Appeared in *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pp. 57–75, 2006.

Local Search with Very Large-Scale Neighborhoods for Optimal Permutations in Machine Translation*

Jason Eisner and **Roy W. Tromble**
Department of Computer Science and Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218
{jason,royt}@cs.jhu.edu

Abstract

We introduce a novel decoding procedure for statistical machine translation and other ordering tasks based on a family of Very Large-Scale Neighborhoods, some of which have previously been applied to other NP-hard permutation problems. We significantly generalize these problems by simultaneously considering three distinct sets of ordering costs. We discuss how these costs might apply to MT, and some possibilities for training them. We show how to search and sample from exponentially large neighborhoods using efficient dynamic programming algorithms that resemble statistical parsing. We also incorporate techniques from statistical parsing to improve the runtime of our search. Finally, we report results of preliminary experiments indicating that the approach holds promise.

1 Introduction

Statistical machine translation (SMT) must choose an apt ordering for its output words. Similarly, multi-document extractive summarization must choose an apt ordering for its output sentences. Yet optimal ordering is computationally difficult for any reasonable definition of "optimal"; the classic example is the Traveling Salesperson Problem (TSP).

Fortunately, ordering problems are important enough that there is a literature on practical techniques to solve them (exactly or approximately). Our goal in this paper is to extend such techniques and show that they are relevant to natural language processing. We show the relevance of a particular novel class of ordering problems, and then introduce a non-trivial new heuristic algorithm for this class.

We make the following specific contributions:

1. **Formalization:** We introduce a flexible "ABC" model for scoring permutations.

2. **Application:** We reformulate SMT decoding as seeking an optimal permutation of the *input* words. This permuted input can then be translated by a simple finite-state transducer.

3. **Approach:** We propose seeking the optimal permutation via a "local search" strategy of iteratively improving the current permutation (Germann et al., 2001). This provides an alternative to the usual beam search methods.

4. **Decoding algorithm:** We show how this local search can consider an *exponentially large* set of candidate improvements at each iteration, efficiently searching this set by dynamic programming. This has previously been done for the TSP (Deĭnenko and Woeginger, 2000), but our new algorithm handles the full ABC model.

5. **Speedups:** We discuss how to speed up our dynamic programming algorithm, which resembles CKY parsing, by applying techniques from natural-language parsing.

6. **Training:** We briefly discuss how to train the costs of the ABC model.

This paper focuses on the novel models and algorithms. Other than a few suggestive numbers, we leave experimental validation to future work.

*This material is based upon work supported by the National Science Foundation under Grants No. 0347822 and 0313193.

2 The Formal Problem

Our algorithmic problem is to search for a minimum-cost permutation of the integers $\{1, 2, \ldots, N\}$, for a given $N > 0$ and a given cost function. We begin by describing the rather broad family of cost functions that our algorithms can handle.

2.1 Notation

A permutation π is a bijective function from $\{1, 2, \ldots, N\}$ to $\{1, 2, \ldots, N\}$. It is often convenient to identify π with the *sequence* $\pi(1), \pi(2), \ldots, \pi(N)$. We therefore usually write $\pi(i)$ as π_i, the i^{th} element of this sequence.

We write π_i^j for the contiguous subsequence $\pi_{i+1}\pi_{i+2}\ldots\pi_j$. For example, $\pi_{j-1}^j = \pi_j$ and $\pi_0^N = \pi$.

As a convention, we will use i, j, k as indices into π. We will use ℓ, m, r, o to refer to the outputs of the π function, as in "$\pi_i = \ell$." Commonly ℓ, m, r, o can be respectively interpreted as "left," "middle," "right," "outside."

Our running example in this paper will be $\pi =$ 1 4 2 5 6 3, drawn from a small machine translation example to be discussed later (Fig. 2).

2.2 Scoring permutations: The ABC model

We wish to find the permutation π that minimizes the total cost

$$A(\pi) + B(\pi) + C(\pi) \tag{1}$$

We search only over π for which this cost is finite; any of the three summands may be infinite.

This "ABC" cost model combines the three kinds of costs that we know at present how to handle algorithmically. For some applications, it is only necessary to use just the A or the B term, but we prefer to present our algorithm in its most general form, both for expository reasons and because it may be useful.

Automaton cost. $A(\pi)$ is defined by a given weighted finite-state acceptor (WFSA) over the alphabet $\{1, 2, \ldots, N\}$. A WFSA is simply an FSA annotated with real-valued costs. The cost of a path in A is the sum of its arc costs, plus a halting cost on its final state. $A(\pi)$ is defined as the minimum cost of any path in A that accepts the sequence π (e.g., 1 4 2 5 6 3). If there is no such path, then $A(\pi) = \infty$.

The WFSA does not need to detect whether the sequence π is actually a permutation, so it need not have size exponential in N (Fig. 1). It is free to *accept* sequences that are not permutations, but it will only be *used* (by equation (1)) to score true permutations π.

Beforeness cost. $B(\pi)$ is defined as $\sum_{i<k} b_{\pi_i,\pi_k}$, where B is an arbitrary given $N \times N$ cost matrix. This is a sum of $\binom{N}{2}$ pairwise costs. For example, $B(1\,4\,2\,5\,6\,3) = b_{1,4}+b_{1,2}+b_{1,5}+b_{1,6}+b_{1,3}+b_{4,2}+b_{4,5}+b_{4,6}+b_{4,3}+b_{2,5}+b_{2,6}+b_{2,3}+b_{5,6}+b_{5,3}+b_{6,3}$.

Cyclic cost. $C(\pi)$ generalizes B to triples. It is defined as $\sum_{i<j<k} c_{\pi_i,\pi_j,\pi_k}$, a sum of $\binom{N}{3}$ costs. C is a given $N \times N \times N$ array.

Although we show in the appendix how to drop this restriction, for algorithmic reasons we require C to have a kind of cyclic symmetry property: $c_{\ell,m,r} = c_{m,r,\ell} = c_{r,\ell,m}$. Thus, a cost equal to $c_{2,6,3}$ will actually be incurred by any permutation of the form $\ldots 2 \ldots 6 \ldots 3 \ldots$, $\ldots 3 \ldots 2 \ldots 6 \ldots$, or $\ldots 6 \ldots 3 \ldots 2 \ldots$. These are permutations in which 2, 6, 3 can be found in that order if the sequence is read "cyclically." Again, the appendix shows how to drop this restriction (at some efficiency penalty).

We remark that the cost model (1) is asymmetric, in the sense that there may be no reversed model A', B', C' such that $A'(\pi^{-1}) + B'(\pi^{-1}) + C'(\pi^{-1})$ gives the same score for all π. Thus, even if there exists a good model of how French reorders into English, say, there may not exist as good a model of the reverse. (This is also true of the IBM translation models (Brown et al., 1993).) In practice, however, we hope that (1) is flexible enough to admit reasonable models in both directions.

2.3 Relation to difficult classical problems

Consider the special case of equation (1) where $B = 0, C = 0$, and A is a "bigram model" such that $A(1\,4\,2\,5\,6\,3)$ is a sum of the form $d(1,4) + d(4,2) + d(2,5) + d(5,6) + d(6,3)$. Fig. 1 shows how to encode such a bigram model as a WFSA for the case $N = 3$.

Finding the optimal π is now a variant of the clas-

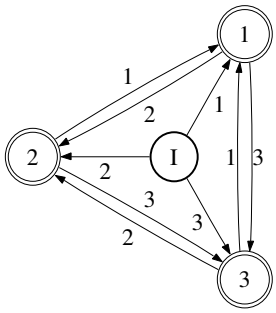

Figure 1: A bigram automaton for a sentence of length $N = 3$, with costs not shown. State "I" is the initial state; each other state is named after the last symbol read before entering that state. This FSA does accept sequences that are not permutations of 1 2 3, but does not bother to include self-loops, since a permutation could never use them.

sical Traveling Salesperson Problem (TSP).[1] The $d(\cdot, \cdot)$ terms represent city-to-city travel costs. The only difference from the usual TSP is that we have omitted the cost $d(3, 1)$ for the salesperson to return home. The usual TSP can be encoded by a simple change to Fig. 1: arbitrarily require 1 to be the initial city of π, and set the halting cost for each state r to be $d(r, 1)$, the cost of returning to that initial city.

Consider also the special case of equation (1) where $A = 0$ and $C = 0$. This is the classical Linear Ordering Problem (LOP): $\min_\pi \sum_{i<k} b_{\pi_i, \pi_k}$ where B is an arbitrary cost matrix.

Both the TSP and the LOP are well-known to be NP-complete in their decision versions. Worse, the minimization versions that concern us are believed to be inapproximable to within any constant factor.[2] Nonetheless, these are only worst-case results, and there have been many practical attacks on such problems. In section 4 we will present our own approach.

2.4 Applications

The methods that we present are applicable to many ordering problems that arise within computer science. For example, the LOP has applications in economics, sociology, graph theory, archaeology, and task scheduling (Grötschel et al., 1984), as well as graph drawing (Eades and Whitesides, 1994). Other useful NP-complete problems, such as the weighted feedback arc set or acyclic subgraph problem, reduce trivially to LOP (Grötschel et al., 1984).

Within natural language processing, our model and algorithm could be applied to ordering words or sentences during text generation. One scenario is multi-document summarization. Lapata (2003) proposed scoring sequences of sentences according to a bigram model. Choosing the optimal sequence is then equivalent to the TSP: the sentences take the place of cities, and the bigram probabilities the costs of traveling between them. Our more flexible "ABC" model allows consideration of other kinds of costs as well. Our A costs would be needed for the improved "content model" of Barzilay and Lee (2004), an HMM-like model with hidden states. Our B costs would express additional discourse preferences that a particular sentence is prerequisite to another.

Other NLP problems also seek optimal permutations. Machine translation output could be evaluated—generalizing a recent proposal of Leusch et al. (2006)—by whether some "low-distortion" permutation of the output scores well under a small language model derived from a set of reference translations. Information extraction systems may wish to reconstruct a temporal order for the extracted events (Mani et al., 2003; Mann, 2006); resolving conflicting cues to the true order may be treated as an instance of the LOP (Glover et al., 1974). Phonology learning involves the computationally hard problem of choosing a rule ordering or constraint ranking (Eisner, 2000); our "ABC" cost model might be used here to approximate the true cost of an ordering or ranking. Finally, one might wish to find multi-word or non-word anagrams that score well under a language model (Morton, 1987; Jordan and Monteiro, 2003).

We now discuss one application in detail: statistical machine translation.

3 Input Reordering for Translation[3]

Word reordering is a central part of statistical machine translation (SMT). We show that under a fam-

[1] Namely, the weighted Hamiltonian path problem.

[2] MIN-TSP is known to be NPO-complete (Orponen and Mannila, 1987), and MIN-LOP is conjectured to be outside APX (Mishra and Sikdar, 2004). We are unaware of previous work on $C(\pi)$, but it is at least as hard as $B(\pi)$, i.e., the LOP.

[3] Readers who are not primarily interested in SMT may wish to skip ahead to the algorithm in section 4.

ily of models we propose (which appears to include IBM Model 4), it is possible to *reduce* the SMT problem to the ordering problem of section 2.

3.1 An $S \circ P \circ T$ pipeline

We regard translation from language f ("French") to language e ("English") as a nondeterministic pipeline (Fig. 2):

- **Input:** A "surface" source string f.
- **Source transduction (S):** Transduce f to a "deep" source string f', consisting of N symbols annotated sequentially with $1, 2, \ldots, N$.
- **Permutation (P):** Permute f' to obtain a "deep" target string e', consisting of the same N symbols in a different order, still annotated with their original positions in f'.
- **Target transduction (T):** Transduce e' to the final "surface" target string e.

Our intuition is that if translation were monotonic—i.e., no word reorderings—then it could be handled simply and reasonably well by a cascade of finite-state transducers (FSTs). These are quite capable on their own of breaking words into their component morphemes, performing monotonic translation of contiguous substrings ("phrases"), inserting or deleting material, reassembling the target language morphemes into words, and rescoring under a language model.

Thus, our "$S \circ P \circ T$" pipeline just introduces a permutation step P into the middle of a simple transducer cascade $S \circ T$, where $\circ$ denotes composition. (Longer cascades can be accommodated in our framework by defining $S = S_1 \circ S_2 \circ \cdots$ and $T = T_1 \circ T_2 \cdots$.) Modeling the permutations and searching over them seems to be the hardest part of translation.

3.2 An example

Fig. 2 shows a possible French-to-English translation. The example presupposes a specific model that partitions the work in a particular way among the S, P, and T steps. In particular, we have chosen to have the intermediate f' and e' strings look like marked-up French, rather than like English or a semantic interlingua.

In the example, the S step performed some morphological analysis and decomposition, including part-of-speech tagging. The P step brought *ne* together with *pas*, allowing *ne pas* to be translated as a phrase (into *not* or *n't*). It also moved *me* from French clitic position to English direct object position. Finally, the T step translated the morphemes phrasally in their new order.

Our S and P steps, together with the costs supplied by T, can be regarded as aggressively preprocessing French (f) into a "French-prime" (e') that still uses French words but can be more easily—indeed monotonically—translated into English (e). This recalls heuristic "analysis-transfer-synthesis" (Brown et al., 1992; Nießen and Ney, 2001), which makes the statistical transfer step easier to learn by linguistically hand-crafting a deterministic preprocessor (analysis) and postprocessor (synthesis). Our approach moves most of the work into the analysis stage ($S \circ P$). We make analysis (i.e., reordering) a *learned, nondeterministic, and rather free* model in its own right, which allows us to get away with monotonic transfer and synthesis (T).

3.3 Permutations, alignments, and fertilities

From Fig. 2 one can read off $\pi = 1\ 4\ 2\ 5\ 6\ 3$, meaning that the target e' consists of the first word of f', followed by the fourth word of f', etc.

In our SMT application, π maps each target position i in e' to the corresponding source position π_i in f'. This is the "reversed" or noisy-channel direction, which is also the direction of π in (Brown et al., 1993).

The "forward" direction is encoded by the inverted permutation $\pi^{-1} = 1\ 3\ 6\ 2\ 4\ 5$, which indicates each f' word's position in e'. Thus π^{-1} corresponds to the traditional alignment vector a (Brown et al., 1993). Of course, our π^{-1} is not an arbitrary IBM-style alignment, as it is strictly 1-to-1.

Fertility (many-to-1) is handled only later, by our T transduction from e' to e. Note that our T also allows many-to-many translations, unlike the IBM models. While phrase-based translation models do allow many-to-many translations (Koehn et al., 2003), they are usually limited to contiguous phrases, while our $S \circ P \circ T$ process is not. A contiguous phrase in the source f may be translated or annotated as a phrase by S, then scattered by P

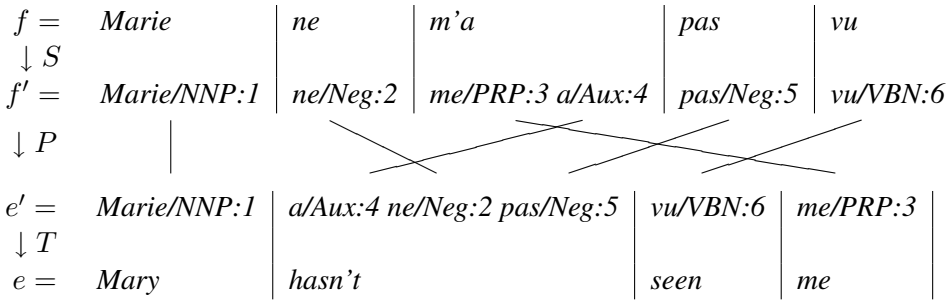

Figure 2: An example of French-to-English translation, involving the permutation $\pi = 1\ 4\ 2\ 5\ 6\ 3$. The French sentence may be glossed as *Mary NEG me'has not seen.* Diagonal lines show the permutation. Vertical separators suggest which "phrases" were transduced by S and T (though as arbitrary finite-state transducers, they need not segment a string into independently translated phrases). Other ways of achieving this translation are possible.

to become discontiguous in e' and e. Conversely, a contiguous phrase in the target e may be translated from a phrase in e' that was brought together from discontiguous words in f.[4] Note especially that S may shatter words of f (such as *m'a* in Fig. 2) and then let P permute the shards (*me a*), bringing together new phrases such as *a ne pas* (which translates as *hasn't*).[5]

3.4 Costs under the $S \circ P \circ T$ model

Given f, there are many possible choices of how to translate it. Of the choices allowed by one's model, some will be better (i.e., lower-cost) than others. We define the total cost of a set of choices as

$$tc(f, f', e', e) = S(f, f') + P(f', e') + T(e', e) \quad (2)$$

Given f, we seek values of the other variables to minimize this total cost. (Alignment is identical except that e is also given.)

The S cost. In this paper, we simplify the computational problem by assuming that S gives a *deterministic* mapping from f to f', such as the identity map or a deterministic tokenizer, tagger, or morphological analyzer. Since f is given, f' is effectively given as well, so the cost $S(f, f')$ is constant and hence unimportant.[6]

The T cost. T may be any finite-state transducer with weights in the tropical semiring $(\mathbb{R}, \min, +)$. What might it look like? A typical noisy-channel approach would define T to be a finite-state composition cascade of the form $(Lang \circ Trans \circ Align)^{-1}$, so that

$$T(e', e) = Lang(e) + Trans(e, e') + Align(e') \quad (3)$$

which models a reversed, English-to-French generative process:

- $Lang(e)$ evaluates the fluency of the English (*Mary hasn't seen me*, in Fig. 2),
- $Trans(e, e')$ evaluates the various English-to-French lexical correspondences (*hasn't* $\rightarrow$ an annotated version of *a ne pas*), and
- $Align(e')$ evaluates the English-to-French reordering, as indicated by the numeric annota-

[4] As our model has only one permutation step, it cannot do both at once: bring several scattered words of f' together, transduce them as a phrase, and then scatter the translated words again. Such an operation might be useful for translating one discontiguous verb-particle construction into another. Instead, the only way for us handle this kind of discontiguous-to-discontiguous translation is with the finite-state memory of T. An alternative (discussed in section 5.2) is to replace T by a synchronous grammar.

[5] Note that S used traditional morphological analysis to shatter *m'a* $\rightarrow$ *me a*. A formally adequate alternative would be for S to shatter *m'a* into the numbered shards *m'a*$_1$ *m'a*$_2$. This can be done mechanically for any source word that might translate non-contiguously (e.g., *retirer*$_1$ *retirer*$_2$ becomes *take ... out*). However, numbered shards increase the size of the vocabulary that P and T must learn from data to deal with, whereas traditional morphological analysis manages to reduce vocabulary size.

[6] Adapting our techniques to nondeterministic S, such as a WFSA, is considerably harder for our techniques. The construction we have in mind effectively requires T and hence A to enforce global constraints on the permutation π. Tromble and Eisner (2006) found it effective to fold global constraints into a WFSA on an as-needed basis, to avoid making the WFSA grow larger than necessary. The same approach might work here.

tions on e' (*1 4 2 5 6 3*).

In general, $Align$ may also wish to consider, say, the part-of-speech annotations on e' (*NNP:1 Aux:4 Neg:2 Neg:5 VBN:6 PRP:3*), since perhaps some reordering behaviors should be costly for a verb but not for an adjective.

It is possible to write the composition cascade more generally, replacing equation (3) with

$$T(e', e) = \min_{e_1, e_2} Lang(e, e_1) + Trans(e_1, e_2) + Align(e_2, e') \quad (4)$$

This allows the language model to annotate e nondeterministically and score this annotated version e_1, which is the version then considered by translation. Similarly, the translation model may annotate e_1 further, producing a further marked-up version e_2 that is evaluated by the alignment model. Thus $Lang$ and $Trans$ could encourage $Align$ to perform certain reorderings just if they make certain choices about how to analyze or translate the English. $Align$ deletes this extra, nondeterministic markup from e_2 to obtain e' (necessary since we are allowing e' to include only annotations that could be deterministically produced from f by S).

For instance, to simulate IBM Model 4 (Brown et al., 1993),[7] we would define all scores in equation (3) to be negated log-probabilities. The language model is defined by $Lang(e) \stackrel{\text{def}}{=} -\log \Pr_{\text{trigram}}(e)$. The translation model $Trans$ would separately translate each English word ("cept") of e to a bracketed list ("tablet") of French words at specified positions, such as *[a/Aux:4 ne/Neg:2 pas/Neg:5]* in Fig. 2, as well as inserting words aligned to the empty cept NULL. Finally, the alignment model $Align$ would, while deleting the brackets of this e_2 to get e', score the bracketed version by considering the differences between successive numbers within a tablet (4, 2, 5 in our example) and successive tablet-initial numbers (1, 4, 6, 3). Our formulation allows many variants of this model, including multi-word cepts, contextual translations, and more sophisticated $Align$ models.

[7] Knight and Al-Onaizan (1998) proposed similar treatments of IBM Models 1–3. Their approach further incorporated the permutation step P^{-1} into the composition cascade, as a very large "permutation acceptor" (see section 6). Our local search algorithm (section 4) can be regarded as a way to explore paths in this permutation acceptor without fully expanding it.

The P cost. How about P, which evaluates the permutation? It is not unreasonable to take $P = 0$, since as we have just seen, T too can evaluate the permutation by reading the numeric annotations on e' (e.g., via $Align$). However, T uses finite-state means only. Our algorithm in section 4.2 will also permit us to add certain non-local costs cheaply, if desired, so in the general case we put

$$P_{f'}(\pi) \stackrel{\text{def}}{=} B_{f'}(\pi) + C_{f'}(\pi) \quad (5)$$

and put $P(f', e') = P(\pi)$ where π denotes the unique permutation relating f' to e'.

The "beforeness cost" function $B_{f'}$ considers which pairs of words in f' (not necessarily adjacent) have reversed their order. The "cyclic cost" function $C_{f'}$ considers which triples of words in f' have changed their cyclic order (i.e., undergone an odd permutation). The form of these functions was given in section 2.2. It is an empirical question whether they can help translation, but we discuss some possible uses in section 3.6.

Note that this B matrix and C array may depend arbitrarily on f'. That is, when trying to translate a given f', we may set our costs to define which permutations we prefer of that particular f'.

3.5 Decoding as permutation

We now show how to reduce the $S \circ P \circ T$ decoding problem to a search over permutations.[17] To minimize equation (2) for a given f, we step through S, P, and T in that order:

- S **step:** Given f, compute f' as deterministically prescribed by S.
- P **step, with lookahead to the** T **step**: Given f', choose e' such that $P(f', e') + T(e', e)$ will be minimized given an optimal choice of e at the next step.
- T **step:** Given e', choose e to minimize $T(e', e)$. This translation e can simply be read off the minimum-weight path in $e' \circ T$, using Viterbi decoding.

The interesting part is the lookahead at the P step. We are seeking a permutation that is not only plausible as a French-to-English reordering,[8] but whose result e' can then be plausibly translated[9] to good English.[10]

To handle this lookahead, we define an "automaton cost" function $A_{f'}$ by

$$A_{f'}(\pi) \stackrel{\text{def}}{=} \min_{e} T(e', e) \tag{6}$$

where e' is the result of permuting f' by π. In other words, $A_{f'}$ scores the permutation π according to whether the permuted French e' that it produces would plausibly admit *any* plausible English translation e. It serves as a sort of proxy language model for permuted French.

Thus, the P step with lookahead should choose e' to minimize

$$P(f', e') + A_{f'}(\pi),$$

or equivalently, should choose π to minimize

$$A_{f'}(\pi) + B_{f'}(\pi) + C_{f'}(\pi) \tag{7}$$

This is just our formal problem from section 2, with sentence-specific A, B, and C costs.

$A_{f'}$ is implemented as a weighted FSA that scores a sequence such as $\pi = 1\ 4\ 2\ 5\ 6\ 3$. It can be easily constructed in practice as

$$A_{f'} \stackrel{\text{def}}{=} \text{domain}(f'^{*} \circ T) \tag{8}$$

where the unweighted FSA f' simply maps any integer $i \in \{1, 2, \ldots, N\}$ to f'_i (where f'_i is the i^{th} word of f', such as *pas/Neg:5*),[11] and domain$(\cdots)$ projects a weighted transducer onto its input tape to obtain a weighted automaton.[12]

From here on, we usually suppress the subscripts on $A_{f'}, B_{f'}$, and $C_{f'}$ for simplicity.

[8] According to the alignment model $Align$ as well as the additional permutation costs P.

[9] According to the translation model $Trans$.

[10] According to the language model $Lang$.

[11] Thus f'^{*} is a one-state machine that maps the integer sequence π to e', and $f'^{*} \circ T$ maps π nondeterministically to various e, with various scores.

[12] As the resulting $A_{f'}$ may contain duplicate arcs of different weights, or other arcs that cannot appear on any optimal path, it may be beneficial to reduce its size by an operation such as local determinization (Mohri, 1997).

3.6 Modeling translation costs

Suppose we wish to construct $Align$ as in section 3.3. A simple approach would be a single-parameter geometric distortion model. Here, $Align$ is a bigram model that considers each of the $N-1$ adjacent pairs in π, namely 1 4, 4 2, 2 5 ... in Fig. 2. Each bigram $\ell\ r$ incurs a cost of $\alpha|r - \ell - 1|$. As an automaton, $Align$ would be represented by a small $(N+1)$-state WFSA (Fig. 1), where the cost of $\ell\ r$ appears on the arc from state ℓ to state r.[13]

A bigram $Align$ model could also have more degrees of freedom, with distinct trainable costs for different $|r - \ell - 1|$ values. The model might be extended to also capture certain *trigrams* for common kinds of *local* rearrangements like 5 6 7 or 5 7 6. There are only a few such trigrams to consider at a given position, so the impact on WFSA size would be small. As long as the features abstract away from absolute word positions to consider (say) only differences, they can be shared across sentence lengths.

The B matrix penalizes pairwise word orders without respect to adjacency, so it can be used to penalize non-local reorderings that a small $Align$ automaton cannot. For example, in Fig. 2, π has the form $\ldots 5 \ldots 3 \ldots$, so it incurs the cost $b_{5,3}$, among others. This penalizes e' for putting the 5th word of f' (*pas/Neg:5*) somewhere before the 3rd word (*me/PRP:3*).

As for C, the $\binom{6}{3} = 20$ summands in Fig. 2 include $c_{2,6,3} > 0$, which penalizes e' for placing the 6th word of f' between the 2nd and 3rd words. Summing such C costs can penalize source neighbors 2 and 3 for being far apart in the target—a rough counterpart to a bigram A model, which can penalize target neighbors that were far apart in the source.

All of these costs might be made sensitive to POS tags, perhaps even particular words, and their context. We would do this by constructing $A_{f'}, B_{f'}$, and $C_{f'}$ from contextual *features* of word pairs and triples in f'. For example, given f', what features characterize words 2 and 5? Perhaps they are a determiner and noun separated only by adjectives, i.e., 2 3 4 5 matches Det Adj* N. Perhaps they also agree

[13] Note that the full A automaton can consider more than just the $Align$ cost. For instance, it might favor the bigram 2 5, since this represents a bigram $f'_2\ f'_5 =$ *ne pas* in e' that is a likely translation (according to $Trans$) of a common English phrase (according to $Lang$).

in grammatical gender. Now, the "before cost" $b_{2,5}$ can be defined as the sum of weights of all features that characterize the pair 2,5 in f'. The bigram cost of $2 \rightarrow 5$ in $Align$ can be described similarly, using a separate feature set.

An intriguing use for B and C is to make our P model sensitive to phrases, even to overlapping or syntactically nested phrases. The S process can mark these phrases in f' by inserting additional "phrase boundary" symbols. Thus noun phrase #4 should be delimited in f' by coindexed brackets $[_{NP4}$ and $]_{NP4}$. The B matrix should require these two matched brackets to stay in order, by assigning cost=∞ to reversing them.[14] Now C can be made to reward phrase coherence, by penalizing words[15] that were inside phrase #4 in f' if they move outside it in e', and vice-versa. Finally, $Align$, B, and C can be made to control the order of the left and right brackets for different phrases—thereby assessing costs for adjacent phrases $(Align)$, non-local phrase reordering (B), and changes to the nesting or overlapping of phrases (C).

4 Decoding

We now turn to the central algorithmic question of how to find a permutation π that minimizes the cost (1). There are $N!$ permutations—usually too many in practice to consider by brute force.

As noted earlier in section 2.3, the problem is computationally intractable even if one limits to simple costs, such as a pure bigram automaton model A (the TSP) or a pure beforeness model B (the LOP).

Germann et al. (2001) reduce such a problem in statistical MT to integer linear programming, noting its NP-completeness (Knight, 1999; Udupa and Maji, 2006). Other recent work has similarly reduced NLP problems to previously studied formal problems (Roth and Yih, 2005; Taskar et al., 2005; McDonald et al., 2005; Tromble and Eisner, 2006).

Those papers drew on exact solution methods. We instead draw on another rich vein of literature—heuristic strategies with some chance of search error.

[14]This may unfortunately reduce the number of feasible neighbors in section 4's local search methods. Hence one might soften this cost in early steps of local search and gradually raise it toward ∞ (somewhat like simulated annealing, section 7).

[15]David A. Smith has suggested to us that clitics or particles might be penalized less for moving out of their phrases.

4.1 Iterated local search

Local search starts with a guess, such as $\pi = 1\ 4\ 2\ 5\ 6\ 3$, and tries to improve it. A trivial strategy would obtain a new guess π' by swapping two adjacent elements of the sequence π. That is, for some i, we set $\pi'_i = \pi_{i+1}$ and $\pi'_{i+1} = \pi_i$. The cost of the revised π' can be found directly from equation (1). Alternatively, one may use a dynamic algorithm that obtains the cost of π' more rapidly by revising the cost of π.[16]

Given π, there are $N-1$ **neighboring** permutations π' that can be obtained in this way ($i = 1, 2, \ldots, N-1$). Local search would replace π with its *lowest-cost* neighbor, iterating until no further local improvement is possible.

Local search has previously been used for SMT decoding (Germann et al., 2001),[17] as well as the TSP (Lin and Kernighan, 1973) and the LOP (Congram, 2000). In general, each step considers some **neighborhood** of candidate solutions that are derived from the current candidate. It greedily moves to the lowest-cost neighbor of the current candidate, repeating until there is no change—that is, until π is locally optimal. This is a "hill-climbing" strategy. The challenge is to design effective neighborhoods that can be rapidly searched at each step for the lowest-cost neighbor.

To help avoid poor local optima, it is common to restart the local search at several random guesses. This produces several possibly distinct local optima. One may then choose the one with the lowest-cost.

Even with random restarts, the naïve swap neighborhood is likely to be slow and to get trapped in lo-

[16]To correct the B term rapidly, add $-b_{\pi_i,\pi_{i+1}} + b_{\pi_{i+1},\pi_i}$. Correcting the C term requires summing over other indices in the sequence π. To obtain a corrected A term in sublinear $o(N)$ time (i.e., without reading π' from scratch), it is necessary to maintain an array of all Viterbi forward and backward costs: for each state q and each $0 \leq i \leq N$, the cost of the best path from the initial state to q that accepts π_0^i, and the cost of the best path from q to a final state that accepts π_i^N.

[17]Germann et al. (2001) directly optimized the English output, e. They therefore needed local moves that would replace English *talking* with *talks* or insert English *about*. Our "lookahead" technique (section 3.5) allows us to optimize only the permuted French, e', and safely defer questions of lexical choice and fertility in e to a later stage. This is not to say that we fully escape the computational cost of dealing with such questions, since A effectively incorporates them via equation (6). However, some of them are genuinely deferred; others may be safely avoided by A* search (section 5.4), which starts with a smaller, approximate version of A.

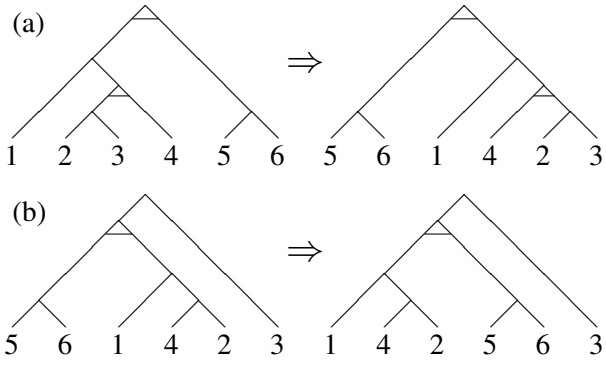

Figure 3: (a) A single twisted tree reordering the identity permutation. Swap nodes are indicated with a horizontal bar. The left shows the tree before swapping, and the right after. (b) A second tree, further reordering the result of (a). The new result could not have been reached from the identity permutation in a single step.

cal minima, since it only considers a tiny number of the possible permutations at each step. Other, more ambitious neighborhoods are possible: for example, π could generate $O(N^2)$ neighbors by moving some element π_i to some *arbitrary* new position in π. The trouble is that these larger neighborhoods generally take longer to search.

There is a wonderful way out: dynamic programming. We will present a family of very large-scale neighborhoods that have exponential size, yet can be searched in quadratic or cubic time, just like the neighborhood above.

4.2 Twisted-sequence neighborhood

Bompadre and Orlin (2005) review several **very large-scale neighborhoods** for the TSP. VLSNs are defined by exponential size but polynomial-time searchability. The lowest-cost neighbor can be found by dynamic programming—essentially, by finite-state or context-free parsing algorithms familiar to NLP.

Here we considerably generalize one of these algorithms to handle arbitrary A, B, and C costs (e.g., we obtain algorithms for the LOP and IBM Model 4). We then sketch how to speed it up using NLP techniques.

Given a current permutation sequence π, we define a large neighborhood $\mathcal{N}(\pi)$ that can be searched by dynamic programming. Each $\pi' \in \mathcal{N}(\pi)$ is defined by some binary tree with π at its fringe, with some of the internal nodes marked as "swap nodes." π' is found by swapping the two child subtrees of each swap node and then reading off the new fringe. An example appears in Fig. 3.

This "twisted-sequence neighborhood" $\mathcal{N}(\pi)$ was independently proposed for the TSP by Deĭnenko and Woeginger (2000). It is also exactly the set of sequences π' that could be related to π by an Inversion Transduction Grammar (Wu, 1997). But unlike these authors, when scoring these sequences, we will consider arbitrary A, B, and C costs.

To find the minimum-cost permutation in $\mathcal{N}(\pi)$, we construct a "parse chart" over π that builds up partial permutations. Each partial permutation has a cost. Consider in particular the contiguous subsequence $\pi_i^k = \pi_{i+1}\pi_{i+2}\ldots\pi_k$. Let $\mathcal{P}_{ikqt}$ denote the minimum total ABC cost of any twisted-sequence permutation of π_i^k that is accepted by some path in A that runs from state q to state t. If $k-i \geq 2$, $\mathcal{P}_{ikqt}$ may be defined recursively as the minimum of

$$\min_{j,s} \mathcal{P}_{ijqs} + \mathcal{P}_{jkst} + \gamma_{i,j,k} \quad \textit{(normal node)} \qquad (9)$$

$$\min_{j,s} \mathcal{P}_{ijst} + \mathcal{P}_{jkqs} + \bar{\gamma}_{i,j,k} \quad \textit{(swap node)} \qquad (10)$$

As the base case, when $k = i+1$, $\mathcal{P}_{ikqt}$ is the weight of the minimum-weight $q \rightarrow t$ arc in A that accepts the "word" π_i.[18]

These recurrences can be computed bottom-up as with CKY parsing (Aho and Ullman, 1972). Indeed, they may be regarded as "parsing" π using a binary CFG with rules of two types. Equation (9) combines a constituent π_i^j with "nonterminal label" $q \rightsquigarrow s$ with an adjacent constituent π_j^k with nonterminal label $s \rightsquigarrow t$, resulting in a constituent π_i^k with nonterminal label $q \rightsquigarrow t$. Equation (10) is similar but handles the case of swap nodes. The costs $\gamma_{i,j,k}$ and $\bar{\gamma}_{i,j,k}$ are analogous to grammar rule weights, although they are position-specific.

Fig. 4 shows the twisted tree from Fig. 3(a) as a parse tree with nonterminal labels of the form $q \rightsquigarrow t$. The best "parse" of the "sentence" π has cost $\min_t \mathcal{P}_{0NIt} + \textit{halting\_cost}(t)$, where I is the

[18] If A contains ϵ-arcs, the base case involves *subpaths* that accept π_i. One may avoid this by a preprocessing step that applies ϵ-closure to A.

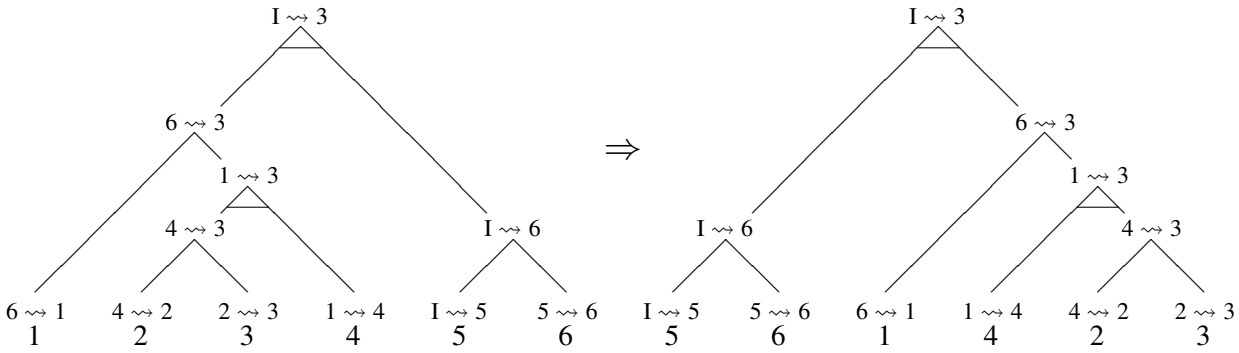

Figure 4: The tree from Fig. 3(a) annotated with "non-terminal" $q \rightsquigarrow t$ state pairs from a bigram model A, as in Fig. 1. The left side shows the tree before swapping, and the right side after. Notice that, at swap nodes, the right child's path ends at the state where the left child's path starts. The start nonterminal S can rewrite as I $\rightsquigarrow$ 3 because that is an accepting path (I is the initial state and 3 is final); the cost of this unary rewrite is set to be the cost of halting at state 3.

initial state of A and t ranges over the final states of A. The actual parse can be recovered by following backpointers in the usual way, and the optimal neighbor π' can be read off this parse.

Runtime depends on the topology of A. If A has Q states, then there are $O(N^3Q^3)$ ways to instantiate the variables i, j, k, q, s, t in equations (9)–(10). The runtime of the algorithm is therefore $O(N^3Q^3)$. In section 5, we will show how both factors can often be reduced.

The "grammar rule weights" $\gamma, \bar{\gamma}$ are

$$\gamma_{i,j,k} \stackrel{\text{def}}{=} \sum_{\ell \in \pi_i^j} \sum_{r \in \pi_j^k} \left(b_{\ell,r} + \sum_{o \in \pi_0^N \setminus \pi_i^k} c_{\ell,r,o} \right) \quad (11)$$

$$\bar{\gamma}_{i,j,k} \stackrel{\text{def}}{=} \sum_{\ell \in \pi_i^j} \sum_{r \in \pi_j^k} \left(b_{r,\ell} + \sum_{o \in \pi_0^N \setminus \pi_i^k} c_{r,\ell,o} \right) \quad (12)$$

Equation (11) is designed to fold some B and C costs into (9) as we assemble two constituents in "normal" order. Let p_1 and p_2 be permutations of π_i^j and π_j^k, respectively. Any permutation π containing p_1p_2 as a substring must place $\ell \in p_1, r \in p_2, o \notin p_1p_2$ either in the order ℓ, r, o (not necessarily adjacently) or in the order o, ℓ, r. Hence we add $b_{\ell,r}$ and $c_{\ell,r,o}$ (which equals $c_{o,\ell,r}$ by fiat) when we assemble p_1p_2. Equation (12) similarly folds B and C costs into (10) when we assemble p_2p_1.

Crucially for our runtime, the apparently expensive sums (11)–(12) can themselves be computed by dynamic programming. Using the recurrences in Fig. 5, *each* $\gamma_{i,j,k}$ or $\bar{\gamma}_{i,j,k}$ can be computed in $O(1)$ instead of $O(N^3)$ time, using previously computed instances of γ and $\bar{\gamma}$ that involve slightly narrower spans, as well as summations over C that were computed during $O(N^3)$ preprocessing.[19]

5 Speedups

As promised, we can improve the asymptotic analysis, and the runtime in practice, in several mutually compatible ways. This is necessary because an $O(N^3Q^3)$ runtime could be prohibitive. For instance, that runtime becomes $O(N^6)$ if $Q = N$, for example if A is a bigram model over the alphabet $1, 2, \ldots, N$.

5.1 Lopsided twisted-sequence neighborhoods

We can optionally reduce the N^3 factor by considering only right-branching trees. This corresponds to the "pyramidal" neighborhood used for the TSP by (Sarvanov and Doroshko, 1981).

Better yet, consider the neighborhood defined by "asymmetrically branching" trees: Fix a small constant h, and when assembling constituents in (9)–(10), only consider triples (i, j, k) such that $(j - i \leq h$ or $k - j \leq h)$. The recurrences in Fig. 5 continue to work and the runtime of the algorithm reduces to $O(N^2Q^3)$, plus the $O(N^3)$ preprocessing step on

[19] This preprocessing step may be faster than $O(N^3)$ for some restricted kinds of C matrices, or omitted if $C = 0$.

$$\gamma_{i,j,k} := \gamma_{i,j,k-1} + \gamma_{i+1,j,k} - \gamma_{i+1,j,k-1} + b_{\pi_{i+1},\pi_k} - \sum_{m \in \pi_{i+1}^{k-1}} c_{\pi_{i+1},m,\pi_k} + \sum_{o \in \pi_0^N \setminus \pi_i^k} c_{\pi_{i+1},\pi_k,o}$$

$$\bar{\gamma}_{i,j,k} := \bar{\gamma}_{i,j,k-1} + \bar{\gamma}_{i+1,j,k} - \bar{\gamma}_{i+1,j,k-1} + b_{\pi_k,\pi_{i+1}} - \sum_{m \in \pi_{i+1}^{k-1}} c_{\pi_k,m,\pi_{i+1}} + \sum_{o \in \pi_0^N \setminus \pi_i^k} c_{\pi_k,\pi_{i+1},o}$$

if $i < j < k$; otherwise $\gamma_{i,j,k} := 0, \bar{\gamma}_{i,j,k} := 0$

Figure 5: Efficient recurrence equations for the grammar weights in (11)–(12). Most of the necessary terms are added in from other γ and $\bar{\gamma}$ values; the subtractions are needed to correct for double-counting. The $\sum$ terms do not depend on j, and can therefore be reused. *Note:* This presentation of the equations relies for correctness on the fact that $c_{i,j,k} = c_{j,k,i} = c_{k,i,j}$.

C.[19] For a bigram model, this Q^3 ($= N^3$) factor is not tight—the total runtime is actually $O(N^4)$. This is fast in practice since there is no hidden grammar constant.

We may additionally allow all "anchored" triples of the form $(0, j, k)$ (or equivalently (i, j, N)).[20] That lets us cover the sequence π with several disjoint, asymmetrically branching trees that permute substrings of π, and then string those local trees together into one big left-branching (or right-branching) tree. It does not damage the $O(N^2Q^3)$ runtime. Unfortunately, here the Q^3 factor *is* tight for a bigram model.

Search space diameter. Such neighborhoods are still quite large. In the last case above, the complete search space of size $N!$ has a small diameter $\leq \log_2 N$ local steps (even for $h = 1$). Hence, when $N = 32$ words, any of the $32! > 10^{35}$ permutations is within 5 steps of any other (if not always 5 *greedy* steps). This is remarkably good for a neighborhood that can be searched in $O(N^2)$ time. The proof of small diameter is simple: Any permutation can be sorted into any desired order by $\log_2 N$ passes of median quicksort, and each pass is easily shown to correspond to one of our local permutations.

Obviously, our larger twisted-sequence neighborhood also gives a diameter $\leq \log_2 N$. It is also easy to show a *lower* bound on the diameter of our search spaces. Even the full twisted-sequence neighborhood has a size that grows only as $\Theta(5.83^N)$ (Zens and Ney, 2003). Since the full search space has size $N!$, it follows that for large N, the diameter must be *at least* $\log_{c \cdot 5.83^N} N! = \ln N! / \ln(c \cdot 5.83^N) \approx N \ln N / N \ln 5.83 \approx 0.39 \log_2 N$.

[20] This requires some new recurrences like those in Fig. 5.

5.2 Very large-scale finite-state neighborhoods

So-called "dynasearch" neighborhoods (Potts and van de Velde, 1995; Congram, 2000; Bompadre and Orlin, 2005) allow any collection of non-overlapping swaps in π to be carried out as a single move. The best move in this neighborhood may again be found by dynamic programming.[21]

Dynasearch neighborhoods are similar to the local reorderings of (Kanthak et al., 2005) and (Kumar and Byrne, 2005) (see section 6). However, local search can iterate them in order to explore permutations that are further afield.

While dynasearch neighborhoods are considerably smaller than the twisted-sequence neighborhoods above, they are still exponential in size. They can be obtained in our approach by restricting to very simple twisted trees, using only triples of the form $(j, j+1, j+2)$, $(0, j, j+1)$, and $(0, j, j+2)$. This makes them considerably faster to search, if perhaps less effective.

Dynasearch neighborhoods can also be characterized as *finite-state lattices* of permutations. This means that they and other finite-state neighborhoods, unlike the neighborhoods above, could still be efficiently searched for the best path (permutation) if we were to replace A by a weighted grammar such as a CFG or TAG. (In the SMT case, this

[21] Germann (2003) used a similar idea for SMT, but chose the collection with a greedy heuristic meant to rapidly approximate a sequence of single, individually greedy moves. By contrast, Dynasearch's global optimization confers a lookahead effect.

allows T to be a weighted synchronous grammar.)

Why is this possible and natural? Our algorithmic strategy for attaching A costs to the neighborhood is always to intersect that neighborhood (an unweighted language) with the cost model A (a weighted language). Context-free and tree-adjoining languages are closed under intersection with finite-state languages. Thus, we were able earlier to intersect a grammatical neighborhood (twisted sequences) with a finite-state cost model A by enriching its nonterminals. Conversely, we may intersect any finite-state neighborhood (dynasearch) with a grammatical cost model by lattice parsing.

5.3 Automaton topology

As section 9.1 will demonstrate, the practical runtime of the twisted-sequence neighborhoods is quite acceptable—because it is cubic or quadratic in N *with no hidden grammar constant*—when A is trivial (i.e., when $Q = 1$). When A is not trivial, we can often reduce the runtime by designing a model that reduces its number of states or arcs. Such a reduction amounts to tying parameters.

Notice that a full bigram model A as in Fig. 1 allows a separate cost for each possible bigram $\ell\ r$ that may appear in π. This may be more parameters than necessary. We could replace the $\ell \xrightarrow{r} r$ arc with a path $\ell \xrightarrow{\epsilon} X \xrightarrow{r} r$ of approximately the same weight. This replaces the $\ell\ r$ cost with a term that depends only on ℓ (and X) plus a term that depends only on r (and X). If we can replace many arcs in this way, while introducing only one or a few shared "backoff states" X, this may greatly reduce the number of arcs and the number of parameters.

If this modified bigram model has only a constant number of arcs accepting each "word" r, then our asymptotic runtime can be as small as $O(N^3)$. This runtime is obtained by using the lopsided twisted-sequence neighborhood of section 5.1 with $h = 1$ and without the additional "anchored" triples. There are then $O(N^2)$ choices of i, j, k in equations (9)–(10), and only $O(N)$ choices of q, s, t given i, j, k.

One way to get such a model is to replace arcs between pairs of indices not sufficiently close together. For example, replace the arc $\ell \xrightarrow{r} r$ if $|r - \ell| > 2$. All such bigrams, for example 2 5, might simply be given constant cost via a single backoff state X. Under this scheme, we would have at most five arcs accepting each index r, including the $X \xrightarrow{r} r$ arc.

5.4 Faster search of exponential neighborhoods

The considerable literature on speeding up parsing can be applied directly to our algorithm. For example, we can heuristically prune constituents (e.g., for a given i, j, discard all but the lowest-cost constituents $\mathcal{P}_{ijqs}$) or use best-first search (Caraballo and Charniak, 1998). While these methods do risk missing the best parse, we are in the midst of a non-optimal greedy local search anyway. Missing some parses only means that we are not searching *quite* as large a neighborhood of permutations at each step.

Among "safe" speedups that are guaranteed to find the best parse of the current permutation π, probably the simplest is to discard subpaths that cost more than π. This still keeps π itself (which is in the neighborhood) as well as any π' that improve on π.

A better "safe" speedup is A* parsing (Klein and Manning, 2003), which assembles constituents in a prioritized order and can stop as soon as a full parse is found. Defining the priority of a constituent $\mathcal{P}_{ijqs}$ requires an admissible (i.e., optimistic) bound on its "outside" cost. Our current implementation computes such bounds by replacing A with a simpler 0th-order (unigram, memoryless) WFSA. In this A', the cost of a self-loop labeled with the "word" r is the best cost of any r-labeled arc in A. Since A' has $Q = 1$, bounds on all outside costs can be computed in $O(N^3)$ or $O(N^2)$, depending on the choice of neighborhood.

More generally, given any partition of the states of A into near-equivalence classes, one can construct a new, coarser, FSA, A', whose states correspond to these classes (cf. Stolcke and Omohundro (1993)). The unigram model above is the limiting case of a single, all-encompassing equivalence class. The cost of the r-labeled arc in A' from class $\bar{q}$ to class $\bar{s}$ is defined as the best cost of any r-labeled arc in A from any $q \in \bar{q}$ to any $s \in \bar{s}$. Now A' provides an admissible estimate of A, and a succession of automata $A, A', A'', \ldots$ can be used for an exact coarse-to-fine search (Geman and Kochanek, 2001).

An independent type of A* heuristic would similarly coarsen the beam-search automaton in section 6 below, then obtain Viterbi forward/backward estimates. Here, coarsening relaxes the requirement that π be a permutation. The priority of a given con-

stituent $\mathcal{P}_{ijqs}$ can use the tighter of the two heuristics.

Finally, we remark that if some substrings of π appear unchanged in its optimal neighbor π', then all of the parse chart's entries for those substrings may be reused at the next step of local search, where we seek optimal neighbors of π'.

6 Other Decoding Options

Much past work in statistical MT has recognized and faced the problem of searching for optimal permutations. Several authors, particularly those working in finite-state frameworks, explicitly model permutation as a step in a composite translation process.

However, past decoders have had to consider the full composite process all at once. Our reduction in section 3.3 shows that it is possible to treat permutation search in isolation. This made it easier for us to adapt existing local search methods for permutation problems like the TSP and the LOP in section 4.

Most past decoders have not used local search at all. Rather, they provide alternative strategies for our problem. We now outline how they would be applied directly to permutation search.

The usual starting point is Viterbi decoding or stack decoding: one seeks the best path through a very large acyclic WFSA, in which each path assembles a different permutation sequence such as $\pi = 1\ 4\ 2\ 5\ 6\ 3$. Each state represents a prefix of a possible permutation, such as $1\ 4\ 2$, from which each directed edge leads to an extended prefix such as $1\ 4\ 2\ 5$ (but not $1\ 4\ 2\ 4$). The edges are weighted so as to accumulate the total permutation cost over the entire path.

As this WFSA is enormous, with $O(N!)$ states, several tricks are commonly used. Following Held and Karp (1962), Knight and Al-Onaizan (1998) collapses states that represent the same subset (e.g., $1\ 4\ 2$ and $2\ 4\ 1$) into a single state $\{1, 2, 4\}$ (Fig. 6). This simplification to $O(2^N)$ states nonetheless admits our full "ABC" cost model. The arcs of the permutation WFST can incorporate the B and C costs. For example, the arc from $\{1, 2, 4\}$ to $\{1, 2, 4, 5\}$ in Fig. 6 would have weight

$$\sum_{r \in \{3,6\}} \left(b_{5,r} + \sum_{\ell \in \{1,2,4\}} c_{\ell,5,r} \right)$$

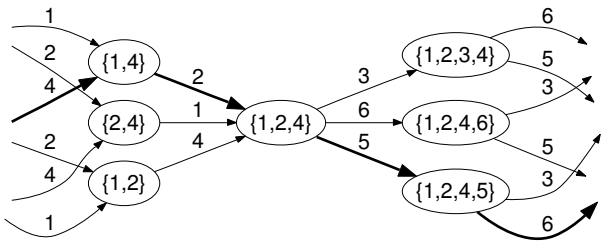

Figure 6: A small portion of the permutation automaton for the example in Fig. 2. Bold arcs are along the desired path $1\ 4\ 2\ 5\ 6\ 3$.

Intersection with $A_{f'}$ from equation (8) incorporates the costs from T, and results in an automaton of size at most $2^N Q$.

At any rate, $O(2^N)$ is still very large for, say, $N = 30$. One approach is to use a pruned beam search, constructing states on demand as they are actually explored (Koehn, 2004; Kanthak et al., 2005).

An alternative is to systematically limit the set of allowed permutations. Kumar and Byrne (2005) describe restriction to movement within a three-phrase window. Kanthak et al. (2005) investigate a variety of subsets of the full permutation set, using similar windows, as well as the set of permutations allowed by Inversion Transduction Grammar (ITG) (Wu, 1997). We will consider the adequacy of this particular subset in section 9.2.

As a hybrid technique, note that permutation decoders of any sort, including beam-search decoders with a very narrow or greedy beam, could be used to produce an initial permutation that is then improved by local search. Indeed, for many problems, (randomized) greedy initialization of local search is a highly effective technique, known as GRASP (Feo and Resende, 1995). Another hybrid possibility is to run our local search algorithm but fall back to a beam-search decoder for all narrow spans (say, all $\mathcal{P}_{ijqs}$ with $j - i \leq 8$).

7 Stochastic Methods

We can interpret our cost model (1) in probabilistic terms via a familiar transformation:

$$p(\pi|f') = \frac{w_{f'}(\pi)}{\sum_{\pi'} w_{f'}(\pi')}. \qquad (13)$$

where

$$w_{f'}(\pi) \stackrel{\text{def}}{=} \exp - \big(A_{f'}(\pi) + B_{f'}(\pi) + C_{f'}(\pi)\big) \tag{14}$$

It is intractable to find the sum of the $N!$ terms in the denominator of (13) (the **partition function**). However, we *can* efficiently compute the **neighborhood partition function**

$$\sum_{\pi' \in \mathcal{N}(\pi)} w_{f'}(\pi') \tag{15}$$

that sums over all permutations in an exponentially large neighborhood of π. Simply convert the "Viterbi algorithm" parser of section 4.2 to the "inside algorithm" version that sums over subtree probabilities instead of minimizing over subtree costs. The inside algorithm has the same runtime complexity, although it does not allow speedups like A*.

The inside algorithm also produces a parse forest from which we can sample permutations of $\mathcal{N}(\pi)$ using the usual top-down parse sampling algorithm (Goodman, 1998, pp. 146–147). $\pi' \in \mathcal{N}(\pi)$ is selected with probability proportional to $w_{f'}(\pi')$ (i.e., $w_{f'}(\pi)$ divided by equation (15)).

The only wrinkle is that these methods sum and sample over the set of *twisted trees*, not the set of permutations. This will count a permutation multiple times if it is defined by multiple different trees. For example, *every* tree on π containing no swap nodes will define π itself. The solution is to eliminate this spurious ambiguity in the parser, at a small constant-time overhead, by recovering only the normal-form trees (Eisner, 1996; Zens and Ney, 2003). Each permutation in the neighborhood is characterized by only one normal-form tree.

Now, to sample from the entire space of permutations, rather than just the subset in one of our neighborhoods, we can use the Metropolis-Hastings algorithm. This means taking a random walk in permutation space, where at each step we attempt to move to a randomly sampled neighbor π' of the current permutation π (instead of the lowest-cost neighbor). An attempted move is randomly rejected, however, with a probability that depends on the neighborhood partition functions of both π and the sampled π'.

To look for the *optimal* permutation, we can replace the hill-climbing local search of section 4 with a simulated annealing search. In other words, we take a random walk as above, but redefine equation (14) to divide the total cost by a *temperature* before exponentiating it. The temperature is gradually lowered toward zero as the random walk progresses.

8 Training

When facing a problem such as machine translation, we typically need to learn our costs from training data. In the simplest case, we are given many instances of f' and, for each, the corresponding gold-standard permutation π^*. The goal is to learn feature weights (see section 3.6) such that π^* will be favored by the cost functions A (or T), B, and C that are derived from f' and the feature weights.

One approach would be to learn feature weights that maximize $p(\pi^* \mid f')$ according to equation (13). This is a standard case of training a log-linear model.[22] The trouble, again, is that we cannot compute the partition function, which we would need for direct optimization. Nor can we compute the expected feature counts under the distribution (13), which we would need for the improved iterative scaling algorithm (Della Pietra et al., 1997). However, we could compute the expectations approximately by sampling permutations as in Section 7.

A second strategy is to approximate the above training criterion and use the neighborhood partition function (15) in place of the true partition function. That is, we would try to maximize the conditional likelihood of π^* given its own neighborhood $\mathcal{N}(\pi^*)$ (perhaps even pruning the neighborhood for speed). For a sufficiently large neighborhood, this contrastive estimation strategy (Smith and Eisner, 2005) may be a good approximation.[23]

A third strategy is to train discriminatively using, for example, the perceptron algorithm (Freund and Schapire, 1998). We can use our iterated local search with the current model parameters, arriving

[22] The training is supervised if we only use B and C costs, but unsupervised in general. The reason is that even though π^* is observed, the path through A that generated π^* may not be uniquely determined by π^*, so we do not observe the counts of transitions in A.

[23] Smith and Eisner themselves used neighborhoods that were considerably smaller. Perhaps their accuracy would have benefited from using our full twisted-sequence neighborhood. It is hard to know, since their choice of neighborhood was designed not only to get a fast approximation to the true partition function, but also to bias their unsupervised learner toward learning "syntactic" hidden states as it learned the cost model A.

at a *local* optimum. We update the parameters if the discovered permutation π differs from the desired π^* and the model actually incorrectly prefers π to π^*.[24]

Separately, one might wish to learn to search effectively. The search procedure is trainable even for a fixed model. We could adapt the parameters of the model to the search procedure (Daumé III and Marcu, 2005), adapt the objective function consulted by local search (Boyan and Moore, 2000), or use reinforcement learning to learn separate parameters for search (Nareyek, 2003). Reinforcement learning is particularly attractive because it could also learn to choose a type of neighborhood for the next search step, based on the search history.

9 Experiments

9.1 Speed and accuracy

We have done some preliminary experiments establishing that our algorithm, even without pruning or other speedups, exceeds beam search's accuracy with comparable runtime. Beam search tends to be faster for short sentences, where iterated local search is overkill, but suffers more on longer sentences. We could hybridize as suggested in section 6.

Also, a version of our code that considers only B costs, again without any pruning (section 5.4), is able to solve standard LOP instances (Schiavinotto and Stützle, 2004) on 60-word sequences in 6–7 iterations that run as fast as 1.8 milliseconds/iteration in the twisted-sequence neighborhood, or 0.4 ms/iteration in our largest quadratic-time neighborhood of section 5.1 ($h = 1$). Using several randomized restarts does help avoid search error (local optima), especially in the latter case.

9.2 Length of local search trajectory

The point of using large neighborhoods is to be able to find the correct answer within a few steps of local search. This reduces the nearsightedness of local search and may improve its hillclimbing speed. As noted before, even our modest neighborhood of section 5.1 produces a local search graph of diameter $\leq \log N$.

[24] We do not necessarily want to carry out the update if the model *already* prefers π^* but simply cannot reach it by local search. Such updates can cause the parameters to diverge. One fallback strategy is to perform the update using not π^* but rather the lowest-loss candidate in $\mathcal{N}(\pi)$, under some loss function.

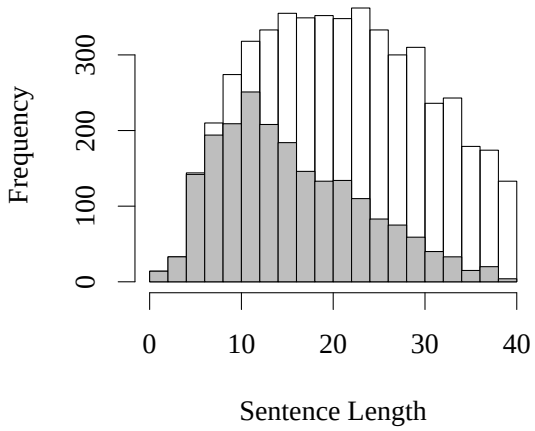

Figure 7: Number of German-English sentences (outlined), and the portion for which the twisted-sequence neighborhood includes the gold-standard alignment (shaded), grouped by length.

An important empirical question is how often the correct ("oracle") permutation is available within a *single* step of our stochastic local search. In this case, a greedy local search decoder is actually optimal, provided that the cost model is adequate to identify this correct permutation.

In particular, Dekai Wu has proposed that if we start with the identity permutation $1\ 2 \ldots N$, then our neighborhood in section 4.2—which corresponds to the permutations admitted by his Inversion Transduction Grammar (Wu, 1997)—will generally contain a reordering that is linguistically adequate for translation.

We conjectured that our neighborhoods would indeed often be large enough in this case for a single step of section 4.2 to consider the oracle permutation. This motivates our use of large neighborhoods.

However, we also conjectured that one step would not *always* suffice (at least not to find the oracle permutation), motivating our use of local search. For example, Fig. 2 shows a permutation that cannot be achieved in one step.

These conjectures were borne out on the first 5000 German-English sentence pairs from the Europarl corpus. The results appear in Fig. 7. The twisted-sequence neighborhood contains the gold-standard permutation for 41.7% of all sentences, including 63.6% of those 2382 containing ≤ 20 words.

Perhaps more important, though, is that one step in this neighborhood is only adequate for 21.9% of examples *longer* than twenty words, which constitute more than half of the corpus. It is especially these longer sentences for which iterated local search compares favorably to beam search, since 2^{30} is a million times greater than 2^{10}.

A caveat is that these measurements were made using machine-generated alignments. Performance might be better with human alignments. Further, there might be a good translation in the neighborhood, even if it differs from the one given in the bitext.

10 Conclusions

We have presented new local search algorithms that seek optimal permutations. The power of these algorithms comes from their use of dynamic programming to search exponentially large neighborhoods. While that technique was previously known, in particular for the Traveling Salesperson Problem, our contributions include

- extensions to further cost functions, including B costs (for the LOP) and novel A and C costs,
- extensions to new neighborhoods,
- bounds on the diameter of the search space,
- the use of parsing efficiency techniques such as pruning, A* search, and coarse-to-fine search,
- a technique for MCMC sampling and simulated annealing,
- some discussion of learning the cost functions.

We also discussed statistical machine translation at length. To show that permutation search can be regarded as the central problem of SMT (cf. footnote 17), we formulated an attractive "$S \circ P \circ T$" framework for describing SMT models, with permutation as the only non-finite-state step. We showed that this framework was expressive (among other things, it covers IBM Model 4) and that it can be decoded by combining permutation search with simple finite-state operations.

We hope that others will find the local search paradigm to be as thought-provoking as we have. In future work, we plan to evaluate the methods proposed here by further experiments.

References

A. V. Aho and J. D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall.

Regina Barzilay and Lillian Lee. 2004. Catching the drift: Probabilistic content models, with applications to generation and summarization. In *Proceedings of HLT-NAACL*, pages 113–120.

Agustin Bompadre and James B. Orlin. 2005. Using grammars to generate very large scale neighborhoods for the traveling salesman problem and other sequencing problems. In M. Jünger and V. Kaibel, editors, *IPCO*, pages 437–451. Springer-Verlag.

Justin A. Boyan and Andrew W. Moore. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, November.

Peter F. Brown, Stephen A. Della Petra, Vincent J. Della Pietra, John D. Lafferty, and Robert L. Mercer. 1992. Analysis, statistical transfer, and synthesis in machine translation. In *Proceedings of the Fourth International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 83–100.

Peter F. Brown, Stephan A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, June.

Sharon Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Richard K. Congram. 2000. *Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation*. Ph.D. thesis, University of Southampton, UK.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany.

S. Della Pietra, V. Della Pietra, and J. Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4).

V.G. Deĭnenko and G.J. Woeginger. 2000. A study of exponential neighborhoods for the traveling salesman problem and for the quadratic assignment problem. *Mathematical Programming, Ser. A*, 78:519–542.

Peter Eades and Sue Whitesides. 1994. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, September.

Jason Eisner. 1996. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 79–86, Santa Cruz, June.

Jason Eisner. 2000. Easy and hard constraint ranking in Optimality Theory: Algorithms and complexity. In *Finite-State Phonology: Proceedings of the 5th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 22–33.

T. A. Feo and M. G. C. Resende. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Yoav Freund and Robert E. Schapire. 1998. Large margin classification using the perceptron algorithm. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 209–217, New York, NY, USA. ACM Press.

Stuart Geman and Kevin Kochanek. 2001. Dynamic programming and the representation of error-correcting codes. *IEEE Transactions on Information Theory*, 47(2):549–568.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 228–235, Morristown, NJ, USA. Association for Computational Linguistics.

Ulrich Germann. 2003. Greedy decoding for statistical machine translation in almost linear time. In *Proceedings of HLT-NAACL*.

F. Glover, T. Klastorin, and D. Klingman. 1974. Optimal weighted ancestry relationships. *Management Science*, 20:B1190–B1193.

Joshua Goodman. 1998. *Parsing inside-out*. Ph.D. thesis, Harvard University, May.

Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1984. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, Nov.–Dec.

M. Held and R. M. Karp. 1962. A dynamic programming approach to sequencing problems. *J. SIAM*, 10(1):196–210.

T.R. Jordan and A. Monteiro. 2003. Generating anagrams from multiple core strings employing user-defined vocabularies and orthographic parameters. *Behavior Research Methods, Instruments, and Computers*, 35(1):129–135, February.

Stephan Kanthak, David Vilar, Evgeny Matusov, Richard Zens, and Hermann Ney. 2005. Novel reordering approaches in phrase-based statistical machine translation. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pages 167–174, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In Marti Hearst and Mari Ostendorf, editors, *HLT-NAACL 2003: Main Proceedings*, pages 119–126, Edmonton, Alberta, Canada, May 27 - June 1. Association for Computational Linguistics.

Kevin Knight and Yaser Al-Onaizan. 1998. Translation with finite-state devices. In *Proceedings of the 3rd AMTA Conference*, pages 421–437, London. Springer-Verlag.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In Marti Hearst and Mari Ostendorf, editors, *HLT-NAACL 2003: Main Proceedings*, pages 127–133, Edmonton, Alberta, Canada, May 27 - June 1. Association for Computational Linguistics.

Philipp Koehn. 2004. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In Robert E. Frederking and Kathryn Taylor, editors, *AMTA*, volume 3265 of *Lecture Notes in Computer Science*, pages 115–124. Springer.

Shankar Kumar and William Byrne. 2005. Local phrase reordering models for statistical machine translation. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 161–168, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Mirella Lapata. 2003. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of ACL*, pages 545–552.

Gregor Leusch, Nicola Ueffing, and Hermann Ney. 2006. CDER: Efficient MT evaluation using block movements. In *Proceedings of EACL*, pages 241–248, Trento, Italy, April.

S. Lin and B. W. Kernighan. 1973. An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, 21(2):498–516, March–April.

Inderjeet Mani, Barry Schiffman, and Jianping Zhang. 2003. Inferring temporal ordering of events in news. In *Proceedings of HLT-NAACL (Companion Volume)*, pages 55–57, Edmonton, May.

Gideon Mann. 2006. *Multi-Document Statistical Fact Extraction and Fusion*. Ph.D. thesis, Johns Hopkins University.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Sounaka Mishra and Kripasindhu Sikdar. 2004. On approximability of linear ordering and related np-optimization problems on graphs. *Discrete Applied Mathematics*, 136(2–3):249–269, February.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2).

Michael S. Morton. 1987. Recursion + data structures = anagrams. *Byte Magazine*, 12(12):325–332, November.

Alexander Nareyek. 2003. Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende and J. P. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, pages 523–544. Kluwer Academic Publishers.

Sonja Nießen and Hermann Ney. 2001. Toward hierarchical models for statistical machine translation of inflected languages. In *Proceedings of the Workshop on Data-Driven Methods in Machine Translation*, pages 1–8, Morristown, NJ, USA. Association for Computational Linguistics.

P. Orponen and H. Mannila. 1987. On approximation preserving reductions: Complete problems and robust measures. Technical Report C-1987-28, Department of Computer Science, University of Helsinki.

C.N. Potts and S.L. van de Velde. 1995. Dynasearch—iterative local improvement by dynamic programming. part i. the traveling salesman problem. Technical report, University of Twente, The Netherlands.

Dan Roth and Wen-tau Yih. 2005. Integer linear programming inference for conditional random fields. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 737–744.

V.I. Sarvanov and N.N. Doroshko. 1981. The approximate solution of the travelling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time. *Software: Algorithms and Programs*, 31:11–13. Mathematical Institute of the Belorussian Academy of Sciences, Minsk. In Russian.

Tommaso Schiavinotto and Thomas Stützle. 2004. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modeling and Algorithms*.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 354–362, Ann Arbor, Michigan, June.

Andreas Stolcke and Stephen Omohundro. 1993. Hidden Markov Model induction by bayesian model merging. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18. Morgan Kaufmann, San Mateo, CA.

Ben Taskar, Simon Lacoste-Julien, and Dan Klein. 2005. A discriminative matching approach to word alignment. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 73–80, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Roy W. Tromble and Jason Eisner. 2006. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proceedings of HLT-NAACL*, pages 423–430.

Raghavendra Udupa and Hemanta K. Maji. 2006. Computational complexity of statistical machine translation. In *Proceedings of EACL*, Trento, Italy, April.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, September.

Richard Zens and Hermann Ney. 2003. A comparative study on reordering constraints in statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 144–151, Sapporo, Japan, July.

Appendix: Non-Cyclic Betweenness Costs

Section 2.2 restricted the array C to have a certain cyclic symmetry property: $c_{\ell,m,r} = c_{m,r,\ell} = c_{r,\ell,m}$. That restriction on C is natural in TSP-like problems where π is intended to be evaluated cyclically. For example, in TSP-like problems where π represents a closed tour on cities, the starting point of π is arbitrary.

However, for some problems, such as machine translation, it would be more useful to have three-way costs $\hat{C}$ that are not constrained in this way.

For example, if ℓ and r were adjacent in the source sequence, we might wish to encourage them to be close to each other in the target sequence π (read non-cyclically). This means setting large values for $\hat{c}_{\ell,m,r}$ and $\hat{c}_{r,m,\ell}$ only, for all m.

Sometimes, other costs (such as $b_{r,\ell} = \infty$) may ensure that ℓ precedes r. In that case, $c_{\ell,m,r}$ fires iff $\ell < m < r$, so it behaves just like $\hat{c}_{\ell,m,r}$.

More generally, what if we wish to augment equation (1) with a fourth term, $\hat{C}(\pi) \stackrel{\text{def}}{=} \sum_{i<j<k} c_{\pi_i,\pi_j,\pi_k}$, that acts just like C but without the restriction that $\hat{c}_{\ell,m,r} = \hat{c}_{m,r,\ell} = \hat{c}_{r,\ell,m}$?

This case can be reduced to the original equation (1), if we are willing to add extra symbols to our set $\{1, 2, \ldots, N\}$.

We may assume without loss of generality that $\hat{C}$ is everywhere ≥ 0. If it is not, then for each ordered triple of distinct integers ℓ, m, r, in parallel, subtract $\min(\hat{c}_{\ell,m,r}, \hat{c}_{m,r,\ell}, \hat{c}_{r,\ell,m})$ from $\hat{c}_{\ell,m,r}$ and add it to $c_{\ell,m,r}$. This does not change the total cost function $A(\pi) + B(\pi) + C(\pi) + \hat{C}(\pi)$.

We can now eliminate $\hat{C}$ step-by-step as follows. If $\hat{C}$ is not yet 0 everywhere, choose an ℓ, r pair such that $(\exists m)\hat{c}_{\ell,m,r} \neq 0$. To handle this ℓ, r pair, increase N by 1, and modify the cost functions A, B, C, and $\hat{C}$ to consider the new last element N as follows:

- At every state of A, add a self-loop that reads N with cost 0.
- Set $b_{N,r} = \sum_m \hat{c}_{\ell,m,r}$. Set to 0 all other entries of B whose row or column is indexed by N.
- For each $1 \leq m \leq N$, set $c_{\ell,m,N} = c_{m,N,\ell} = c_{N,\ell,m} = \hat{c}_{\ell,m,r}$; then set $\hat{c}_{\ell,m,r} = 0$. Set to 0 all other entries of C that are indexed in part by N.
- Set to 0 all entries of $\hat{C}$ that are indexed in part by N.

The above transformation increased N by 1 while zeroing some $\hat{C}$ entries and adding some new B and C entries involving N. This did not change the problem, in the following sense. Suppose π was some permutation over $\{1, 2, \ldots, N-1\}$. Consider all "refinements" π' that are obtained by inserting the extra symbol N somewhere into π. We claim that the minimum-cost refinement π' (under the new cost functions) has the same cost as π (under the old cost functions):

- If ℓ precedes r in the sequence π, then the minimum-cost position for N is just after r. At that position each old cost $\hat{c}_{\ell,m,r}$, for m such that $\ell < m < r$, has been set to 0 but exactly replaced by a new, equal cost $c_{\ell,m,N}$. Placing N any later would incur extra (positive) costs of the form $c_{\ell,o,N}$. Placing N any earlier might save on some of the (positive) costs $c_{\ell,m,N}$, but would incur a cost $b_{N,r}$ that was at least as great as the total savings.
- If ℓ follows r in the sequence π, then zeroing $\hat{c}_{\ell,m,r}$ was harmless because no costs of this form would have been incurred by π. Creating (positive) costs involving N was also harmless: the minimum-cost position for N is just after ℓ, at which position none of the costs involving N are incurred.

Repeating this transformation, we eventually eliminate $\hat{C}$ (setting it stepwise to 0) while increasing N to at most N^2. By induction on the claim above, every original permutation π has some refinement π' whose cost has been preserved, and no refinement whose cost has improved. It follows that if π was optimal under the old problem (with $\hat{C}$), then one of its refinements π' is optimal under the new problem (without $\hat{C}$). We may therefore seek π' under the new problem and delete its extra symbols to obtain π.

Of course, the increased size of the problem (increased N) will slow down each local search step, and may also make it harder to find the correct permutation by local search. It is therefore wise to keep $\hat{C}$ small in the sense that few new symbols need to be added. The machine translation example mentioned at the start of this appendix used $\hat{c}_{\ell,m,r}$ only for ℓ, r that were adjacent in the source string, which requires adding only $N - 1$ new symbols.

The local search may be sensitive to the initial placement of the new symbols. A greedy placement would introduce each N immediately after the rightmost of the ℓ, r with which it is associated. However, it may be helpful to randomize this initial placement and try several random restarts.