mathbf{f} \mid \mathbf{e})$.

discussed below). Also, it can employ trigrams in the language model, while the optimal decoder is limited to bigrams. The disadvantage is that its time and space complexity are exponential in the length of the input sentence. As our experimental results show, in practice, the stack decoder cannot be used to translate sentences that are more than 20 words long.

5. Greedy decoding

Since the decoding problem is NP-hard [7], we can expect the time required for optimal decoding to increase exponentially with the input length. However, research has shown that for many instances of NP-hard problems, acceptable solutions can be found in polynomial time using greedy methods [8,11]. Instead of deeply probing the search space, these algorithms typically start out with a random, approximate solution and then try to improve it incrementally until a satisfactory solution is reached. In many cases, greedy methods quickly yield surprisingly good solutions.

We conjecture that such greedy methods may prove to be helpful in the context of MT decoding. Our greedy decoding algorithm starts with an English gloss of the French input sentence. The gloss is constructed by aligning each French word f_j with its most likely English translation $e_{a(j)} = \arg\max_e t(e \mid f_j)$. (Note that we use the 'direct' translation probability for constructing the gloss but the 'indirect' probability to evaluate the translation/alignment.) For example, in translating the French sentence "*Bien entendu , il parle de une belle victoire.*", the greedy decoder initially assumes that a good translation of it is "*Well heard, it talking a beautiful victory.*" because the best translation of "*bien*" is "*well*", the best translation of "*entendu*" is "*heard*", and so on. The alignment corresponding to this translation is shown at the top of Fig. 2.

Once the initial alignment is created, the greedy decoder tries to improve it, i.e., find an alignment (and implicitly translation) of higher probability, by applying one of the following operations:

- **translateOneOrTwoWords**$(j, e'_{a(j)}, k, e'_{a(k)})$ changes the translation of one or two French words, those located at positions j and k, from $e_{a(j)}$ and $e_{a(k)}$ into $e'_{a(j)}$ and $e'_{a(k)}$. If $e_{a(j)}$ is a word of fertility 1 and $e'_{a(j)}$ is NULL, then $e_{a(j)}$ is deleted from the translation. If $e_{a(j)}$ is the NULL word, or $\phi_{a(j)} > 1$, the word $e'_{a(j)}$ is inserted into the translation at the position that yields the alignment of highest probability. Fertilities are adjusted accordingly. The equivalent holds for $e_{a(k)}$ and $e'_{a(k)}$. If $e_{a(j)} = e'_{a(j)}$ or $e_{a(k)} = e'_{a(k)}$, this operation amounts to changing the translation of a single word.
- **translateAndInsert**$(j, e'_{a(j)}, e_x)$ changes the translation of the French word located at position j from $e_{a(j)}$ into $e'_{a(j)}$ and simultaneously inserts word e_x at the position that yields the alignment of highest probability. Word e_x is selected from an automatically derived list of words with a high probability of having fertility 0. When $e_{a(j)} = e'_{a(j)}$, this operation amounts to inserting a word of fertility 0 into the alignment.
- **removeWordOfFertility0**(i) deletes the infertile word at position i in the current alignment.

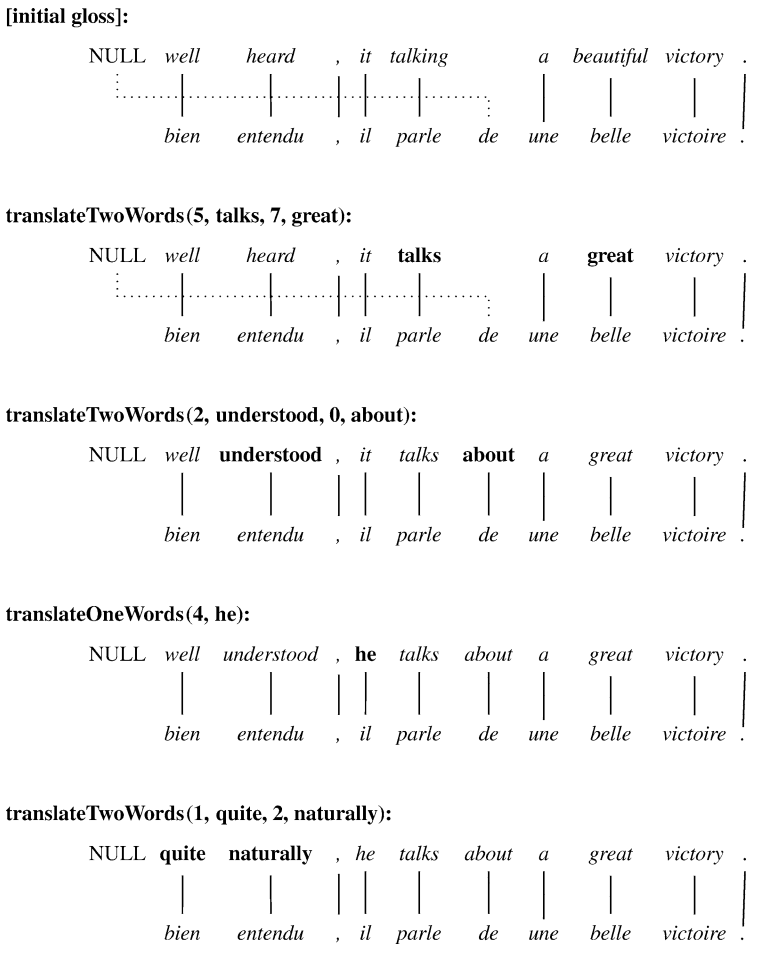

Fig. 2. Example of how the greedy decoder produces the translation of French sentence "*Bien entendu, il parle de une belle victoire.*"

- **swapSegments**(i_1, i_2, j_1, j_2) creates a new alignment from the old one by swapping non-overlapping English word segments $[i_1, i_2]$ and $[j_1, j_2]$. During the swap operation, all existing links between English and French words are preserved. The segments can be as small as a single word or as long as $|e| - 1$ words, where $|e|$ is the length of the English sentence.
- **joinWords**(i, j) eliminates from the alignment the English word at position i (or j) and links the French words generated by e_i (or e_j) to e_i (or e_j).

In a stepwise fashion, starting from the initial gloss, the greedy decoder iterates exhaustively over all alignments that are one operation away from the alignment under consideration. At every step, the decoder chooses the alignment of highest probability, until the probability of the current alignment can no longer be improved. When it starts

from the gloss of the French sentence "*Bien entendu, il parle de une belle victoire.*", for example, the greedy decoder alters the initial alignment incrementally as shown in Fig. 2, eventually producing the translation "*Quite naturally, he talks about a great victory.*". In the process, the decoder explores a total of 77421 distinct alignments/translations, of which "*Quite naturally, he talks about a great victory.*" has the highest probability.

We chose the operation types enumerated above for two reasons: (1) they are general enough to enable the decoder to escape local maxima and to modify a given alignment in a non-trivial manner in order to produce good translations; (2) they are relatively inexpensive (timewise). The most time consuming operations in the decoder are **swapSegments, translateOneOrTwoWords,** and **translateAndInsert**. **SwapSegments** iterates over all possible non-overlapping span pairs that can be built on a sequence of length $|e|$. **TranslateOneOrTwoWords** iterates over $|f|^2 \times |t|^2$ alignments, where $|f|$ is the size of the French sentence and $|t|$ is the number of translations we associate with each word (in our implementation, we limit this number to the top 10 translations). **TranslateAndInsert** iterates over $|f| \times |t| \times |z|$ alignments, where $|z|$ is the size of the list of words with a high probability of having fertility 0 (typically 1024; 128 in the experiments in Section 7).

Section 7 reports results that concern two versions of the greedy decoder—one version applies all operations described in this section while the other is optimized for speed.

The main advantage of the greedy decoder comes from its speed. As our experiments demonstrate, the greedy decoder can produce translations much faster than the other decoders. The greedy decoder is a typical instance of an "anytime algorithm": the longer it runs, the better the translation it finds. The main disadvantage of the greedy decoder pertains to the size of the space it explores, which is very small. The farther away a good translation is from a gloss, the less likely the greedy decoder is to find it.

6. Integer programming decoding

Knight [7] likens MT decoding to finding optimal tours in the Traveling Salesman Problem (TSP; Garey and Johnson [4])—choosing a good word order for decoder output is similar to choosing a good TSP tour. Because any TSP instance can be transformed into a decoding problem instance, Model 4 decoding is provably NP-hard in the length of **f**. It is interesting to consider the reverse direction—is it possible to transform a decoding problem instance into a TSP instance? If so, we may take great advantage of previous research into efficient TSP algorithms. We may also take advantage of existing software packages, obtaining a sophisticated decoder with little programming effort.

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**. A sample integer program (IP) looks like this:

```
minimize objective function:
  3.2 * x1 + 4.7 * x2 - 2.1 * x3
subject to constraints:
  x1 - 2.6 * x3 > 5
  7.3 * x2 > 7
```

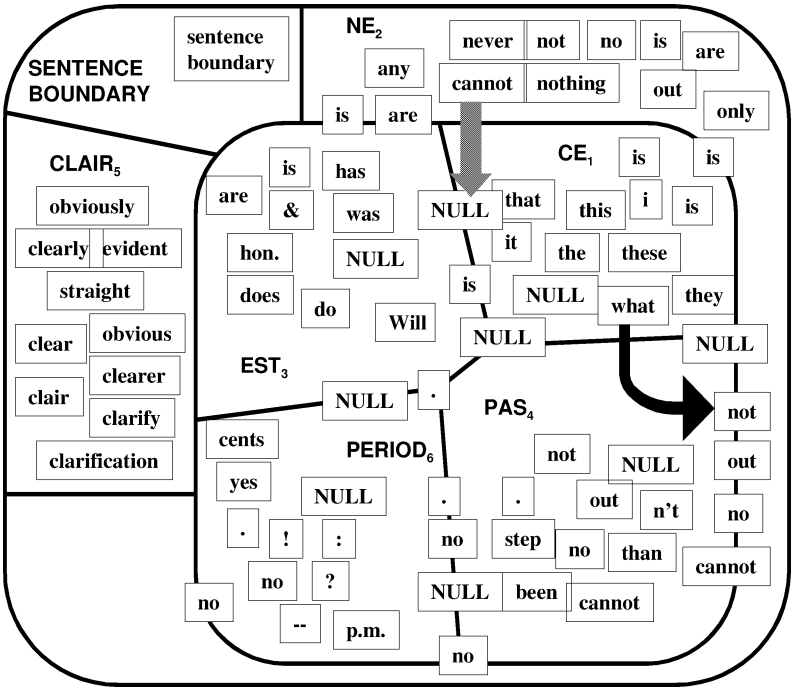

Fig. 3. A **salesman graph** for the input sentence **f** = "*CE NE EST PAS CLAIR.*" A few hotels are omitted for readability, e.g., the *NULL* hotel at the intersection of *EST*, *PAS*, and *PERIOD*.

A solution to an IP is an assignment of integer values to variables. Solutions are constrained by inequalities involving linear combinations of variables. An optimal solution is one that respects the constraints and minimizes the value of the objective function, which is also a linear combination of variables. We can solve IP instances with generic problem-solving software such as **lp_solve** or **CPLEX**.[7] In this section we explain how to express MT decoding (Model 4 plus English bigrams) in IP format.

We first create a **salesman graph** like the one in Fig. 3. To do this, we set up a **city** for each word in the observed sentence **f**. City boundaries are shown with bold lines. We populate each city with ten **hotels** corresponding to ten likely English word translations. Hotels are shown as small rectangles. The **owner** of a hotel is the English word inside the rectangle. If two cities have hotels with the same owner x, then we build a third x-owned hotel on the border of the two cities. More generally, if n cities all have hotels owned by x, we build $2^n - n - 1$ new hotels (one for each non-empty, non-singleton subset of the cities) on various city borders and intersections. Finally, we add an extra city representing the sentence boundary and populate it with one hotel.

We define a **tour of cities** as a sequence of hotels (starting at the sentence boundary hotel) so that each city is visited exactly once before returning to the start. If a hotel sits on

[7] Available at ftp://ftp.ics.ele.tue.nl/pub/lp_solve and http://www.cplex.com.

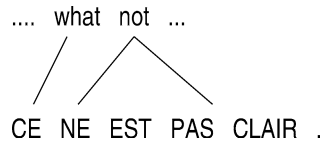

Fig. 4. Example of partial word alignment.

the border between two cities, then staying at that hotel counts as visiting **both** cities. We can view each tour of cities as corresponding to a potential decoding $\langle \mathbf{e}, a \rangle$. The owners of the hotels on the tour give us **e**, while the hotel locations yield a.

The next task is to establish real-valued (asymmetric) distances between pairs of hotels, such that the length of any tour is exactly $-\log(P(\mathbf{e}) \cdot P(a, \mathbf{f} \mid \mathbf{e}))$. Because log is monotonic, the shortest tour will correspond to the likeliest decoding.

The distance we assign to each pair of hotels consists of some small piece of the Model 4 formula. The usual case is typified by the large black arrow in Fig. 3. Because the destination hotel "*not*" sits on the border between cities "*NE*" and "*PAS*", it corresponds to a partial alignment in which the word "*not*" has fertility two (see Fig. 4).

If we assume that we have already paid the price for visiting the "*what*" hotel, then our inter-hotel distance need only account for the partial alignment concerning "*not*":

$$\begin{aligned} \textit{distance} = &-\log\big(bi(\text{not}|\text{what})\big) - \log\big(n(2 \mid \text{not})\big) \\ &- \log\big(t(\text{NE} \mid \text{not})\big) - \log\big(t(\text{PAS} \mid \text{not})\big) \\ &- \log\big(d_1(+1 \mid \textit{class}(\text{what}), \textit{class}(\text{NE}))\big) \\ &- \log\big(d_{>1}(+2 \mid \textit{class}(\text{PAS}))\big) \end{aligned}$$

where $bi(\text{not}|\text{what})$ is the language model probability of *not* following *what*.

NULL-owned hotels are treated specially. We require that all non-NULL hotels be visited before any NULL hotels, and we further require that at most one NULL hotel be visited on a tour. (This is accomplished by establishing a zero distance from a NULL hotel to the sentence boundary hotel, and an infinite distance to any other destination.) Note that if we were to allow travel from a NULL hotel to a regular hotel, we would have already lost the state information required for assigning bigram and head-distortion costs. Moreover, the NULL fertility sub-formula is easy to compute if we allow only one NULL hotel to be visited: ϕ_0 is simply the number of cities that this hotel straddles, and m is the number of cities minus one. This case is typified by the large straight arrow shown in Fig. 3. The cost of this segment is:

$$\begin{aligned} \textit{distance} = &-\log\binom{6-2}{2} - 2 \cdot \log(p_1) - (6-4)\log(1-p_1) \\ &- \log\big(t(\text{CE} \mid \text{NULL})\big) - \log\big(t(\text{EST} \mid \text{NULL})\big) \\ &- \log\big(bi(\text{sentence-boundary}|\text{cannot})\big). \end{aligned}$$

The last term takes care of the final bigram.

Between hotels that are located (even partially) in the same city, we assign an infinite distance in both directions, as travel from one to the other can never be part of a tour. For

6-word French sentences, we normally come up with a graph that has about 80 hotels and 3500 finite-cost travel segments.

So far we have glossed over the issue of zero-fertility words. If we disallow adjacent zero-fertility words, then we need only allow for the possibility of inserting a single zero-fertility word when en route from one hotel to another. We can choose from the possibilities $\{none, z_1, z_2, \ldots\}$ purely locally by comparing bigram and $n(0 \mid e)$ probabilities, as mentioned in Section 5. We decide which zero-fertility words to emit (if any) between different pairs of hotels, and we take distances to be those produced by these choices. This adds neither hotels nor travel segments of a graph, but requires additional computation for each segment.

The next step is to cast tour selection as an integer program. Here we adapt a **subtour elimination** strategy used in standard TSP. We create a binary (0/1) integer variable x_{ij} for each pair of hotels i and j. $x_{ij} = 1$ if and only if travel from hotel i to hotel j is on the itinerary. The objective function is straightforward:

$$\text{minimize:} \quad \sum_{(i,j)} x_{ij} \cdot \mathit{distance}(i, j).$$

This minimization is subject to three classes of constraints. First, every city must be visited exactly once. That means exactly one tour segment must exit each city:

$$\forall_{c \in \mathit{cities}}: \sum_{i \text{ located at least partially in } c} \sum_{j} x_{ij} = 1.$$

Second, the segments must be linked to one another, i.e., every hotel has either (1) one tour segment coming in and one going out, or (2) no segments in and none out. To put it another way, every hotel must have an equal number of tour segments going in and out:

$$\forall_i: \sum_j x_{ij} = \sum_j x_{ji}.$$

Third, it is necessary to prevent multiple independent sub-tours. To do this, we require that every proper subset of cities have at least one tour segment leaving it:

$$\forall_{s \subset \mathit{cities}}: \sum_{i \text{ located entirely within } s} \sum_{j \text{ located at least partially outside } s} x_{ij} \geqslant 1.$$

There are an exponential number of constraints in this third class.

Finally, we invoke our IP solver. If we assign mnemonic names to the variables, we can easily extract $\langle \mathbf{e}, a \rangle$ from the list of variables and their binary values. The shortest tour for the graph in Fig. 3 corresponds to this optimal decoding: *it is not clear.*

We can obtain the second-best decoding by adding a new constraint to the IP to stop it from choosing the same solution again—if the optimal tour consists of k segments, we require that the sum of the variables corresponding to the segments be less than k. We can create a list of n-best solutions simply by repeating this procedure.[8] If we simply

[8] Strictly speaking, this may not be a true n-best list as our formulation makes available only one zero-fertility choice between each pair of hotels.

replace "minimize" with "maximize," we can obtain the longest tour, which corresponds to the **worst** decoding, in this case: "*clair hon. i ! than are.*" Finding the worst translation is somewhat more time-consuming than finding the best, as there appears to be more competition.

We see a number of advantages to the IP approach in general:

(1) A decoder can be built very rapidly, with very little programming, thus helping to validate a proposed linguistic model.
(2) Optimal n-best results can be obtained.
(3) Generic problem solvers offer a range of user-customizable search strategies, thresholds, etc.

There are also a number of disadvantages:

(1) Other knowledge sources (e.g., wider English context) may not be easily integrated;
(2) Performance is slow.

7. Experiments and discussion

For consistency, the experiments reported in this section were set up so that all decoders worked on the same search space. The integer programming decoder explores this space exhaustively. The stack and greedy decoders explore only a portion of it. In all experiments, we decoded using only the top ten translations of a word,[9] as determined during training, and a list of 128 words of fertility 0, which was also extracted automatically from the corpus. For the experiments reported in Table 1, we used a bigram language model. The results reported in Table 2 were obtained using a trigram model.

The test collection consists of 505 sentences, uniformly distributed across the lengths 6, 8, 10, 15, and 20. We evaluated all decoders with respect to (1) speed, (2) search optimality, and (3) translation accuracy. The last two factors may not always coincide, as Model 4 is an imperfect model of the translation process—there is no guarantee that a numerically optimal decoding is actually a good translation.

Let $\hat{\mathbf{e}}$ be the optimal decoding and $\mathbf{e}'$ the best decoding found by a decoder. We consider six possible outcomes:

Error classification	$\mathbf{e}' = \hat{\mathbf{e}}$	$\hat{\mathbf{e}}$ is perfect	$\mathbf{e}'$ is perfect
1. no error (NE)	yes	yes	yes
2. pure model error (PME)	yes	no	no
3. deadly search error (DSE)	no	yes	no
4. fortuitous search error (FSE)	no	no	yes
5. harmless search error (HSE)	no	yes	yes
6. compound error (CE)	no	no	no

[9] According to $t(e|f)$.

Table 1
Comparison of decoders on sets of 101 test sentences. All experiments in this table use a bigram language model. Translation errors can be syntactic, semantic, or both. Errors were counted on the sentence level, so that every sentence can have at most one error in each category

len	decoder	time	SE	TE	NE	PME	DSE	FSE	HSE	CE	BLEU
6	IP	47.50	0	57	44	57	0	0	0	0	0.206
6	stack	0.79	5	58	43	53	1	0	0	4	0.209
6	greedy	0.07	18	60	38	45	5	2	1	10	0.197
8	IP	499.00	0	76	27	74	0	0	0	0	0.157
8	stack	5.67	20	75	24	57	1	2	2	15	0.162
8	greedy	2.66	43	75	20	38	4	5	1	33	0.147

len: input sentence length; **time:** average translation time (in sec./sent.); **SE:** search errors; **TE:** translation errors; **NE:** no error; **PME:** pure model errors; **DSE:** deadly search errors; **FSE:** fortuitous search errors; **HSE:** harmless search errors; **CE:** compound error; **BLEU:** score according to the IBM BLEU metric.

For the purpose of this evaluation, a translation is judged perfect if it (1) renders the full meaning of the input sentence in the translation, and (2) is flawless English. These judgments were made by a human evaluator.

We have found it very useful to have several decoders on hand. It is only through IP decoder output, for example, that we can know the stack decoder is returning optimal solutions for so many sentences (see Table 1). The IP and stack decoders enabled us to quickly locate bugs in the greedy decoder, and to implement extensions to the basic greedy search that can find better solutions. (We came up with the greedy operations discussed in Section 5 by carefully analyzing error logs of the kind shown in Table 1.) The results in Table 1 also enable us to prioritize the items on our research agenda. Since the majority of the translation errors can be attributed to the language and translation models we use (see column PME in Table 1), it is clear that significant improvement in translation quality will come from better models.

In addition to the subjective evaluation, we also assessed the decoders' performance with the IBM BLEU metric [10]. The BLEU score is an automatic measure of MT quality that is based on the degree of overlap between n-grams in a candidate translation and one ore more (human) reference translations. In our experimental setting, we used only one reference translation. While the BLEU scores reflect the rank order of our subjective evaluation well, we provide them primarily as "ballpark figure" estimates of the decoders' performance—they should not be considered an accurate measure of performance for test corpora of the size used in our evaluation.

The most interesting conclusion that we can draw from the numbers in Tables 1 and 2 is that even though the numbers of search errors differ significantly between the decoders (column SE in Table 1), even for this small test set, the measures of translation quality do not (Tables 1 and 2).

Depending on the application of interest, one may choose to use a slow decoder that provides optimal results or a fast, greedy decoder that provides non-optimal but acceptable results. One may also run the greedy decoder using a time threshold, as an instance of an anytime algorithm. When the threshold is set to one second per sentence (the greedy$_1$ label in Table 2), the performance is affected only slightly.

Table 2
Comparison between decoders using a trigram language model. Greedy* and greedy$_1$ are greedy decoders optimized for speed

Length	Decoder	Av. time (in sec./sent.)	Erroneous translations	BLEU
6	stack	13.72	42	0.282
6	greedy	1.58	46	0.226
6	greedy*	0.07	46	0.202
8	stack	45.45	59	0.231
8	greedy	2.75	68	0.188
8	greedy*	0.15	69	0.174
10	stack	105.15	57	0.271
10	greedy	3.83	63	0.247
10	greedy*	0.20	68	0.225
15	stack	>2000	74	0.225
15	greedy	12.06	75	0.202
15	greedy*	1.11	75	0.190
15	greedy$_1$	0.63	76	0.189
20	greedy	49.23	86	0.219
20	greedy*	11.34	93	0.217
20	greedy$_1$	0.94	93	0.209

Acknowledgement

This work was supported by DARPA-ITO grant N66001-00-1-9814.

References

[1] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, P. Roossin, A statistical approach to machine translation, Comput. Linguistics 16 (2) (1990) 79–85.
[2] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lai, R. Mercer, Method and system for natural language translation, US Patent 5,477,451, 1995.
[3] P. Brown, S. Della Pietra, V. Della Pietra, R. Mercer, The mathematics of statistical machine translation: Parameter estimation, Computat. Linguistics 19 (2) (1993) 263–311.
[4] M. Garey, D. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.
[5] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. System Sci. Cybernet. 4 (2) (1968) 100–107.
[6] F. Jelinek, A fast sequential decoding algorithm using a stack, IBM Res. J. Res. Development 13 (1969) 675–685.
[7] K. Knight, Decoding complexity in word-replacement translation models, Comput. Linguistics 25 (4) (1999) 607–615.
[8] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky, Determining computational complexity from characteristic 'phrase transitions', Nature 800 (8) (1999) 133–137.
[9] F. Och, N. Ueffing, H. Ney, An efficient A* search algorithm for statistical machine translation, in: Proceedings of the ACL Workshop on Data-Driven Machine Translation, Toulouse, France, 2001, pp. 55–62.
[10] K. Papineni, S. Roukos, T. Ward, J. Henderson, F. Reeder, Corpus-based comprehensive and diagnostic MT evaluation: Initial Arabic, Chinese, French, and Spanish results, in: Proceedings of the Human Language Technology Conference, San Diego, CA, 2002, pp. 124–127.

[11] B. Selman, H. Levesque, D. Mitchell, A new method for solving hard satisfiability problems, in: Proceedings of AAAI-92, San Jose, CA, 1992, pp. 440–446.
[12] C. Tillmann, S. Vogel, H. Ney, A. Zubiaga, A DP-based search using monotone alignments in statistical translation, in: Proceedings of the 35th ACL, 1997, pp. 289–296.
[13] Y. Wang, A. Waibel, Decoding algorithm in statistical machine translation, in: Proceedings of the 35th ACL, 1997, pp. 366–372.
[14] D. Wu, A polynomial-time algorithm for statistical machine translation, in: Proceedings of the 34th ACL, 1996, pp. 152–158.