SEARCH AND LEARNING FOR THE LINEAR ORDERING PROBLEM WITH AN APPLICATION TO MACHINE TRANSLATION

by

Roy Wesley Tromble

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

April, 2009

Abstract

This dissertation is about ordering. The problem of arranging a set of n items in a desired order is quite common, as well as fundamental to computer science. Sorting is one instance, as is the Traveling Salesman Problem. Each problem instance can be thought of as optimization of a function that applies to the set of permutations.

The dissertation treats word reordering for machine translation as another instance of a combinatorial optimization problem. The approach introduced is to combine three different functions of permutations. The first function is based on finite-state automata, the second is an instance of the Linear Ordering Problem, and the third is an entirely new permutation problem related to the LOP.

The Linear Ordering Problem has the most attractive computational properties of the three, all of which are NP-hard optimization problems. The dissertation expends significant effort developing neighborhoods for local search on the LOP, and uses grammars and other tools from natural language parsing to introduce several new results, including a state-of-the-art local search procedure.

Combinatorial optimization problems such as the TSP or the LOP are usually given the function over permutations. In the machine translation setting, the weights are not given, only words. The dissertation applies machine learning techniques to derive a LOP from each given sentence using a corpus of sentences and their translations for training. It proposes a set of features for such learning and argues their propriety for translation based on an analogy to dependency parsing. It adapts a number of parameter optimization procedures to the novel setting of the LOP.

The signature result of the thesis is the application of a machine learned set of linear ordering problems to machine translation. Using reorderings found by search as a preprocessing step significantly improves translation of German to English, and significantly more than the lexicalized reordering model that is the default of the translation system.

In addition, the dissertation provides a number of new theoretical results, and lays out an ambitious program for potential future research. Both the reordering model and the optimization techniques have broad applicability, and the availability of machine learning makes even new problems without obvious structure approachable.

Thesis committee:
Advisor: Jason Eisner[1]
Reader: Sanjeev Khudanpur[2]
Committee member: Chris Callison-Burch[3]

[1] Associate Professor, Computer Science, Johns Hopkins University
[2] Associate Professor, Electrical and Computer Engineering, Johns Hopkins University
[3] Assistant Research Professor, Computer Science, Johns Hopkins University

For Nicole

Acknowledgements

The work, and even the text, of this dissertation would not be what it is without the steady influence of my advisor, Jason Eisner. He had a hand in generating many of the ideas that appear here, helped shape still more, and provided extensive guidance to improve their presentation, both here and elsewhere. His intellectual curiousity is an inspiration, and his willingness to embrace ideas beyond his current knowledge was essential to this dissertation's foundation, as well as an encouragement of my broad interests.

Sanjeev Khudanpur and Chris Callison-Burch rounded out my thesis committee. Both provided valuable insight and feedback while I prepared the dissertation as well as during its defense. In spite of the contributions of my committee, I of course take full responsibility for any mistakes or omissions you may find here.

I spent most of my tenure at Johns Hopkins in NEB 332, which I shared over the years with David Smith, Noah Smith, Paul Ruhlen, Markus Dreyer, and John Blatz. These and numerous other colleagues helped make the years of hard work feel worthwhile.

My first three years in the Ph.D. program were generously supported by a National Defense Science and Engineering Graduate Fellowship, through the American Society for Engineering Education. My fellowship was sponsored by the Office of the Deputy Under Secretary of Defense for Science and Technology and the Army Research Office. The remainder of my time in the Ph.D. program was supported by the National Science Foundation under grants No. 0347822 and 0313193.

Finally, I must thank my family for their love and support. My parents instilled in me a love of learning that will stay with me throughout my life. My siblings remain my closest friends, and have always been willing to commiserate during their own academic pursuits. My wife, Nicole, has been with me every step of this journey. She encouraged me to apply to graduate schools, she accompanied me in our move to the city from small town Idaho, and she has kept me focused on my goal throughout, while providing constant companionship and plenty of other things to talk about. Last of all, my new daughter Rachel has been a welcome distraction during these past ten months, and I look forward to sharing parts of this experience with her some day.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ordering problems are pervasive in computer science and outside: Given a collection of items, arrange them in a sequence according to some rules. When the rules consist of a total order, the problem is simply sorting, one of the fundamental studies of computer science. Definition 2.5 in Section 2.11.1 will formally define a total order. The essential property of a total order relation is that it is transitive—if a belongs before b and b belongs before c, then a necessarily also belongs before c.

Transitivity doesn't always hold in practice. Among items with pairwise precedence preferences derived from measurements of properties of the items, cycles exist. One formulation of ordering rules under this more general setting leads to the Linear Ordering Problem: Find the sequential arrangement that maximizes the sum of the satisfied pairwise precedences. It will be given formally in Definition 2.2 in Section 2.1. It is an intractable combinatorial optimization problem. The corresponding decision problem—is there a permutation of the items with score at least K?—is known to be NP-complete.

A different criterion for ordering leads to the Traveling Salesman Problem, another NP-complete combinatorial optimization problem. The TSP can be thought of as concerning itself with the *local* structure of the permutation, while the LOP is concerned with *global* structure. In addition to the LOP, this dissertation considers a generalization of the TSP expressed using finite-state automata.

These and one more completely new function of permutations are adopted to address the problem of word reordering for machine translation. Machine translation is another difficult problem, several formulations of which are known to be NP-hard as well. The approach taken here is to assume that translation would be easy if it was always monotonic. This is probably not true, but it is certainly true that it would be less difficult. The assumption allows reordering to become the primary focus of the translation problem, and the formulation uses a trick involving lookahead through a finite-state transducer to convert the entire translation problem into a permutation problem.

More generally, this dissertation concerns itself with three areas of study, and makes novel connections among the three. The first is local search for the Linear Ordering Problem, the second is reordering for Machine Translation, and the third is machine learning. The dissertation applies techniques from natural language parsing to the Linear Ordering Problem, applies the Linear Ordering Problem to reordering for Machine Translation, and finally uses machine learning to improve reordering models.

1.1 The Linear Ordering Problem

The Linear Ordering Problem arises in a surprising number of applications in fields including social choice theory, economics, archaeology, and information retrieval, among others. Section 2.1.1 will give a more complete enumeration, and describe the details of each. Here, it is enough to observe that this problem, in spite of its difficulty, is of significant practical interest. Many exact solution methods exist for the Linear Ordering Problem, but all necessarily start to fail when the problems become too large. Search is therefore a common approach to the LOP.

The work of this thesis derived from a novel dynamic programming algorithm for efficiently computing costs of permutations under the Linear Ordering Problem. The idea was to use shared substructure to convert the scores for pairwise *item* orderings, given in the LOP, into scores for pairs of adjacent *blocks* of items. Computing the score for arranging a single pair of adjacent blocks would require $\Theta(n^2)$ time, for some blocks taken from a collection of size n, but the dynamic program computes the scores for all $\Theta(n^3)$ such pairs of blocks in only $\Theta(n^3)$ time—constant time per pair of adjacent blocks. The dynamic program will appear in detail in Figure 2.26 of Section 2.8.2.

Another essential property of the dynamic program is that the score for arranging a particular pair of blocks does not depend at all on the permutations of items *within* those blocks. As a result, it remains a $\Theta(n^3)$ operation to compute the best score considering all possible *nested* arrangements of pairs of adjacent blocks. This makes it convenient to use a grammar to describe the process.

The dynamic program works given an existing sequential arrangement—a permutation—of the collection of items. The resulting set of block orderings for which computation is efficient thus depends on the starting permutation. The dynamic program induces a *neighborhood* about the starting permutation—a set of permutations linked to that starting point by the property of efficient computation.

Section 2.2 reviews several local search neighborhoods that have been applied to the Linear Ordering Problem, and adds the Block Insertion neighborhood described above. Constant-time-per-neighbor search is an important property of each. Section 2.4 describes simultaneous versions of each neighborhood, including nested block insertions. Each of these neighborhoods has an exponential number of members, but the same polynomial search time as their single-move counterparts. All these neighborhoods fit into a hierarchy, where the number of computations required for search is related to the number of local maxima.

Neighborhoods lead directly to local search algorithms. This dissertation explores one very simple local search approach—*greedy* local search, or *hillclimbing*—in Section 2.5.2. It applies hillclimbing to benchmark Linear Ordering Problems using a number of neighborhoods, including the one described informally above, and subsets of it searchable in $\Theta(n^2)$ time instead, in Section 2.6.

Evaluation of local search methods for the Linear Ordering Problem uses XLOLIB, a collection of large random problems derived from real-world economics data. A shortcut version of this dissertation's Block Insertion neigborhood proves to have excellent performance on this benchmark, surpassing prior methods at hillclimbing with random restarts.

On these same benchmarks, the performance of the very large-scale neighborhood (VLSN) search is disappointing. It fails to outperform the shortcut methods. This may be due to the fact that though the VLSNs contain far more permutations than their simple counterparts, they have the same local maxima. This is a key point of discussion.

Procedural descriptions of neighborhoods are common in the literature, but it proves both elegant and useful to describe them using grammars. Both finite-state and context-free grammars appear. The Linear Ordering Problem supplies weights for the grammar rules, and the dissertation then applies techniques from natural language processing to find best permutations in the neighborhoods expressed by the weighted grammars. For example, instead of finding just the single best permutation, a parser can find the K best permutations in the neighborhood, for some $K > 1$. Other such methods include pruning, A* parsing, and forest rescoring.

Many useful computations that a parser can perform depend on a one-to-one correspondence between permutations and derivations of those permutations under the grammar. The simplest grammars describing neighborhoods often have spurious ambiguity, representing some permutations with many distinct derivations. After the introduction of grammars in Section 2.8, Section 2.9 proceeds to complicate those grammars in order to arrive at normal forms that prohibit multiple derivations. Some of these normal forms are novel.

The immediate consequence of the normal forms is that they make possible the computation of the sizes of the neighborhoods—the cardinalities of the sets of unique permutations that they contain. Section 2.10 derives recurrence equations from the normal-form grammars and uses them to perform these computations for several neighborhoods. More important consequences of normal form arise when the Linear Ordering Problem is considered from the standpoint of learning.

Finally, Section 2.11 invokes analogies to sorting algorithms to analyze the diameters of graphs induced by the neighborhoods, and gives a linear-time shift-reduce parser for determining neighborhood membership. It also introduces a LOP matrix that defines a total order, which will prove useful for machine learning.

1.2 Machine Translation

The field of natural language processing includes ordering problems of its own. One application is multi-document extractive summarization, which Section 2.1.1 will touch upon.[1] Of particular interest to this dissertation is the problem of machine translation. Adequate translation between human languages, using software, is an important and difficult task.

Different human languages express the same or nearly the same concepts in a variety of ways. One such divergence is word ordering. An example of particular interest to this dissertation is the difference between German and English. German verbs often occur at the very ends of their clauses. English verbs, on the other hand, usually occur between their subjects and their objects. As a result, machine translation from German to English must "move" the concept represented by the German verb, across a potentially long distance occupied by intervening words, in order to produce a fluent translation. Section 3.4 will discuss this and other divergences between German and English in more detail.

This dissertation combines a reordering model derived from the Linear Ordering Problem and other permutation problems with monotone finite-state translation. Weighted finite-state transducers are a weighted generalization of finite-state automata that map strings from one regular language to strings from another. Section 3.5 will give formal definitions. It will also explore possible representations for the monotonic WFST translation model. In general, such transducers are capable of capturing all sorts of interesting phenomena related to monotonic translation, from the morpheme level up to phrases. Also, if the target language model is finite-state, composition allows it to participate in the ordering model as well.

Section 3.6 will introduce the Linear Ordering Problem as a model of word reordering for machine translation. The idea is to assign scores to pairs of words in the sentence for remaining in order or reversing order based on features observable in the sentence as a whole. This section will add a model of monotonic translation that may also express preferences for different orders, using WFSTs.

The finite-state automata actually perform two roles in this model. The first is to locally constrain the reordering, as in the TSP. The second is to perform monotonic translation. Through WFST composition, these two models can combine into a single transducer.

Section 3.9 will address the implications of this automaton model for the grammars and algorithms that Chapter 2 will apply to the Linear Ordering Problem alone. The situation is not nearly as tidy as for the LOP. The grammars must account for paths through the automaton by keeping track of states, and that leads to another cubic factor in the asymptotic runtime, this one in the number of states $|Q|$. The resulting

[1]Information ordering in general is a difficult task, even for humans. I can attest to this after trying to construct a linear ordering of this introduction from the heavily interrelated topics of the ensuing chapters.

runtime of $\Theta(n^3 |Q|^3)$ is quite large for the models of interest. A trigram language model component alone contributes a factor of n^6, without accounting for translation at all.

Sections 3.9 and 3.10 will adapt several methods from the natural language parsing and machine translation communities to the grammars of this dissertation. Section 3.9.2 proposes A* search as one possible solution to the runtime problem. Section 3.9.3 proposes another based on the most successful LOP search algorithm that Chapter 2 will present—Block $\mathcal{LS}_f$. Section 3.10 introduces further approximations, in line with more standard MT decoders. Exact methods for neighborhood search fail to be practical because of the order of the polynomials in the asymptotic runtime.

This chapter will also use the models derived from the LOP alone as a preprocessing step. Section 3.8 will show that this is a rewarding approach. A standard phrase-based decoder that uses German′ as input instead of German produces significantly better translations into English under several automatic evaluation measures. In particular, using the LOP reordering model as preprocessing outperforms the standard lexicalized reordering model integrated into the phrase-based Moses decoder. German′ reorders German using the LOP search methods of Chapter 2.

1.3 Machine Learning

Using a Linear Ordering Problem for machine translation requires a matrix of precedence scores. The success of the translation experiments depends heavily on finding a parametric model from which such a matrix can be derived for each given input sentence, as well as a good setting for the parameters of the model. Section 4.4 proposes a linear model for each matrix entry, and adapts a set of features from the task of dependency parsing, arguing an analogy between the two tasks. The rest of Chapter 4 searches for a good set of parameters for the linear model.

Parameter optimization is a search problem of a rather different character from combinatorial optimization. The size and nature of the permutation search space renders several standard machine learning algorithms unfit for this problem, and leads to the development of alternatives.

Even simple learning algorithms such as the perceptron do not apply out of the box. The problem is that the standard perceptron update requires access to the best output according to the current model. In this case, finding that best output means solving the Linear Ordering Problem. The only alternative is to adapt the perceptron to deal with non-optimal outputs. Fortunately, this is straightforward and it seems to work well. Section 4.7 will adapt perceptrons to the LOP learning problem, while Section 4.9 will introduce explicit large-margin updates of the same type. Section 4.8 will explicitly account for the fact that learning is taking place in the context of greedy local search.

Likelihood-based methods are more attractive than perceptrons in some respects.

For one thing, they take more information into account at each update. However, interpreting LOP scores as probabilities means interpreting the LOP matrix as a log-linear model and computing an intractable partition function. Section 4.11 will propose sampling as one solution, and several tractable alternatives based on conditional likelihood distributions given a single neighborhood.

These likelihood-based methods use the grammars of Chapter 2, with parsing algorithms modified to sum over derivations instead of simply finding one best. Use of the normal-form grammars that Chapter 2 will introduce is therefore necessary to avoid skewing the distribution toward permutations with more derivations. In addition, some of the methods make use of K-best lists, for which normal form is also appropriate.

The most reliable evaluation of learned parameters is the translation task of Chapter 3. However, this evaluation is quite expensive. Chapter 4 will also explore several other measures of reordering performance, including Kendall's τ rank correlation and a monolingual version of BLEU score, and will measure correlations among them.

Some of these loss functions have useful properties that make them attractive to use either directly for optimization or as a convergence criterion. Although these do produce models that can improve translation quality, their prediction is not perfect. Experiments on learning vary several meta-parameters, including feature sets, learning methods, learning rates, oracle orderings, and parallelization.

1.4 Summary

In sum, this work connects several problems in novel ways. It applies sophisticated dynamic programming, familiar to natural language parsing, to improve search for the Linear Ordering Problem. It introduces the Linear Ordering Problem, and its local search methods, to machine translation, where it stands in contrast to existing reordering models in important ways. Finally, it introduces machine learning for the Linear Ordering Problem, and adapts several learning algorithms using search and neighborhoods. Each of these areas is vast and this dissertation only invokes small portions of each, but it also opens up new possibilities that may lead to further fruitful cross-pollination.

Some of the ideas appearing here were published in Eisner and Tromble (2006), with minimal empirical validation. Other ideas are completely novel to the dissertation. All of the experimental results reported here are new.

1.5 Road Map

The novel contributions and results of the dissertation are contained in the next three chapters, described at a high level above. Chapter 2 is largely independent of the other two, though there will be some forward references. Chapters 3 and 4, however, are heavily interdependent. For some readers, it may make sense to read Chapter 4 first, because Chapter 3 will invoke the models learned there.

The last section of each chapter will present a summary of the chapter and highlight the most important points therein. Chapter 5 will briefly summarize the contributions of the dissertation again.

The Appendix contains technical information that would be a digression in the main text. Appendix A contains a description of several machine translation evaluation measures and significance tests. Appendix B presents the German part of speech and dependency label sets from the TIGER treebank.

Chapter 2

Local Search for the Linear Ordering Problem

This chapter describes the linear ordering problem (LOP), surveys its history, and describes a benchmark problem set called LOLIB. It then introduces the concept of a *neighborhood* and describes several neighborhoods with nice computational properties, including very large-scale neighborhoods, which have exponential size but polynomial search time. Next, it applies greedy local search, which works with any neighborhood, to the benchmark problems.

The second half of the chapter, beginning with Section 2.8, presents results of a more theoretical nature. First, it defines the very large-scale neighborhoods in terms of grammars that generate them. Then it moves on to normal-form grammars, which eliminate the derivational ambiguity of the simple grammars. Normal-form grammars allow computation of the neighborhood size, because they have only one derivation per neighbor. The chapter's final section considers the graph that neighborhoods induce on the set of permutations, and investigates the graph diameter, as well as other properties.

The results of this chapter stand on their own and may prove of interest to some or any of the application areas of the Linear Ordering Problem. At the same time, the algorithms presented here are essential to the rest of the dissertation, and some forward references occur where appropriate.

2.1 The Linear Ordering Problem

The linear ordering problem is an NP-complete permutation optimization problem, like the famous traveling salesman problem (TSP). Membership in NP of the decision-problem formulation of the LOP will be clear from its definition below. Karp (1972) reduced the vertex cover problem to the minimum feedback arc set problem, the weighted version of which is equivalent to the LOP, demonstrating that it is

NP-hard.[1]

Definition 2.1 *Let $\mathcal{S}$ be an arbitrary set. A* **permutation** $\pi : \mathcal{S} \mapsto \mathcal{S}$ *is a bijection from $\mathcal{S}$ onto itself.*

In this dissertation, $\mathcal{S}$ will usually be the set $\{1, 2, \ldots, n\}$, for some natural number n. Although it is defined as a function, it will be useful to think of a permutation π as a *sequence*:

$$\pi = \pi_1 \ \pi_2 \ \ldots \ \pi_n \overset{\text{def}}{=} \pi^{-1}(1) \ \pi^{-1}(2) \ \ldots \ \pi^{-1}(n). \tag{2.1}$$

That is, π is defined by the number that it maps to 1, followed by the number that it maps to 2, etc. For example, if π is 3 1 2 then $\pi(1) = 2$, $\pi(2) = 3$, and $\pi(3) = 1$.

Definition 2.2 *Let Π_n be the set of permutations of the integers $\{1, 2, \ldots, n\}$, such that $|\Pi_n| = n!$. Then, given an $n \times n$ matrix B, $B[i, j] \in \mathbb{R}$, of scores, the* **linear ordering problem** *(LOP) is to find the optimal permutation:*

$$\pi^* = \arg\max_{\pi \in \Pi_n} B(\pi) = \arg\max_{\pi \in \Pi_n} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} B[\pi_i, \pi_j]. \tag{2.2}$$

$B[\ell, r]$ represents the value of putting ℓ before r in the permutation, and the total value of a permutation is the sum of the values of all $\binom{n}{2}$ pairwise orderings.

An equivalent problem statement is to find a permutation of the rows and columns of the matrix B that maximizes the sum of the entries above the diagonal (Garcia, Pérez-Brito, Campos, and Martí, 2006).

In words, the object of the LOP is to find a total order of a set of objects that best satisfies a set of, in general, mutually-inconsistent pairwise precedence preferences. The LOP setting has no regard for adjacency. Unlike the traveling salesman problem, in the LOP two items will be adjacent in π^* only because nothing fits between them, never because they belong together.

2.1.1 History and Applications

Charon and Hudry (2007) attributed the linear ordering problem to Condorcet (1785), rediscovered by Kemeny (1959), and a different formulation to Slater (1961). The first formulation comes from the origins of social choice theory. The problem is to rank a set of candidates given voters' preferences among them. Here, the matrix entry $B[\ell, r]$ holds the number of voters who prefer candidate ℓ to candidate r.

The second formulation comes from psychological experiments involving paired comparisons, where the problem is to find a linear ordering maximally consistent

[1]The maximization version of the LOP, given here, is known to be APX-complete, meaning there is no polynomial time approximation scheme, unless P=NP. The minimization version is APX-hard, and conjectured to be outside APX (Mishra and Sikdar, 2004).

with a subject's possibly inconsistent preferences. Here, $B[\ell, r] \in \{0, 1\}$ indicates whether the subject preferred item ℓ to item r.

Another instance of the linear ordering problem originated in the field of economics, where it was posed as minimizing the sum of entries below the diagonal in permutations of *input-output matrices* (Chenery and Watanabe, 1958). An input-output matrix represents relationships among sectors of an economy. Here, $B[\ell, r]$ is a monetary measure of the flow of goods from sector ℓ to sector r. The optimal ordering of the economic sectors provides insight into the structure of the economy.

The problem is also known as *Kemeny's problem* or *Kemeny's rule*, the *median order problem*, and the *permutation problem* (Charon and Hudry, 2007). It is equivalent to the minimum reversing set problem, the minimum weighted feedback arc set problem, and the maximum acyclic subgraph problem. Lenstra (1973) also formulates it as an instance of the quadratic assignment problem.

Glover, Klastorin, and Klingman (1974) proposed finding a minimum weighted feedback arc set for chronological ordering of pottery types discovered at gravesites. This is the stratification problem of archaeology. Here, $B[\ell, r]$ is some measure of the occurrences of pottery type ℓ below type r at a single site. The measure proposed by Glover et al. (1974) accounts for the relative depths of the artifacts.

In the area of graph drawing, the problem of minimizing the number of crossing edges in a two-layer bipartite graph with the order of one layer fixed can be cast as a linear ordering problem (Eades and Whitesides, 1994; Jünger and Mutzel, 1997). Here, $B[\ell, r]$ is the number of resulting crossings among the edges incident upon nodes ℓ and r if ℓ should precede r. This can be computed because the order of their neighbors—in the other layer—is fixed. In this case, (2.2) is a minimization.

Cohen, Schapire, and Singer (1999) proposed a linear ordering problem for search engine combination. In this setting, $B[\ell, r]$ is an aggregation of the rankings of several search engines. This bears an obvious resemblance to Kemeny's problem, but differs because the "voters" are weighted. Section 4.2 on page 130 discusses this work in more detail as an instance of learning the matrix B.

Finally, Grötschel, Jünger, and Reinelt (1984) cited Boenchendorf (1982) as the source of another equivalent problem—minimizing total weighted completion time in one-machine scheduling. Grötschel et al. (1984) also proposed ranking players or teams in sports tournaments as another application.

One potential application of the LOP in natural language processing is multi-document extractive summarization. This bears some resemblance to the stratification problem—compiling information from multiple sites—but is more complex because of the extraction component—only a relatively small subset of the many available sentences will be used. Barzilay, Elhadad, and McKeown (2002) proposed a "Majority Ordering" procedure that creates an instance of the linear ordering problem for multidocument summarization. Like Cohen et al. (1999), they used a greedy heuristic to find a candidate ordering.

Chapter 3 of this dissertation will introduce word reordering for machine translation as another application of the LOP. It will cast word reordering in terms of the LOP by letting the $\{1, 2, \ldots, n\}$ represent indices into a sentence. The matrix entry $B[\ell, r]$ then assigns value to the relative order of the pair of words ℓ and r. They derive from a given sentence as follows:

$$B[\ell, r] = \boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{w}, \ell, r), \tag{2.3}$$

where $\boldsymbol{\theta}$ is a weight vector to learn (see Chapter 4), $\boldsymbol{\phi}$ is a vector of feature functions (see Section 4.4), and $\boldsymbol{w}$ contains the words in the sentence. In addition to the words themselves, Section 4.4 will introduce features that consider parts of speech and dependency parses.

2.1.2 LOLIB and XLOLIB

The economics application of the linear ordering problem—triangulation of input-output matrices—is probably the most studied of any of those the previous section describes. It is also an application for which there is a great deal of data available.

LOLIB is a collection of 49 input-output matrices for fifty sectors of various European economies. It is maintained by Gerhard Reinelt.[2] The problem size, $n = 50$, is small enough to be solved using exact methods (see Section 2.5), and is therefore not sufficient to challenge local search methods. However, these real-world problems have a different character from randomly generated problem instances, such as those used by Mitchell and Borchers (2000) and Laguna, Martí, and Campos (1999).

For this reason, Schiavinotto and Stützle (2004) introduced XLOLIB, problem instances of size $n = 150$ and $n = 250$ generated by resampling matrix entries from corresponding LOLIB problems. They show that these new problem instances resemble the real LOLIB problems more closely than other random problems using several measures.

The following sections introduce several neighborhoods and search procedures for the LOP, and Section 2.6 applies a search quality comparison described there to these procedures, benchmarking performance on XLOLIB250.

2.2 Neighborhoods

This section reviews several neighborhoods for permutation search known in the literature, and introduces neighborhoods new to this work. It describes these neighborhoods in terms of sets of functions, and places them all into a subsumption hierarchy. Neighborhoods constitute the inner loop of the local search algorithms that Section 2.5.2 describes. It is obviously intractable to consider *all* the permutations, so local search looks only at those in a single neighborhood at a time. This presentation

[2] http://www.informatik.uni-heidelberg.de/groups/comopt/software/LOLIB/index.html

of neighborhoods focuses on properties related to search, especially the complexity of finding the best neighbor.

Recall that Π_n represents the set of permutations of the integers from 1 to n, such that $|\Pi_n| = n!$ and that Definition 2.1 described a permutation as a function. It is possible to compose permutation functions to produce new permutations.

Let π be one permutation in Π_n and let σ be another, possibly identical. Then,

$$\sigma(\pi)_i = \pi^{-1}(\sigma^{-1}(i)) = \pi_{\sigma_i}. \tag{2.4}$$

For example, let σ be the permutation 2 5 3 1 4, and let π be the permutation 3 1 5 2 4. Then $\sigma(\pi) = 1\ 4\ 5\ 3\ 2$. That is, it is the second member of π followed by the fifth, third, first, and fourth.

Definition 2.3 *A* **neighborhood** *is a set of permutations.*

That is, if $\mathcal{N}$ is a neighborhood then $\mathcal{N} \subseteq \Pi_n$, for some n. We can also treat a neighborhood as a function, such that

$$\mathcal{N}(\pi) = \{\sigma(\pi) \mid \sigma \in \mathcal{N}\}$$

In a slight abuse of notation, also let *neighborhood* refer to a class of neighborhoods $\mathcal{N}_n$ derived by some rule for each n. That is the sense the following sections use.

2.2.1 Transposition

Schiavinotto and Stützle (2004) experimented with several different neighborhoods from the linear ordering problem literature, including transposition, which they called *swap*, insertion, and interchange.

The simplest neighborhood of interest is the transposition neighborhood,

$$\text{Trans}_n = \left\{ \begin{array}{c} 2\ 1\ 3\ 4\ \ldots\ n-1\ n, \\ 1\ 3\ 2\ 4\ \ldots\ n-1\ n, \\ 1\ 2\ 4\ 3\ \ldots\ n-1\ n, \\ \vdots \\ 1\ 2\ 3\ 4\ \ldots\ n\ n-1 \end{array} \right\}. \tag{2.5}$$

Trans_n has size $|\text{Trans}_n| = n - 1$ and consists of all permutation functions with symmetric difference distance 1 from the identity permutation. Section 2.11.2 defines this distance measure.

2.2.2 Insertion

Let the Insert_n^k neighborhood refer to the set of permutation functions of length n that move the kth item to a new position,

$$\text{Insert}_n^k = \left\{ \begin{array}{l} k\ 1\ 2\ \ldots\ k-1\ k+1\ \ldots\ n, \\ 1\ k\ 2\ \ldots\ k-1\ k+1\ \ldots\ n, \\ \qquad\qquad \vdots \\ 1\ 2\ \ldots\ k\ k-1\ k+1\ \ldots\ n, \\ 1\ 2\ \ldots\ k-1\ k+1\ k\ \ldots\ n, \\ \qquad\qquad \vdots \\ 1\ 2\ \ldots\ k-1\ k+1\ \ldots\ k\ n, \\ 1\ 2\ \ldots\ k-1\ k+1\ \ldots\ n\ k \end{array} \right\}. \tag{2.6}$$

This neighborhood omits the identity permutation, and has size $n-1$ as a result. Picking up k leaves a sequence of $n-1$ items, with n possible destinations corresponding to the zero-width offsets (see Figure 2.1), but one of those destinations puts k back where it started.

The full $\text{Insert}_n = \bigcup_{k\in\binom{n}{1}} \text{Insert}_n^k$ neighborhood has size $(n-1)^2$—it is the union of n neighborhoods of size $n-1$ each, but those neighborhoods are not entirely disjoint. Insert_n subsumes Trans_n, and the members of Trans_n each appear twice. $\text{Trans}_n \subset \text{Insert}_n$ for $n > 2$. Trans_3 contains just 2 1 3 and 1 3 2, while Insert_3 also contains 2 3 1 and 3 1 2.

2.2.3 Interchange

The neighborhood that Schiavinotto and Stützle called *interchange* consists of the exchange of any two items in the permutation, which need not be adjacent. It is clearly a superset of Trans_n. It can also be thought of as two simultaneous insertions—interchange of positions i and j moves π_i to position j and π_j to position i. The interchange neighborhood has size $\binom{n}{2}$, and is therefore smaller than Insert_n for $n > 2$.

In addition, the local maxima of the interchange neighborhood are a superset of the local maxima of Insert_n, as Section 2.5.3 will argue. This neighborhood therefore offers no computational advantage over Insert_n, and this dissertation will not consider it further.

2.2.4 Block Insertion

This section introduces a new set of block insertion neighborhoods. Block insertions have apparently not previously been used for local search for the linear ordering problem, though there is related work—see Section 2.7 for a discussion.

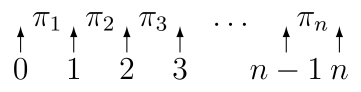

Figure 2.1: Zero-width offsets as indices into a permutation.

Definition 2.4 *For a given permutation π, let (i, j) refer to the* **subsequence** *between the zero-width offsets (see Figure 2.1) i and j. That is,*

$$(i, j) \overset{\text{def}}{=} \pi_{i+1}\ \pi_{i+2}\ \ldots\ \pi_j. \tag{2.7}$$

In this notation, the entire permutation π is $(0, n)$, and the three subsequences $(0, i), (i, j), (j, n)$ combine to form the entire sequence. (i, i) denotes an empty subsequence.

In a slight abuse of notation, let the binomial coefficients have the following interpretation as sets:

$$\begin{array}{ll} \binom{n}{1} \overset{\text{def}}{=} \{i \mid 1 \leq i \leq n\} & \binom{n+1}{1} \overset{\text{def}}{=} \{i \mid 0 \leq i \leq n\} \\ \binom{n}{2} \overset{\text{def}}{=} \{(i, j) \mid 1 \leq i < j \leq n\} & \binom{n+1}{2} \overset{\text{def}}{=} \{(i, j) \mid 0 \leq i \leq n\} \\ \binom{n}{3} \overset{\text{def}}{=} \{(i, j, k) \mid 1 \leq i < j < k \leq n\} & \binom{n+1}{3} \overset{\text{def}}{=} \{(i, j, k) \mid 0 \leq i < j < k \leq n\} \end{array} \tag{2.8}$$

Let the neighborhood $\text{Insert}_n^{(i,j)}$, $(i, j) \in \binom{n+1}{2}$ consist of all permutations that move the subsequence (i, j) to a new position k, with $k < i$ or $j < k$. The block insert neighborhoods generalize the insert neighborhoods, with $\text{Insert}_n^k \equiv \text{Insert}_n^{(k-1,k)}$.

If $k < i$, the resulting permutation is $(0, k)\ (i, j)\ (k, i)\ (j, n)$. This could equivalently be considered a transposition of blocks (k, i) and (i, j). If $j < k$, on the other hand, the result is $(0, i)\ (j, k)\ (i, j)\ (k, n)$. This is a transposition of blocks (i, j) and (j, k). For example, $k = 1$ gives the following member of $\text{Insert}_6^{(3,5)}$:

$$1\ \underbrace{4\ 5}_{(3,5)}\ 2\ 3\ 6.$$

This neighborhood also omits the identity permutation, and has size $n - (k - i)$ as a result—(i, k) has $k - i$ items, so removing it leaves a sequence of $n - (k - i)$ items, and therefore $n - (k - i)$ possible new destinations for (i, k) .

Call the entire neighborhood

$$\text{BlockInsert}_n = \bigcup_{(i,k) \in \binom{n+1}{2}} \text{Insert}_n^{(i,k)}. \tag{2.9}$$

The size of BlockInsert_n is $\binom{n+1}{3} = \frac{n^3 - n}{6}$. Summing the sizes of the individual neighborhoods would give twice this number, since each transposition $(0, i)\ (j, k)\ (i, j)\ (k, n)$ is a member of two neighborhoods—$\text{Insert}_n^{(i,j)}$ and $\text{Insert}_n^{(j,k)}$.

Because $\text{Insert}_n^k \equiv \text{Insert}_n^{(k-1,k)}$, it is clear that $\text{Insert}_n \subseteq \text{BlockInsert}_n$. The inclusion is strict for $n > 3$. For example, BlockInsert_4 includes 3 4 1 2, which Insert_4 does not.

2.2.5 Limited-width Block Insertion

BlockInsert_n allows subsequences of any length to relocate. Limiting the size of the subsequence that can move to lengths $\leq w$ leads to a subset of BlockInsert_n:

$$\text{Insert}_n^{\leq w} \overset{\text{def}}{\equiv} \bigcup_{(i,k)\in\binom{n+1}{2},\ k-i\leq w} \text{Insert}_n^{(i,k)}. \tag{2.10}$$

The special case $w = 1$ is $\text{Insert}_n^{\leq 1} \equiv \text{Insert}_n$. If $w \geq \frac{n}{2}$, then $\text{Insert}_n^{\leq w} \equiv \text{BlockInsert}_n$.

The size of $\text{Insert}_n^{\leq w}$ is more complicated to compute than the others. As with the definition of BlockInsert_n, the sets that $\text{Insert}_n^{\leq w}$ unites are not disjoint—every transposition of (i, j) and (j, k) such that both have width $\leq w$ appears in two subsets. There does not appear to be an elegant closed form, so it will suffice to observe that

$$\left|\text{Insert}_n^{\leq w}\right| = \Theta(wn^2). \tag{2.11}$$

2.3 Neighborhoods and Computation

Under Definition 2.3, any subset of the permutations is a neighborhood. What makes the neighborhoods described in Section 2.2 interesting are their computational properties. In general, computing the best permutation in a neighborhood $\mathcal{N}$ under a $n \times n$ linear ordering problem matrix B costs $\Theta(|\mathcal{N}|\, n^2)$ time. Computing $B(\pi)$ costs $\Theta(n^2)$ for each neighbor π. However, the neighborhoods in Section 2.2 have the special property that their best neighbor can be found in $\Theta(|\mathcal{N}|)$ time—constant time per neighbor.

2.3.1 Search in the Transposition Neighborhood

The Trans_n neighborhood, of size $\Theta(n)$ would cost $\Theta(n^3)$ to search naïvely. However, assuming the total score $B(\pi)$ of the current permutation π is known, the score of the best permutation in $\text{Trans}_n(\pi)$ can be computed in $\Theta(n)$ time.[3] Each of the neighbors swaps a single pair of adjacent elements in π, say π_i and π_{i+1}. The resulting permutation incurs all of the same scores as π except instead of $B[\pi_i, \pi_{i+1}]$, it incurs $B[\pi_{i+1}, \pi_i]$. Therefore, letting $\pi^{(i)}$ represent the ith member of Trans_n,

$$B(\pi^{(i)}) = B(\pi) - B[\pi_i, \pi_{i+1}] + B[\pi_{i+1}, \pi_i]. \tag{2.12}$$

The score of each permutation in Trans_n can thus be computed in constant time.

[3]Even if $B(\pi)$ is unknown, it is still possible to compute the best neighbor in $\text{Trans}_n(\pi)$ by this method. Simply ignore the $B(\pi)$ term.

2.3.2 Search in the Insertion Neighborhood

Similar observations lead to improved runtime for Insert_n^k as well. Again, assuming $B(\pi)$ is known, searching $\text{Insert}_n^k(\pi)$ requires only $\Theta(n)$ time. Inserting π_k one position to the left or to the right is simply an adjacent transposition. Computing the scores of these neighbors first allows computation of the scores of inserting to positions $k-2$ and $k+2$ using the same technique. That is, let $\pi^{(i)}$ now represent the permutation that results from inserting π_k at position i. Then,

$$B(\pi^{(k-2)}) = B(\pi^{(k-1)}) - B[\pi_{k-2}, \pi_k] + B[\pi_k, \pi_{k-2}]. \tag{2.13}$$

The scores of $\pi^{(k-3)}, \ldots, \pi^{(1)}$ follow similarly, each in constant time given its predecessors. Likewise $\pi^{(k+2)}, \ldots, \pi^{(n)}$. The entire Insert_n neighborhood can thus be searched in $\Theta(n^2)$ time. Schiavinotto and Stützle used this method for their $\mathcal{LS}_f$ procedure, which Section 2.5 describes.

2.3.3 Search in the Block Insertion Neighborhood

The individual $\text{Insert}^{(i,j)}$ neighborhoods cannot be searched in constant time per member separately. However, the entire BlockInsert_n neighborhood can. Start by searching all the $\text{Insert}_n^{(i,i+1)}$ neighborhoods—the simple insert neighborhoods—as described in Section 2.3.2. Then move to $\text{Insert}^{(i,i+2)}$, etc.

Consider $\text{BlockInsert}_n(\pi)$, where again $B(\pi)$ is already known. Let $\pi^{(i,j,k)}$ be the permutation that transposes (i,j) and (j,k) and let

$$\Delta(i,j,k) = B(\pi^{(i,j,k)}) - B(\pi). \tag{2.14}$$

The procedure described below computes each Δ in constant time by dynamic programming. First, observe that

$$\Delta(i,j,k) = \sum_{\ell=i+1}^{j} \sum_{r=j+1}^{k} \Delta[\pi_\ell, \pi_r], \tag{2.15}$$

where

$$\Delta[\pi_\ell, \pi_r] = B[\pi_r, \pi_\ell] - B[\pi_\ell, \pi_r] \tag{2.16}$$

is the change in score due to transposing π_ℓ and π_r. That is, the total difference between the scores of the permutations is determined by differences in pairwise orderings from $(i,j) \times (j,k)$.

These sums overlap in advantageous ways. For example,

$$\Delta(i+1,j,k) = \sum_{\ell=i+2}^{j} \sum_{r=j+1}^{k} \Delta[\pi_\ell, \pi_r],$$

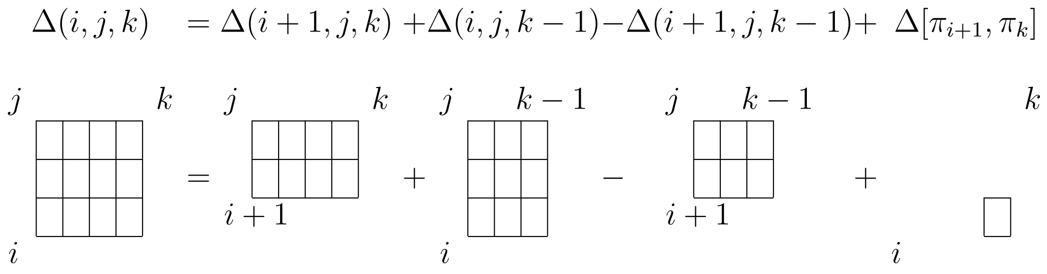
$\Delta(i,j,k) \quad = \Delta(i+1,j,k) + \Delta(i,j,k-1) - \Delta(i+1,j,k-1) + \Delta[\pi_{i+1},\pi_k]$
j
k
i
$=$
j
k
$i+1$
$+$
j
$k-1$
i
$-$
j
$k-1$
$i+1$
$+$
k
i

Figure 2.2: Visualization of block insertion dynamic programming. Each box represents a single $\Delta[\pi_\ell,\pi_r]$, with $\ell \in (i,j)$ running from bottom to top and $r \in (j,k)$ running from left to right.

so,

$$\Delta(i,j,k) - \Delta(i+1,j,k) = \sum_{r=j+1}^{k} \Delta[\pi_{i+1},\pi_r].$$

This reduces the computational cost from $\Theta(n^2)$ to $\Theta(n)$ if $\Delta(i+1,j,k)$ is already available. Further, notice that

$$\Delta(i,j,k-1) - \Delta(i+1,j,k-1) = \sum_{r=j+1}^{k-1} \Delta[\pi_{i+1},\pi_r], \tag{2.17}$$

so that the difference between these two differences is just $\Delta[\pi_{i+1},\pi_k]$. Therefore, if quantities are computed in the correct order and stored, each Δ can be arrived at incrementally in constant time:

$$\Delta(i,j,k) = \underbrace{\Delta(i+1,j,k)}_{\leftarrow} + \underbrace{\Delta(i,j,k-1)}_{\rightarrow} - \Delta(i+1,j,k-1) + \Delta[\pi_{i+1},\pi_k]. \tag{2.18}$$

Figure 2.2 illustrates this dynamic program with overlapping rectangles.

Consider a particular neighborhood $\text{Insert}_n^{(\ell,r)}(\pi)$, which inserts the block (ℓ,r) both to its left and to its right. Insertions to the left score $B(\pi)+\Delta(i,\ell,r)$, where $i<\ell$, and require access to $\Delta(i+1,\ell,r)$, $\Delta(i,\ell,r-1)$, and $\Delta(i+1,\ell,r-1)$. The latter two involve a narrower block, $(\ell,r-1)$, which will always be available when considering the block insert neighborhoods from narrowest to widest. The first term, on the other hand, involves the same span (ℓ,r). As the arrow indicates, these terms must therefore be computed in right-to-left order.

Similarly, insertions to the right score $B(\pi)+\Delta(\ell,r,k)$, where $r<k$. In addition to terms involving a narrower span, these will require access to $\Delta(\ell,r,k-1)$, which must therefore be computed left-to-right.

The procedure that results is to start with $\Delta(\ell-1,\ell,r)$ and decrement the first index until arriving at $\Delta(0,\ell,r)$. Then start with $\Delta(\ell,r,r+1)$ and increment the third index until arriving at $\Delta(\ell,r,n)$.

The base cases of the dynamic program, $\Delta(i,i,k)$ and $\Delta(i,k,k)$, must be initialized to zero. Using these update equations, the entire BlockInsert$_n$ neighborhood can be searched in $\Theta(n^3)$ time, or constant time per neighbor.

The next section describes procedures for searching related neighborhoods of exponential size in the same polynomial time, using similar dynamic programming.

2.4 Very Large-Scale Neighborhoods

Each of the neighborhoods described in the previous section can be generalized by considering multiple nested or non-overlapping changes simultaneously. The result, in each case, is a neighborhood of size exponential in the length of the permutation, which can nonetheless be searched in polynomial time. These are Very Large-Scale Neighborhoods (VLSNs) (Ahuja, Orlin, and Sharma, 2000). From the standpoint of local search, considering an exponential number of neighbors in polynomial time is an enormous boon.

2.4.1 Non-Overlapping Transpositions

Exploration of the Trans$_n$ neighborhood computes $\Delta[\pi_{i-1},\pi_i]$ for all $i \in \binom{n}{1}$. If many of these are positive, it would be beneficial to make all the changes at once. This idea leads to a neighborhood that allows many simultaneous non-overlapping transpositions. This is one of the Dynasearch neighborhoods of Potts and van de Velde (1995). Refer to this neighborhood as Trans$^*_n$.

Transposing π_{i-1} and π_i renders those items unavailable for other transpositions. The neighborhood cannot also transpose π_{i-2} and π_{i-1} or π_i and π_{i+1}. Therefore, it may not necessarily be possible to make *all* of the transpositions with positive Δs. Computing the best neighbor in Trans$^*_n(\pi)$ requires a bit of sophistication as a result.

Section 2.8.1 later in this chapter describes this neighborhood in terms of both a context-free grammar and a finite-state acceptor. Section 2.10.1 shows that it does indeed have exponential size. For now, a simple dynamic program for computing the best neighbor will suffice. Let Δ^*_i be the change in score of the best rearrangement, in the Trans$^*_n$ neighborhood, of the subsequence of π indexed by $(0,i)$. Δ^*_0 and Δ^*_1 are both zero because no transpositions are possible on the empty sequence or a sequence of length one. For $2 \le i \le n$,

$$\Delta^*_i = \max\left(\Delta^*_{i-1},\ \Delta^*_{i-2} + \Delta[\pi_{i-1},\pi_i]\right). \tag{2.19}$$

The best neighbor up to position i either uses the best neighbor up to $i-1$ and defers the question of whether π_i swaps with π_{i+1}, or it swaps π_i with π_{i-1} and combines

```
procedure TRANS*$(B, \pi)$
    $\Delta^*[0] \leftarrow 0$
    $\Delta^*[1] \leftarrow 0$
    $p[1] \leftarrow 0$
    for $i \leftarrow 2$ to $|\pi|$ do
        $\Delta[\pi_{i-1}, \pi_i] \leftarrow B[\pi_i, \pi_{i-1}] - B[\pi_{i-1}, \pi_i]$
        if $\Delta^*[i-1] \geq \Delta^*[i-2] + \Delta[\pi_{i-1}, \pi_i]$ then
            $\Delta^*[i] \leftarrow \Delta^*[i-1]$
            $p[i] \leftarrow i-1$
        else
            $\Delta^*[i] \leftarrow \Delta^*[i-2] + \Delta[\pi_{i-1}, \pi_i]$
            $p[i] \leftarrow i-2$
        end if
    end for
    for $(i \leftarrow |\pi|;\ i > 0;\ i \leftarrow p[i])$ do
        if $p[i] = i-2$ then
            SWAP$(\pi_{i-1}, \pi_i)$
        end if
    end for
    return $\pi$
end procedure
```

Figure 2.3: Pseudocode for computing the best neighbor in $\text{Trans}_n^*(\pi)$

that with the best neighbor up to $i-2$. Finding the best neighbor given the best score Δ_n^* is a simple matter of keeping back-pointers during dynamic programming and following them from the end. Figure 2.3 shows pseudocode.

2.4.2 Nested Insertions and Block Insertions

The key insight in generalizing the BlockInsert_n neighborhood is that the change in score to transpose adjacent blocks (i, j) and (j, k) is independent of the order of the items within those blocks. That is, $\Delta(i, j, k)$ from (2.15) is the same regardless of permutations of the items within (i, j) or (j, k).

A top-down description of this neighborhood, called BlockInsert_n^*, may be the simplest. Choose any index $1 \leq j \leq n-1$ and choose either to keep $(0, j)$ and (j, n) in order with no change of score, or to exchange them, changing the score by $\Delta(0, j, n)$. In either case, perform the same procedure recursively on $(0, j)$ and (j, n). Whenever the sequence under consideration has width one, only one permutation is possible and the recursion ends. Any permutation that can be achieved by this procedure is a member of the neighborhood.

Just as a finite-state automaton can represent the Trans_n^* neighborhood, so can a context-free grammar represent BlockInsert_n^*. The top-down description of neighborhood member generation hints at this fact. The details of the grammar, and the size of the neighborhood, which requires introduction of normal form, will wait for later sections.

A bottom-up dynamic program based on the intuitions discussed so far is straightforward. Let $\Delta^*_{(i,k)}$ be the change in score of the best rearrangement of the (i,k) subsequence of π. Each $\Delta^*_{(k-1,k)}$ is zero, because $(k-1,k)$ consists of a single item that cannot be rearranged on its own. For all $(i,k) \in \binom{n+1}{2}$ such that $k-i \geq 2$,

$$\Delta^*_{(i,k)} = \max_{i<j<k} \left(\Delta^*_{(i,j)} + \Delta^*_{(j,k)} + \max\left(0,\ \Delta(i,j,k)\right) \right), \tag{2.20}$$

where $\Delta(i,j,k)$ is as in (2.15) and can be computed according to its own dynamic program, as in (2.18).

To derive the VLSN version of Insert_n, Insert_n^*, restrict the grammar that derives BlockInsert_n^*. In the top-down description, rather than choosing $j \in \binom{n-1}{1}$, restrict j to $\{1, n-1\}$. Then one of the two sub-constituents always has width one and the recursive procedure applies only to the other. This generalizes simply to a VLSN version of $\text{Insert}_n^{\leq w}$ as well.

Finding the best neighbor in BlockInsert_n^* is $\Theta(n^3)$ for two reasons:

- The dynamic program (2.20) computes $\Theta(n^2)$ quantities, each of which is a maximization over $\Theta(n)$ quantities, and

- There are $\Theta(n^3)$ block transposition scores, each computed in constant time using the dynamic program (2.18).

Restricting j in the dynamic program of (2.20) also leads to dynamic programs for searching these restricted VLSNs. Restricting j to $\{i+1, k-1\}$ means each computation in (2.20) is a maximization over a constant number of items, reducing that to $\Theta(n^2)$.

What is less obvious is that the runtime of the second part reduces as well. Because the dynamic program of (2.18) only refers to *narrower* Δs to compute $\Delta(i,j,k)$, if (i,j,k) satisfies the restrictions on j then all of the other quantities will also be available. As a result, finding the best neighbor in Insert_n^* is $\Theta(n^2)$, and finding the best neighbor in the VLSN version of $\text{Insert}_n^{\leq w}$ is $\Theta(wn^2)$.

Figure 2.4 summarizes the neighborhoods this section describes by placing them into a hierarchy. The next section treats the linear ordering problem as a search problem and invokes the neighborhoods of this section for iterated greedy local search.

$$\begin{tikzcd}
 & \mathrm{BlockInsert}_n^* & \\
\mathrm{BlockInsert}_n \arrow[ur] & & \mathrm{Insert}_n^{\leq w*} \arrow[ul] \\
\mathrm{Insert}_n^{\leq w} \arrow[u] \arrow[urr] & & \mathrm{Insert}_n^* \arrow[u] \\
\mathrm{Insert}_n \arrow[u] \arrow[urr] & \mathrm{Interchange}_n \arrow[ur] & \mathrm{Trans}_n^* \arrow[u] \\
 & \mathrm{Trans}_n \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.4: A Hasse diagram relating permutation neighborhoods. A directed edge $\mathcal{N}_1 \to \mathcal{N}_2$ from neighborhood $\mathcal{N}_1$ to neighborhood $\mathcal{N}_2$ indicates that $\mathcal{N}_1 \subseteq \mathcal{N}_2$.

2.5 Search

Because the linear ordering problem is NP-hard, there is no polynomial time algorithm for finding a solution, in general, unless P = NP.[4] The literature on the LOP includes many proposals of exact solution methods that can be efficient in practice for relatively small problems. Ultimately, as the problem size grows larger, these methods will fail, and the only recourse will be inexact algorithms. This section primarily concerns itself with those.

In particular, this section describes two basic approaches to search for the linear ordering problem. The first builds a candidate permutation from the ground up, one index at a time—*constructive search*. The second starts with a complete candidate permutation, possibly chosen at random, and makes a sequence of small updates, each of which results in another complete permutation. When the update is within the confines of a neighborhood, this is *local search*.

2.5.1 Constructive Search

The best known example of a constructive approach to the linear ordering problem is the greedy procedure of Becker (1967). This procedure builds a permutation π from left to right according to the following rule:

$$\pi_1 = \arg\max_{\ell \in \binom{n}{1}} \frac{\sum_{r=1}^{n} B[\ell, r]}{\sum_{r=1}^{n} B[r, \ell]}, \tag{2.21}$$

where $\binom{n}{1}$ indicates the set $\{1, 2, \ldots, n\}$ as in (2.8). $B[\ell, \ell]$ is assumed to be zero for all ℓ. The procedure arrives at the successive elements of π by eliminating π_1 from the matrix and following the same procedure recursively.

This rule is potentially problematic because nothing guarantees that the denominator is not zero, or negative, in practice. Cohen et al. (1999) proposed a very similar algorithm which uses an additive decision rule instead:

$$\pi_1 = \arg\max_{\ell \in \binom{n}{1}} \sum_{r=1}^{n} B[\ell, r] - \sum_{r=1}^{n} B[r, \ell]. \tag{2.22}$$

Although they went to the trouble of proving that their algorithm is within a factor of two of optimal, its performance is dismal on the examples from XLOLIB250. This chapter does not make use of these procedures further, but mentions them for completeness. While they are of little use alone, they might make good initializers for other methods, including greedy local search.

[4]This dissertation assumes that P $\neq$ NP and does not address the question further.

2.5.2 Greedy Local Search

Greedy local search, given a neighborhood $\mathcal{N}$, is trivial to describe:

1. Begin with an arbitrary permutation $\pi^{(0)}$—perhaps the identity permutation or one chosen at random.

2. At each time $t = 1, 2, \ldots$, set $\pi^{(t)}$ to the best permutation in the neighborhood $\mathcal{N}(\pi^{(t-1)})$.[5]

3. Repeat until no better permutation is available. If this happens at time $T + 1$, then $\pi^{(T)}$ is a *local maximum* of the neighborhood $\mathcal{N}$.

Many *metaheuristic* search procedures, such as iterated local search (ILS) or memetic algorithms, invoke greedy local search as a subroutine. Some of these are described in more detail in Section 2.7. This chapter uses only the simplest of these—random restarts—for benchmarking the neighborhoods.

2.5.3 Local Maxima

Each of the neighborhoods introduced in Sections 2.2 and 2.4 is compatible with greedy local search. Generally speaking, the larger neighborhoods have fewer local maxima. For example, BlockInsert$_n$ has fewer local maxima than Insert$_n$, which in turn has fewer than Trans$_n$.

The relationship between Insert$_n$ and Interchange$_n$ is of particular interest. Interchange$_n$ is not a subset of Insert$_n$, nor even of BlockInsert$_n$. However, its local maxima are a superset of those of Insert$_n$. To see this, let $\pi^{i \leftrightarrow j}$ be the permutation that results from interchanging π_i and π_j, and consider the relative score:

$$B(\pi^{i \leftrightarrow j}) - B(\pi) = \sum_{r=i+1}^{j} \Delta[\pi_i, \pi_r] + \sum_{\ell=i+1}^{j-1} \Delta[\pi_\ell, \pi_j]. \tag{2.23}$$

The two terms of the score are the same as the relative score of inserting π_i immediately after π_j and the score of inserting π_j immediately after π_i, respectively. For this score difference to be greater than zero, at least one of the terms must be positive, which implies that there exists an improving insertion. Therefore, if there are no improving insertions—a local maximum for Insert$_n$—then there are also no improving interchanges.

An important fact about the very large-scale neighborhoods of Section 2.4, is that they do not have fewer local maxima than their simple counterparts. For example, if a permutation π is a local maximum for Trans$_n$, then it is also a local maximum of Trans$_n^*$. This fact is easy to miss, especially considering the enormous size differences

[5]In general there may be ties. Any tie-breaking strategy is acceptable.

among these neighborhoods, which Table 2.5 on page 71 shows. The reason for this fact is that each of the VLSNs consists of *independent* applications of changes that are compatible with the simple neighborhoods. Trans_n^* can make several simultaneous transpositions, but the same set of changes could be made using successive Trans_n neighbors. If no single transposition is available to Trans_n, then Trans_n^* is stuck as well.

The same argument applies, with a bit more complexity, to the Insert_n^* and BlockInsert_n^* neighborhoods. If there is any positive $\Delta(i,j,k)$, then both BlockInsert_n and BlockInsert_n^* contain a better neighbor. If there is no positive $\Delta(i,j,k)$, then both neighborhoods are at a local maximum.

This is not the end of the story, however. While the VLSNs have the same local maxima, they may be less susceptible to arriving at bad local maxima, because they can "see" further than their simple counterparts. The simplest possible example occurs with the following LOP matrix:

$$B = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & - & 0 & 1 & 1 \\ 2 & 2 & - & 0 & 1 \\ 3 & 0 & 3 & - & 0 \\ 4 & 0 & 0 & 2 & - \end{array}.$$

Starting from the permutation $\pi^{(0)} = 1\ 2\ 3\ 4$, both Trans_4 and Trans_4^* have the following scores:

$$\begin{aligned} \Delta[1,2] &= B[2,1] - B[1,2] = 2, \\ \Delta[2,3] &= B[3,2] - B[2,3] = 3, \text{ and} \\ \Delta[3,4] &= B[4,3] - B[3,4] = 2. \end{aligned}$$

The greedy choice for Trans_4 is 1 3 2 4, while the greedy choice for Trans_4^* is 2 1 4 3. Both are local maxima, but the one that Trans_4^* finds is better.

This is not to say that the VLSNs will always find better solutions. If we make a small change to B,

$$B' = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & - & 0 & 1 & 1 \\ 2 & 2 & - & 0 & \mathbf{0} \\ 3 & 0 & 3 & - & 0 \\ 4 & 0 & \mathbf{2} & 2 & - \end{array},$$

then 1 3 2 4 is no longer a local maximum, and Trans_4 will move on to the better permutation 1 3 4 2, which has a higher score than the maximum where Trans_4^* is still stuck. It is an empirical question which case is more common in practice. Section 2.6.2 addresses this question.

2.5.4 Shortcuts

Schiavinotto and Stützle (2004) proposed a shortcut version of the Insert_n neighborhood that they called $\mathcal{LS}_f$. The procedure, in the notation of this chapter, is the following:

1. Let k run from 1 to n.
2. For each k, search Insert_n^k, the set of permutations achieved by moving π_k.
3. If Insert_n^k contains an improvement over the current permutation, immediately return it and do not consider larger k.

That is, $\mathcal{LS}_f$ searches in the neighborhood Insert_n^1 first. It moves to the best permutation in that neighborhood if one improves on the current permutation. If not, search moves on to the Insert_n^2 neighborhood. It continues in this manner until Insert_n^n, and if no improvement is found, it stops.

$\mathcal{LS}_f$ has the same local maxima as the Insert_n neighborhood. When it reaches a local maximum, it searches the entire Insert_n neighborhood, requiring $\Theta(n^2)$ time, before stopping. However, if there are successive improvements available involving the successive first few elements of $\pi^{(t)}$, then many of the steps of $\mathcal{LS}_f$ may take only $\Theta(n)$ time. Also, the shortcut behavior means $\mathcal{LS}_f$ settles, in general, for a neighbor that is not the best neighbor in Insert_n. Nonetheless, Schiavinotto and Stützle found that it performed the best of several options they considered under their metaheuristic search procedures.

The following, novel, search procedure, called Block $\mathcal{LS}_f$, builds on the procedure of $\mathcal{LS}_f$:

1. Let w run from 1 to $\left\lfloor \frac{n}{2} \right\rfloor$.
2. Let i run from 0 to $n-1$.
3. If $\text{Insert}_n^{(i,i+w)}$ contains any improvements, return the best one immediately.

The first iteration of the outer loop, when $w = 1$, is identical to $\mathcal{LS}_f$. If the current permutation is a local maximum for Insert_n, then Block $\mathcal{LS}_f$ moves on to block transpositions.

Figure 2.5 shows pseudocode for a more general width-limited case of Block $\mathcal{LS}_f$. Line 12 decrements ℓ from $i-1$ down to zero, while line 15 inrements r from $j+1$ up to n, to ensure that the Δ quantities get computed in the right order, as discussed in Section 2.3.3.

Block $\mathcal{LS}_f$, as described, has the same local maxima as search in the BlockInsert_n neighborhood, and requires $\Theta(n^3)$ time for the last step. The next section will also consider variants with a maximum width. If the maximum width is $O(1)$ then every step of the width-limited Block $\mathcal{LS}_f$ is $O(n^2)$, like $\mathcal{LS}_f$.

Considering blocks of width w or more is only beneficial in the following situation:

```
procedure BlockLSf($B, \pi, \hat{w}$)
    $n \leftarrow |\pi|$
    for $i \leftarrow 0$ to $n-1$ do
        for $k \leftarrow i+1$ to $n$ do
            $\Delta[i,i,k] \leftarrow 0$
            $\Delta[i,k,k] \leftarrow 0$
        end for
    end for
    for $w \leftarrow 1$ to $\hat{w}$ do
        for $i \leftarrow 0$ to $n-w$ do
            $k \leftarrow i+w$
            for $\ell \leftarrow i-1$ to $0$ do
                $\Delta[\ell,i,k] \leftarrow$ Compute($B, \pi, \Delta, \ell, i, k$)
            end for
            for $r \leftarrow k+1$ to $n$ do
                $\Delta[i,k,r] \leftarrow$ Compute($B, \pi, \Delta, i, k, r$)
            end for
            $\hat{\ell} \leftarrow \arg\max_{\ell} \Delta[\ell,i,k]$
            $\hat{r} \leftarrow \arg\max_{r} \Delta[i,k,r]$
            if $\max(\hat{\ell}, \hat{r}) > 0$ then
                if $\hat{\ell} \geq \hat{r}$ then
                    return Transpose($\pi, \hat{\ell}, i, k$)
                else
                    return Transpose($\pi, i, k, \hat{r}$)
                end if
            end if
        end for
    end for
    return $\pi$
end procedure

function Compute($B, \pi, \Delta, i, j, k$)
    return $\Delta[i+1,j,k] + \Delta[i,j,k-1] - \Delta[i+1,j,k-1] + B[\pi_k, \pi_{i+1}] - B[\pi_{i+1}, \pi_k]$
end function
```

Figure 2.5: Pseudocode for Block $\mathcal{LS}_f$, including a width-limit parameter. For full Block $\mathcal{LS}_f$, $\hat{w}$ is set to $\left\lfloor \frac{n}{2} \right\rfloor$.

- Two blocks of size at least w increase the score when transposed.
- No improvement is possible by transposing blocks such that one is smaller than w.

As w increases, the likelihood that this situation occurs becomes smaller and smaller, as $\Theta(w^2)$ matrix entries are involved. Therefore, there should be diminishing returns from searching up to larger and larger w. The next section shows that this is indeed the case for the problems in XLOLIB250.

2.6 Benchmarks

Greedy local search procedures differ along several dimensions. In general, all of the following will vary:

- the time to run each iteration,
- the number of iterations to reach a local maximum, and
- the local maxima.

This section proposes a search quality comparison that accounts for all these differences. The goal of search for the LOP is presumably to find the highest scoring permutation possible. The comparison considers the trajectory of the score of the best permutation found so far versus the clock time required to find it. In order to reduce variance, the comparison starts each search procedure from many random permutations, and randomly permutes those random starting points many times, producing an average best score trajectory to plot versus clock time.

The procedure for each search method and for each of the 49 XLOLIB250 instances is as follows:

1. From each of 100 random starting permutations, run the local search algorithm until a local maximum is reached. Record the score of the local maximum and the total clock time required for search to stop.
2. For each of 1000 random permutations of the 100 starting points, compute the best score achieved after each multiple of some interval, e.g. one second. This *simulates* many sequences of random restarts, but merely reuses the statistics from the one sequence of 100.
3. Plot the average best score at each time interval.

Call $\pi^{(t)}$ the best permutation found by a search method by time t, including as many random restarts as necessary. Ideally, computation of the averages in step 3 would range over all possible sequences of random restarts. That would give the exact

Run	**Score**	**Iterations**	**Time (s)**	**Total (s)**
1	9.04149e+06	1151	1.03	1.03
2	8.99885e+06	1072	1.01	2.04
3	8.9842e+06	1142	1.06	3.10
4	9.06643e+06	1171	0.97	4.07
5	9.03331e+06	1128	1.02	5.09
6	9.07546e+06	1126	0.95	6.04
7	9.0733e+06	1070	0.88	6.92
8	9.00814e+06	876	0.84	7.76
9	9.06367e+06	1286	1.18	8.94
10	8.97501e+06	957	0.90	9.84

Table 2.1: Example benchmark data from one XLOLIB250 instance.

Time (s)	**Best Score**
1	0
2	9.04149e+06
3	9.04149e+06
4	9.04149e+06
5	9.06643e+06
6	9.06643e+06
7	9.07546e+06
8	9.07546e+06
9	9.07546e+06
10	9.07546e+06

Table 2.2: Best score vs. time interval for the data from Table 2.1.

Name	**Description**
block	Block $\mathcal{LS}_f$ with no maximum width
block w	Block $\mathcal{LS}_f$ with maximum width w
lsf	$\mathcal{LS}_f \equiv$ Block $\mathcal{LS}_f$ with maximum width 1

Table 2.3: Benchmark search algorithms

expectation of $B(\pi^{(t)})$. Unfortunately, the number of sequences of 100 random restarts is $250!^{100}$. The plots that result show an approximation to the true expectation.

As an example of steps 1 and 2, Table 2.1 shows ten random restarts on the first XLOLIB250 instance for one local search method. Using the identity permutation of the random restarts, Table 2.2 shows the best score achieved so far at intervals of one second.

2.6.1 Results

Figures 2.6–2.12 show performance of several local search algorithms on all 49 problems from XLOLIB250. The systems in question are described in Table 2.3.

Each figure was generated according to the search quality procedure described above, using 1000 randomly generated permutations of 100 random restarts for greedy local search.

The results demonstrate that, while Block $\mathcal{LS}_f$ necessarily requires more time than $\mathcal{LS}_f$ to reach a local maximum from a given starting permutation, Block $\mathcal{LS}_f$ also frequently finds better local maxima than $\mathcal{LS}_f$, and the improved maxima are found far more quickly than the alternative of starting $\mathcal{LS}_f$ from a new random permutation. There are five qualitatively different types of benchmark graphs:

1. By far the most common type has each of the width-limited Block $\mathcal{LS}_f$ curves dominating the red Block $\mathcal{LS}_f$ curve, which in turn dominates the magenta $\mathcal{LS}_f$ curve. The first graph in Figure 2.6 is a good example.

2. The next most common type differs from the first only in the position of the yellow Block $\mathcal{LS}_f$ $w = 2$ curve, which falls below the red Block $\mathcal{LS}_f$ curve. The fourth graph in Figure 2.8 is a good example.

3. Less frequently, the green Block $\mathcal{LS}_f$ $w = 3$ curve also falls below the red Block $\mathcal{LS}_f$ curve. The bottom left graph of Figure 2.10 is the best example.

4. The magenta $\mathcal{LS}_f$ curve occasionally crosses the red Block $\mathcal{LS}_f$ curve. This happens in the sixth graph of Figure 2.6.

5. Finally, twice, the $\mathcal{LS}_f$ curve also crosses the width-limited Block $\mathcal{LS}_f$ curves—the fifth graph in Figure 2.8 and the second graph in Figure 2.11.

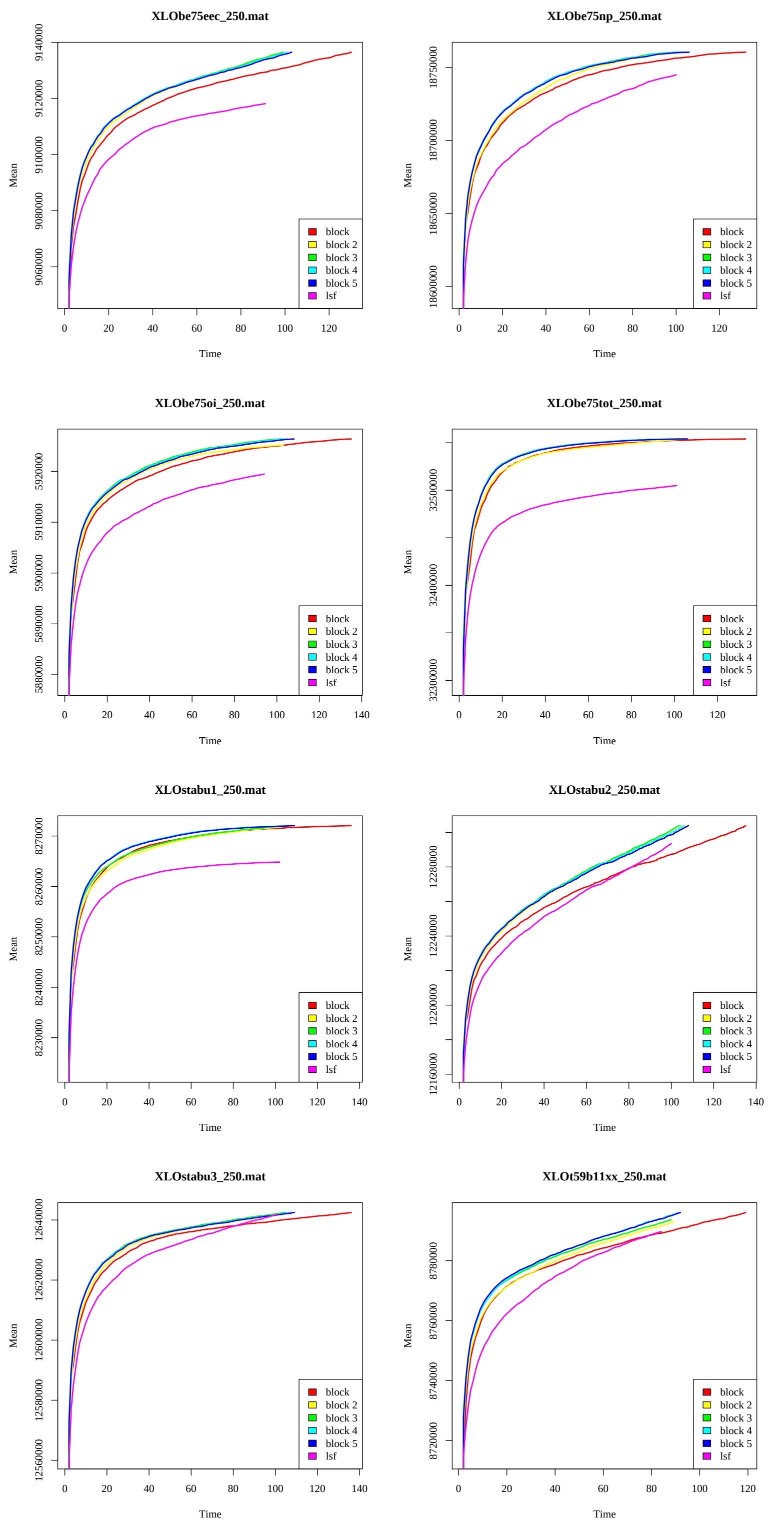

Figure 2.6: XLOLIB250 search quality results, part 1.

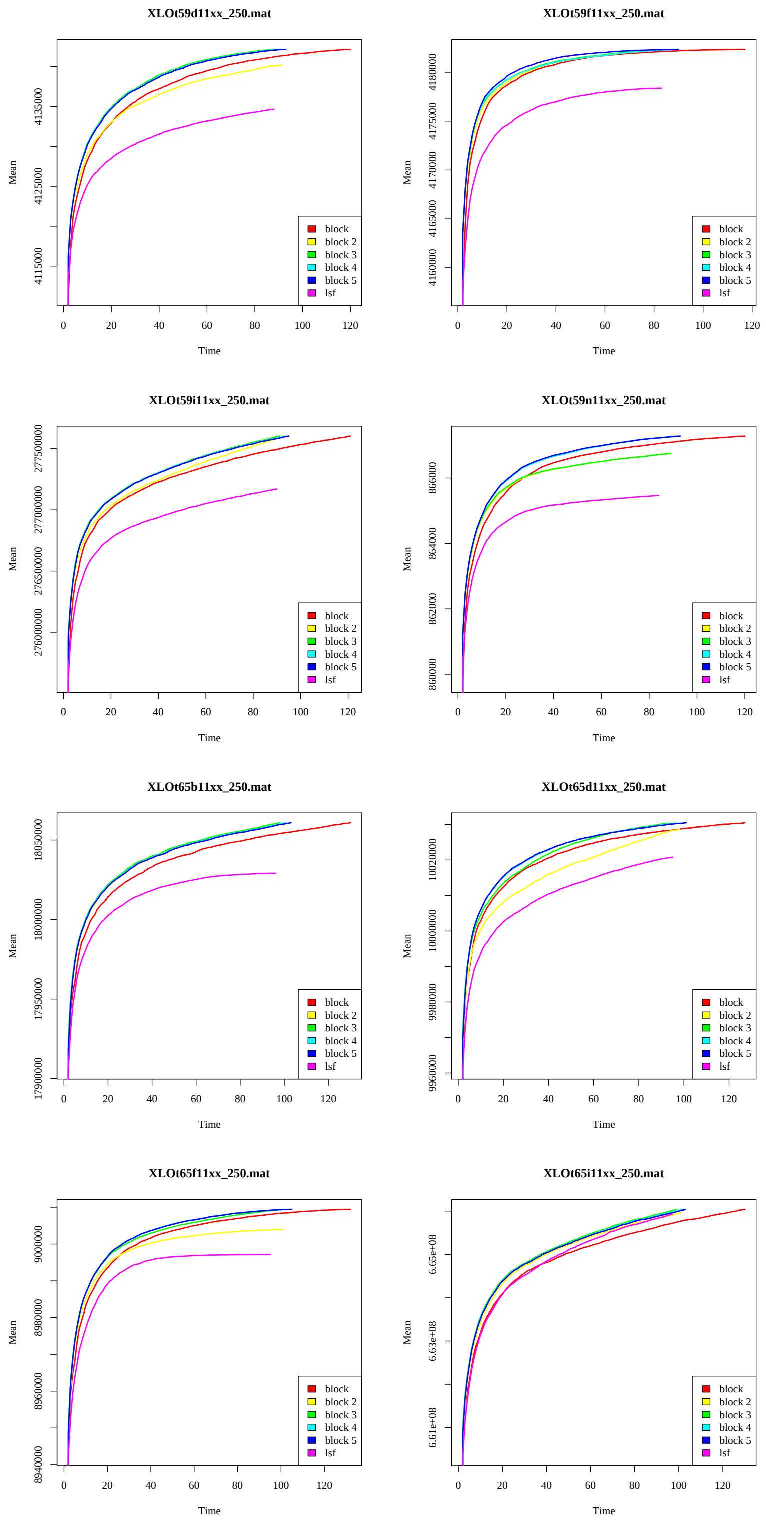

Figure 2.7: XLOLIB250 search quality results, part 2.

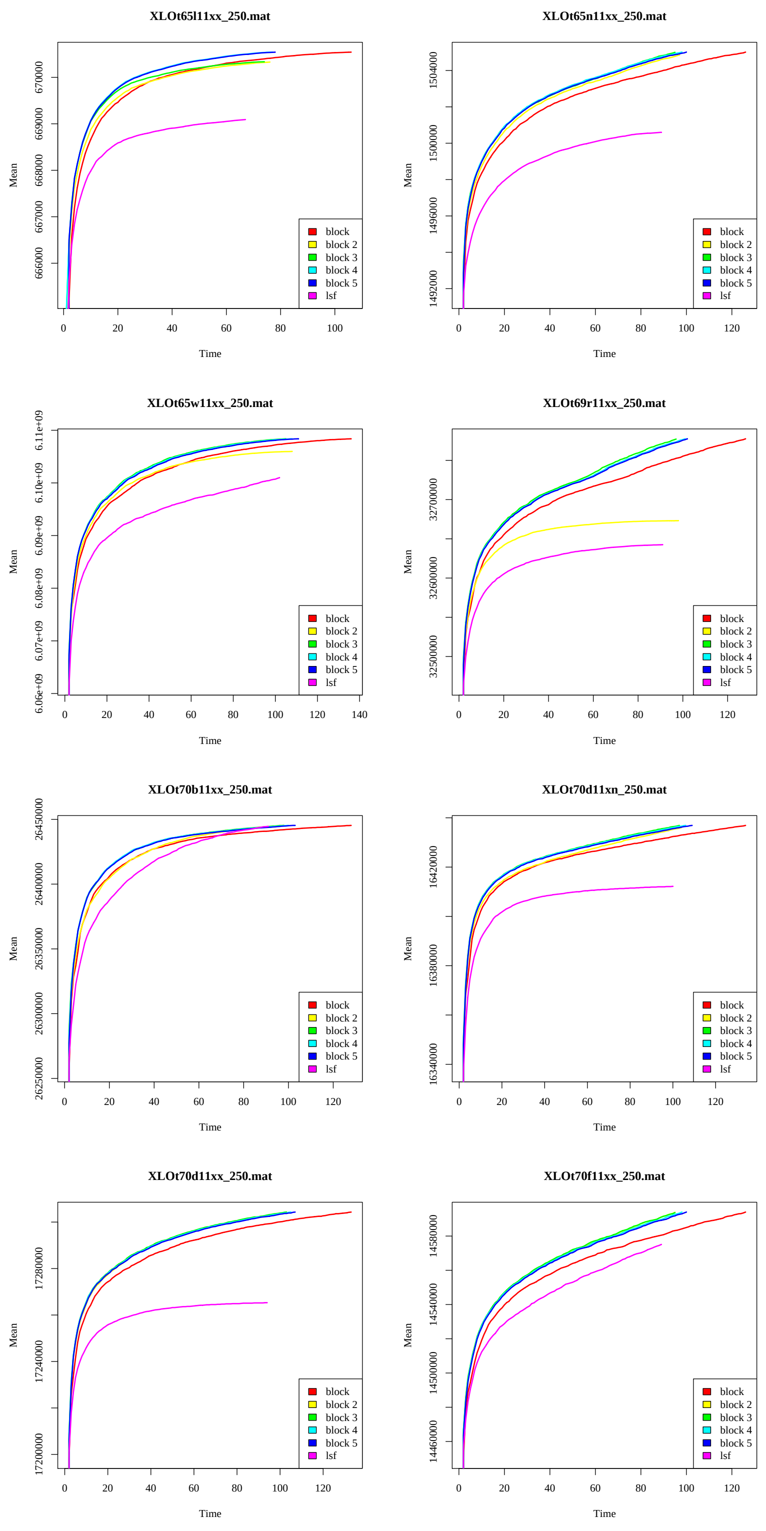

Figure 2.8: XLOLIB250 search quality results, part 3.

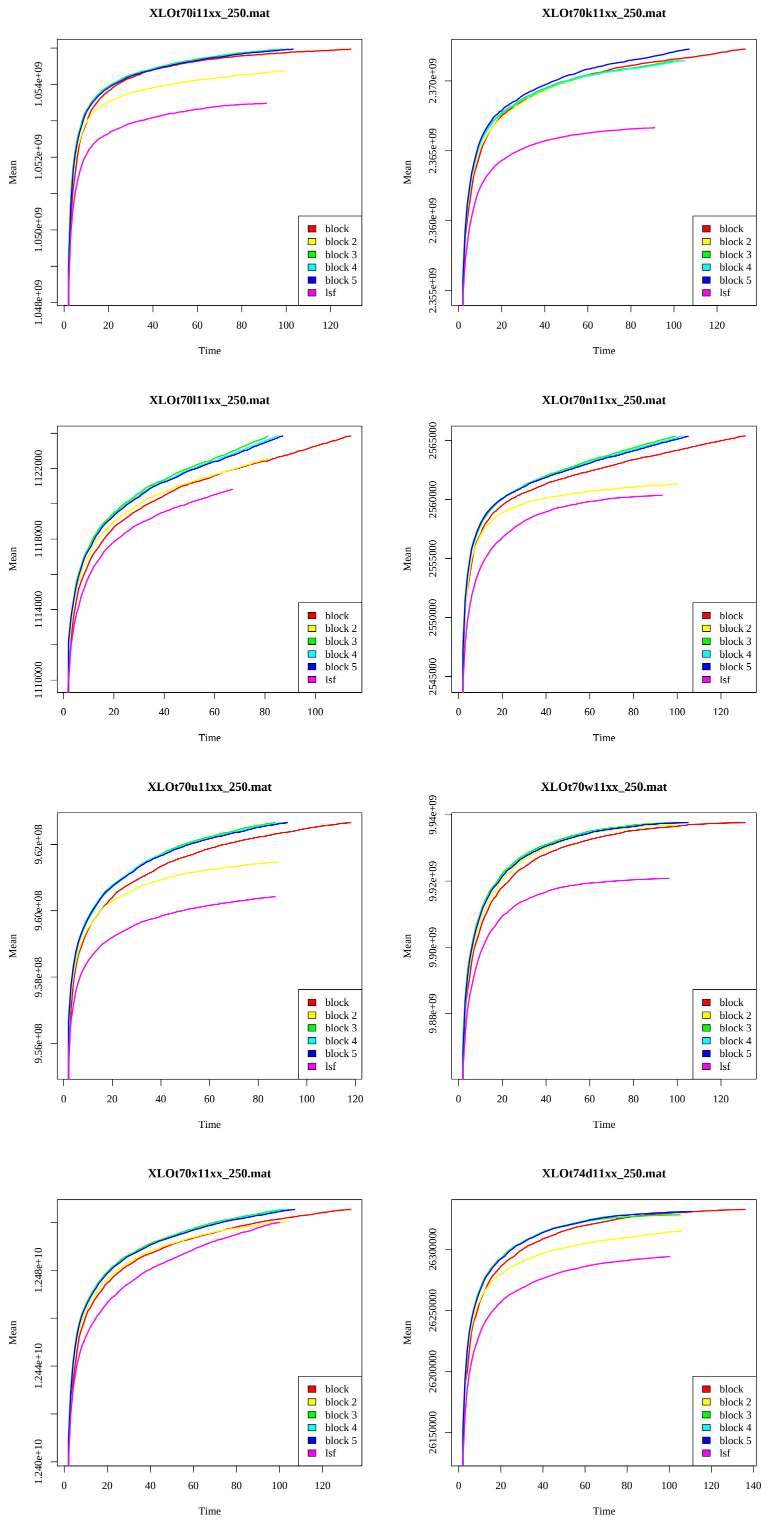

Figure 2.9: XLOLIB250 search quality results, part 4.

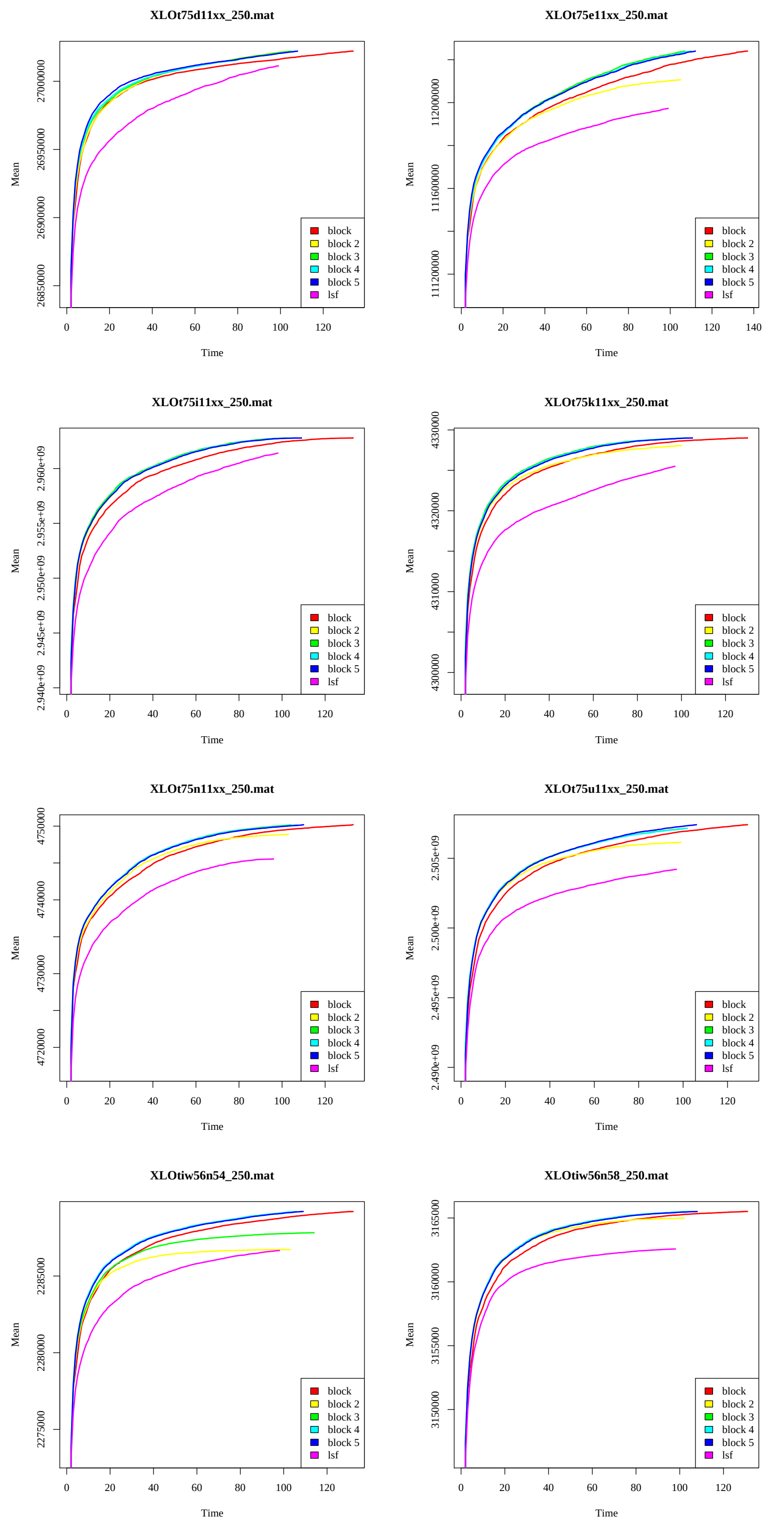

Figure 2.10: XLOLIB250 search quality results, part 5.

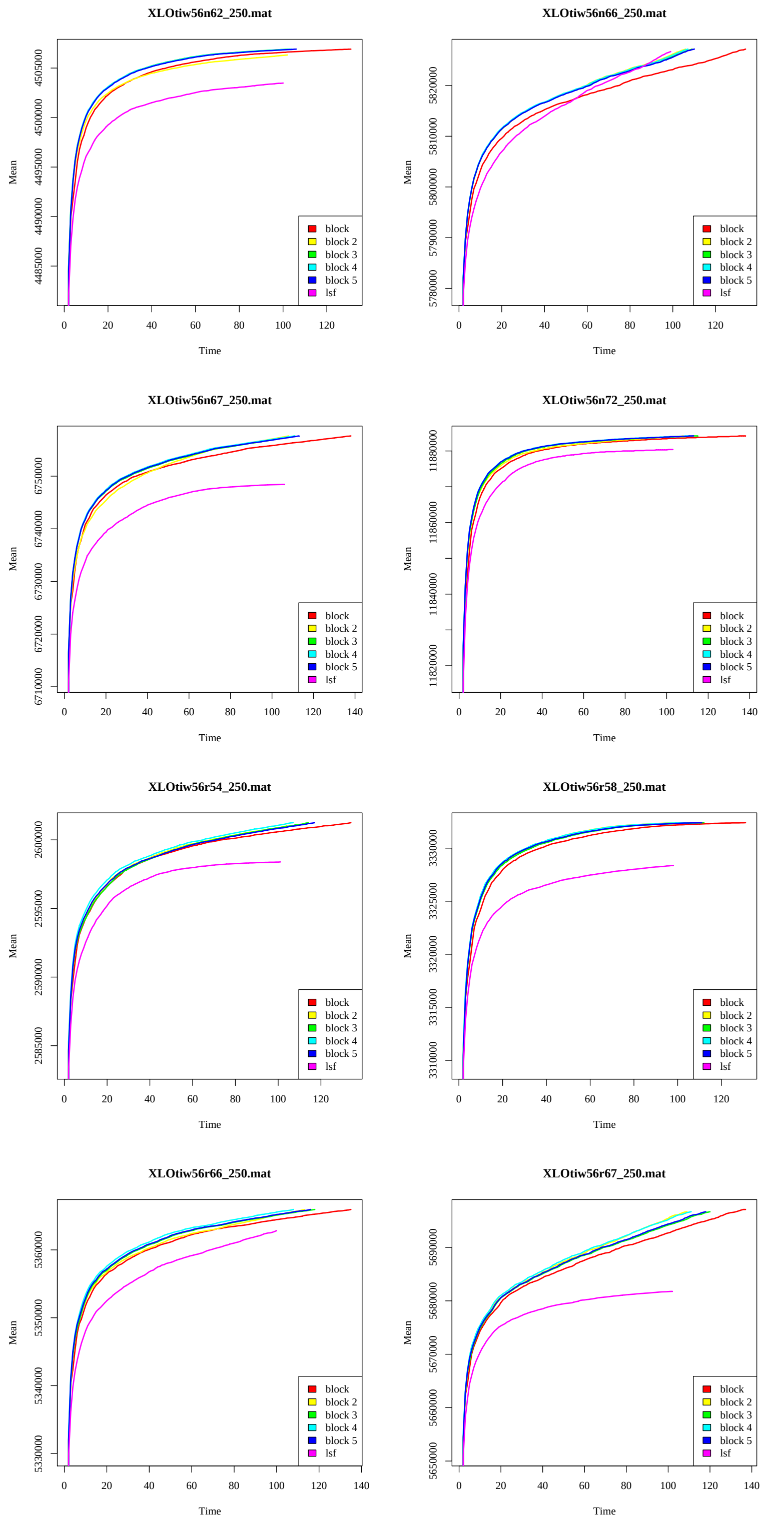

Figure 2.11: XLOLIB250 search quality results, part 6.

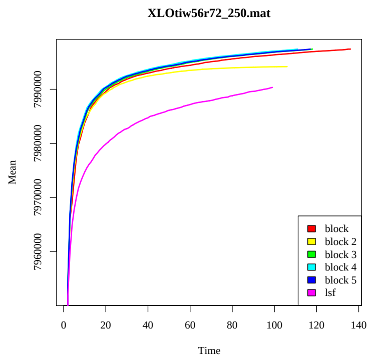

Figure 2.12: XLOLIB250 search quality results, part 7.

Figures 2.13–2.19 show similar search quality graphs comparing $\mathcal{LS}_f$ to VLSN search with the Insert$_{250}^{\leq 3*}$ neighborhood, which showed the best performance among VLSNs. These graphs include error bars of a sort, showing a single standard deviation for both curves. Most of the time the differences between the two methods are within one standard deviation. In rare cases, one clearly outperforms the other.

2.6.2 Discussion

Why doesn't local search with very large-scale neighborhoods work better? For one thing, they don't have different local maxima than their simple counterparts, as Section 2.5.2 explains. This is not entirely convincing, though. Insert$_n^*$ has the same local maxima as $\mathcal{LS}_f$, but Insert$_n^{\leq 3*}$, to which $\mathcal{LS}_f$ is compared, has fewer. This VLSN can take advantage of all the same improvements as Block $\mathcal{LS}_f$ $w = 3$.

A more important reason may be that the VLSNs don't take shortcuts. Every step with Insert$_n^*$, or its wider-block variants, is $\Theta(n^2)$, and every step with BlockInsert$_n^*$ is $\Theta(n^3)$. The good news is that these steps can move many items at once, completely reversing the permutation, for example, if it is advantageous. The bad news is that they can't make a few small changes relatively quickly, like $\mathcal{LS}_f$.

This is a confusing point, however, because Block $\mathcal{LS}_f$ suffers from the same problem any time it makes a change that $\mathcal{LS}_f$ can't also make—it is $\Theta(n^2)$ to discover that $\mathcal{LS}_f$ is stuck. Therefore, it seems safe to conclude that much of the advantage of Block $\mathcal{LS}_f$ over VLSN search comes from its hybridization with $\mathcal{LS}_f$, which makes some changes in $\Theta(n)$ time. It is also possible to hybridize VLSN search with $\mathcal{LS}_f$, running $\mathcal{LS}_f$ to a local maximum and then invoking the very large neighborhood to try to jump out of it. However, it is not clear that the VLSNs offer any advantage over Block $\mathcal{LS}_f$ in this respect, because they have the same local maxima.

Another possible explanation for the disappointing performance of VLSN search is that the vast majority of the neighbors in the neighborhood are relatively far

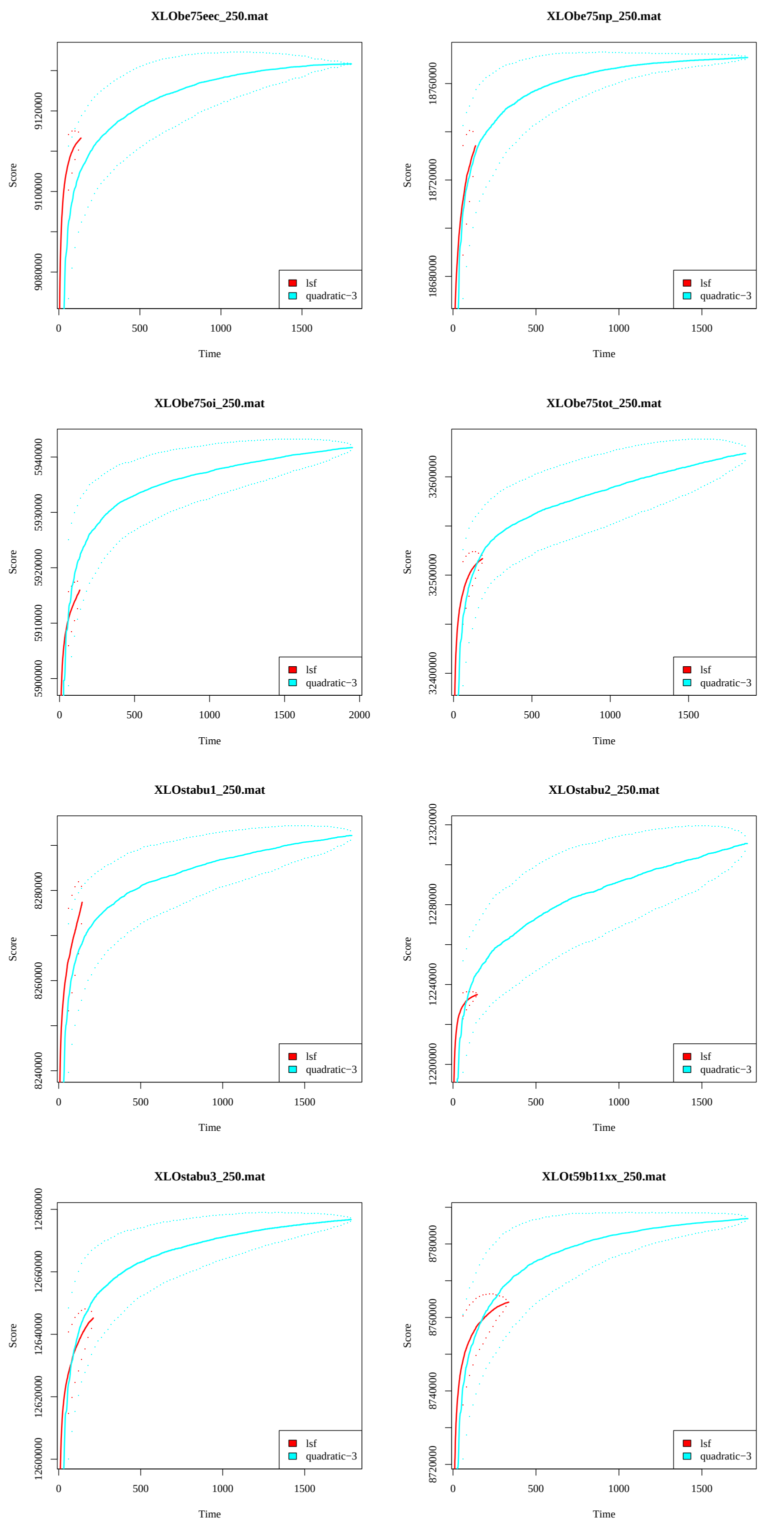

Figure 2.13: XLOLIB250 VLSN search quality results, part 1.

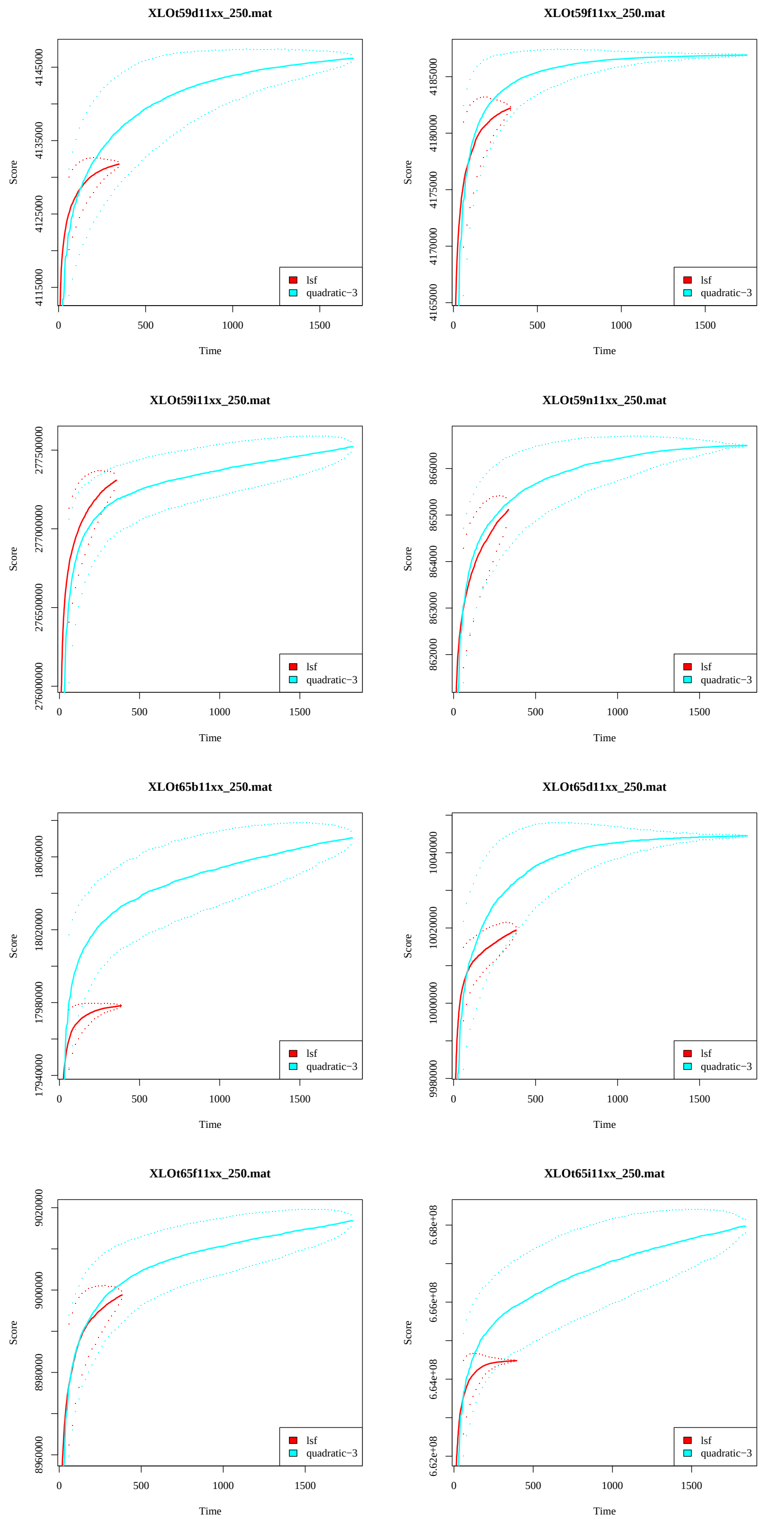

Figure 2.14: XLOLIB250 VLSN search quality results, part 2.

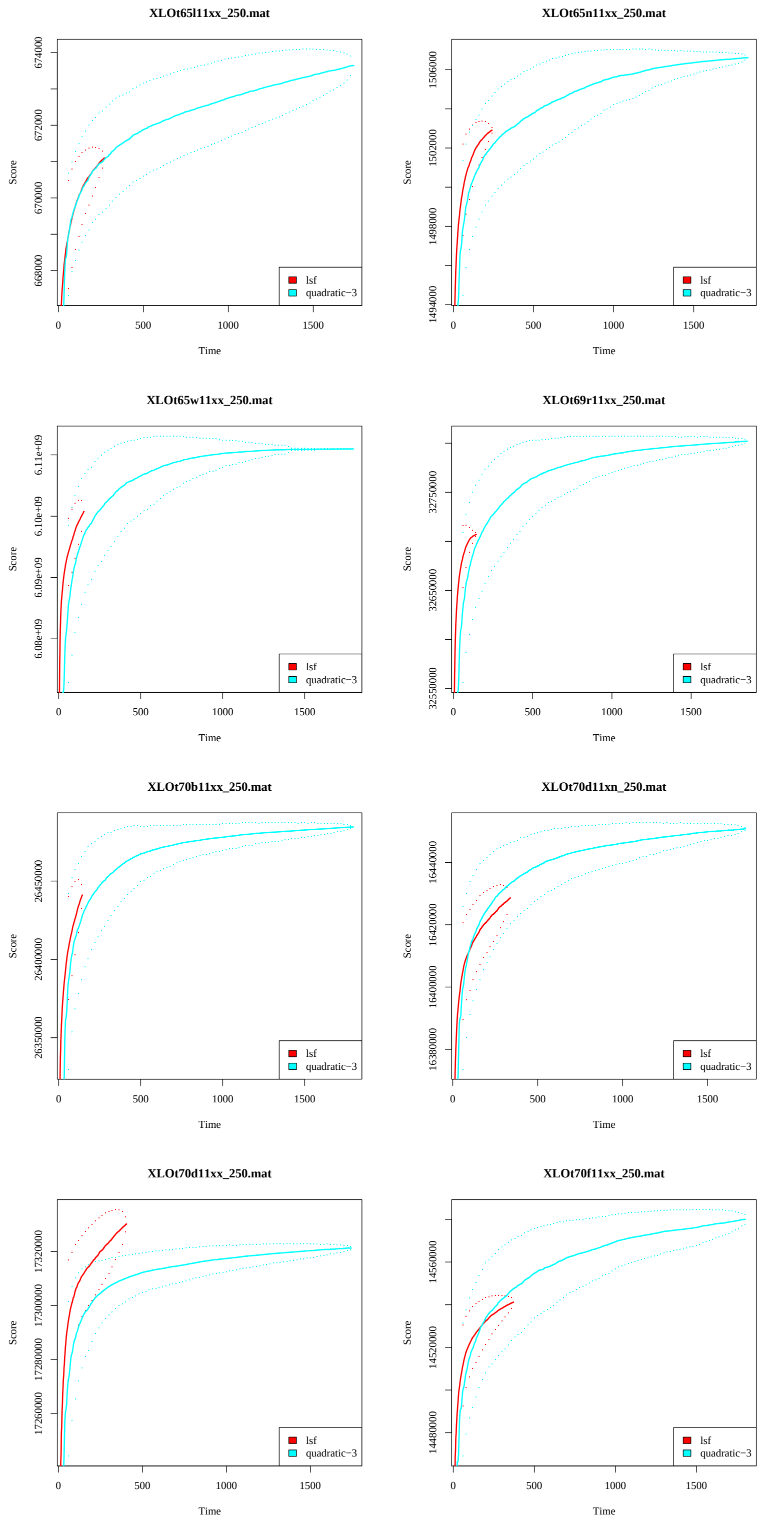

Figure 2.15: XLOLIB250 VLSN search quality results, part 3.

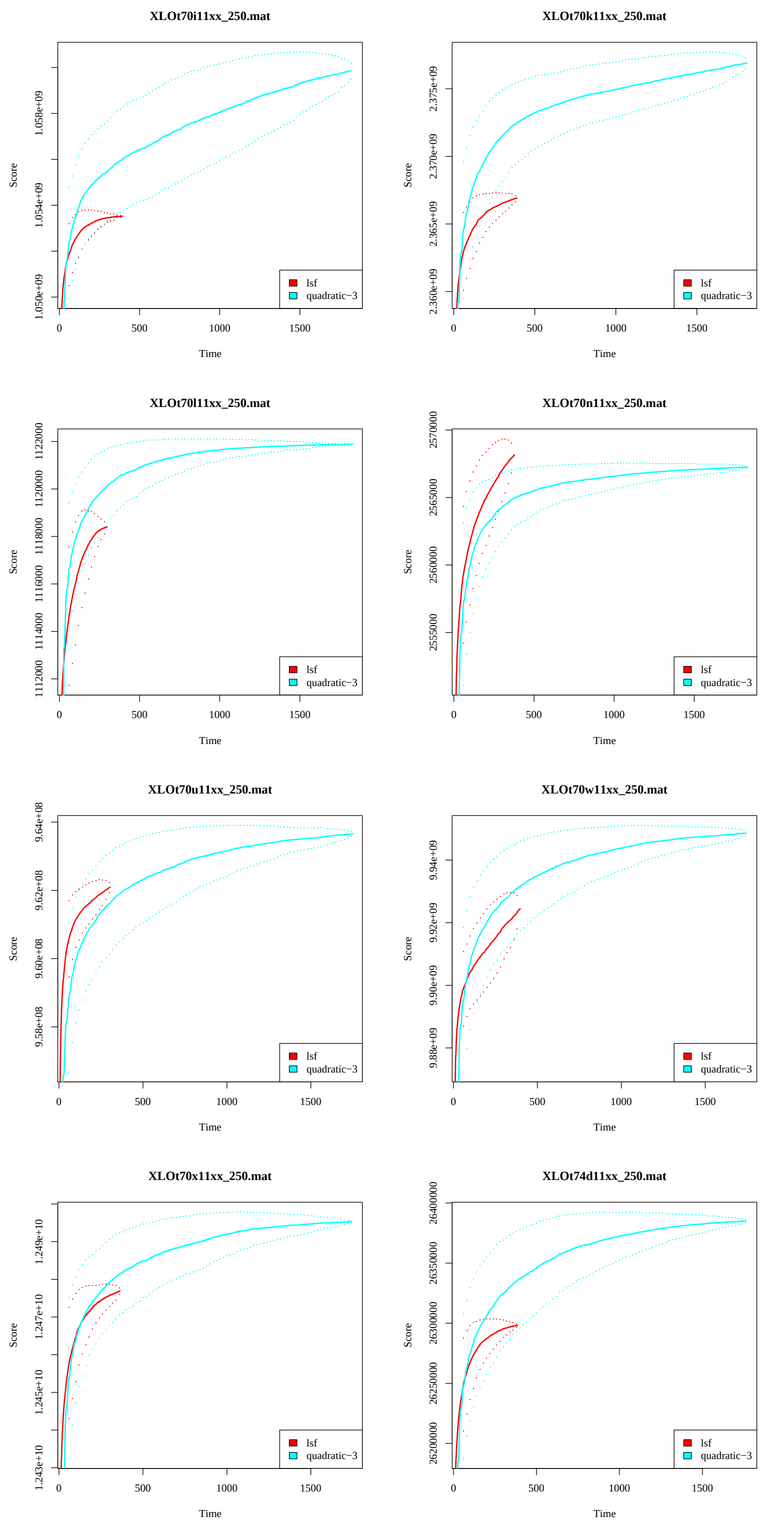

Figure 2.16: XLOLIB250 VLSN search quality results, part 4.

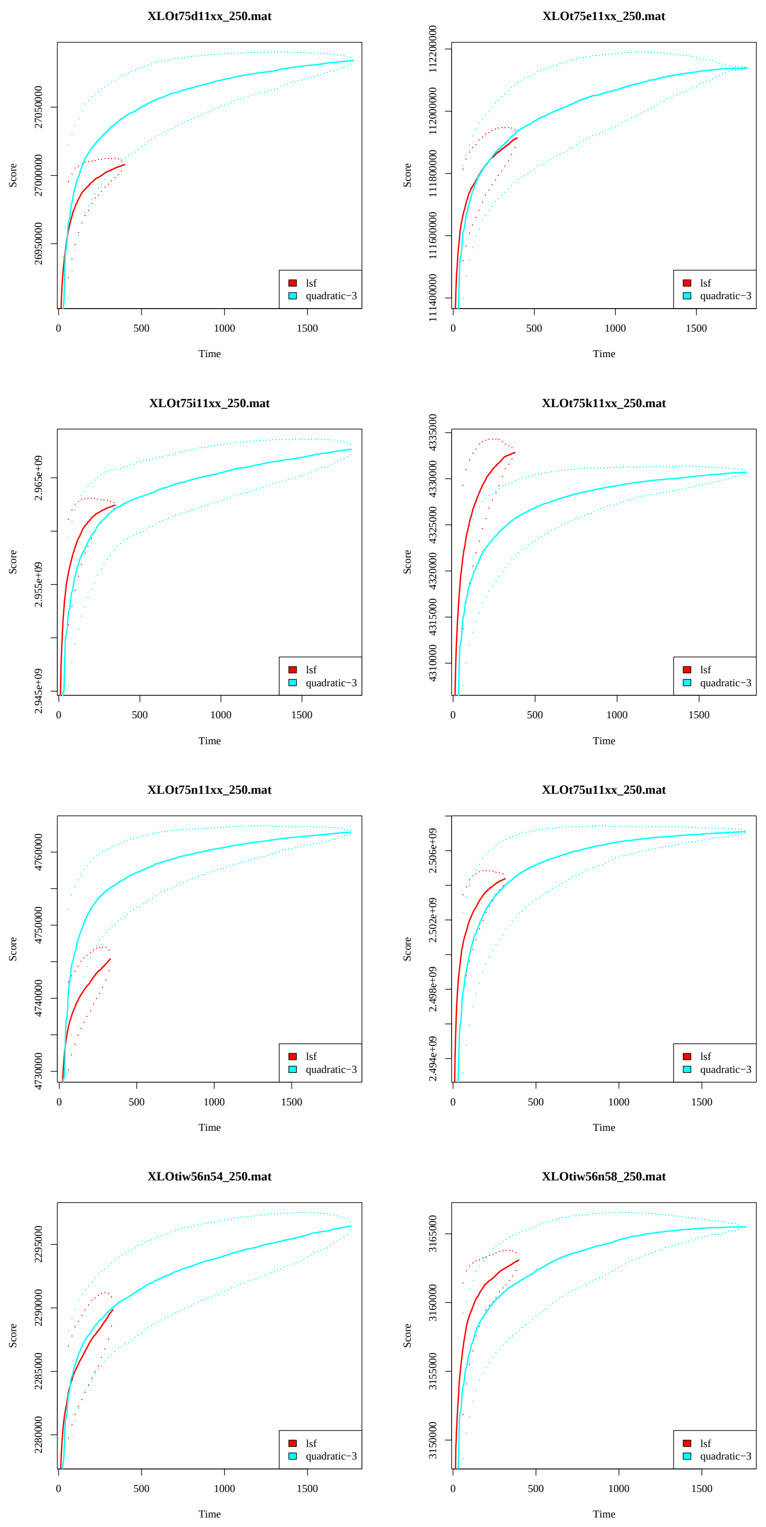

Figure 2.17: XLOLIB250 VLSN search quality results, part 5.

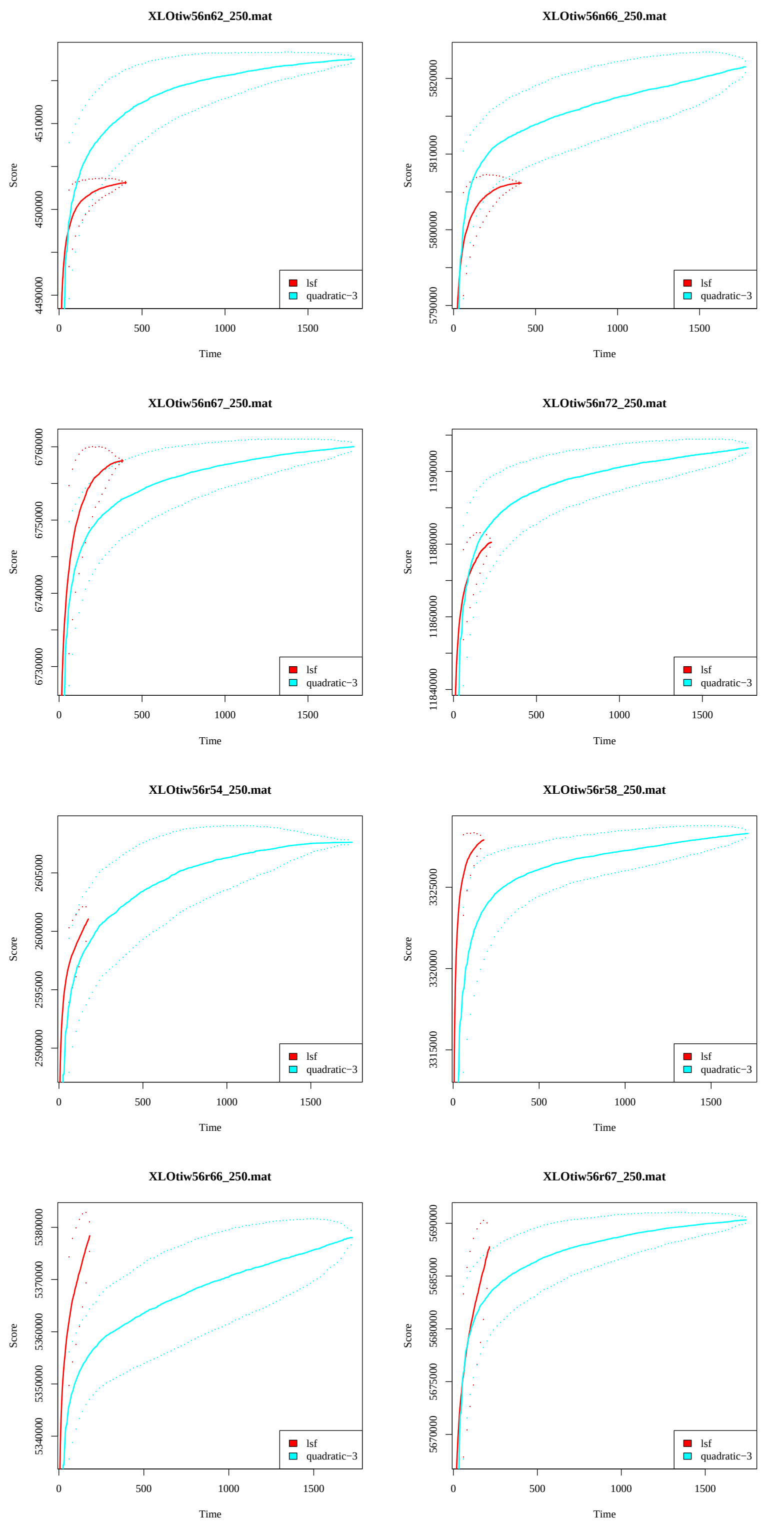

Figure 2.18: XLOLIB250 VLSN search quality results, part 6.

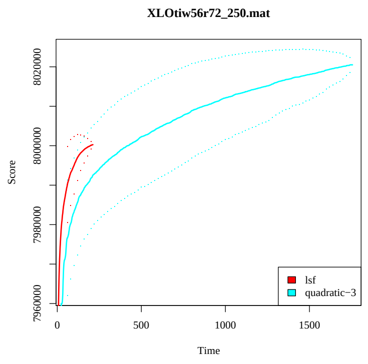

Figure 2.19: XLOLIB250 VLSN search quality results, part 7.

away from the current permutation in terms of the number of transposed pairs—the symmetric difference distance of Section 2.11.2—, so if the current permutation is relatively good, most of the permutations in the very large-scale neighborhood will be bad.

A final suggestion is that $\mathcal{LS}_f$ gains its advantage by improving the current permutation quickly, and then moving on to another random starting point. Trying lots of random restarts is better than spending a long time looking for modifications to just one. Block $\mathcal{LS}_f$ outperforms $\mathcal{LS}_f$ because it can often find further changes, still relatively quickly. The VLSNs fall behind because the further improvements they make take too long.

To conclude, Block $\mathcal{LS}_f$ seems a clear improvement over a state-of-the-art local search method. Although it is somewhat more complicated to implement than $\mathcal{LS}_f$, this should hardly be a deterrent to its application, given its superior performance on these benchmarks.

2.7 Related Work

2.7.1 Exact Methods

Grötschel et al. (1984) used integer linear programming to solve the LOP. They combined cutting planes, based on their analysis of the 0/1-polytope associated with the LOP, with branch and bound, and used the simplex method to solve successive linear program relaxations. They reported successful solutions of input-output matrix problems up to size $n = 60$.

Mitchell and Borchers (1996) also used cutting planes to solve the LOP with linear programming. Instead of the simplex method, they used the primal-dual interior point method. They reported results on problems up to size $n = 79$.

Charon and Hudry (2006) used a sophisticated application of branch-and-bound to solve the LOP. They applied the Lagrangian relaxation of Arditti (1984) as an evaluation function to bound the quality of partial solutions. They solved problems of size as large as $n = 100$.

These results show that exact solutions to the LOP are possible for many practical applications. As processor speeds continue to increase and these methods become increasingly sophisticated, yet more LOP instances will fall into this class. Still, the difficulty of the LOP guarantees that there will always be instances that defeat these methods, so the study of inexact methods will remain important.

2.7.2 Precedents

Körte and Oberhofer (1971) introduced the notion of *relatively optimal ranking*, and an ordering relation based upon it. A triple of indices (i, j, k) is a relatively optimal ranking if the block (i, j) has a higher score before the block (j, k) than after it. This work introduced two search procedures—one exact, the other approximate—based on this notion. The exact procedure considers all permutations in lexicographic order, but shortcuts search as soon as a partial permutation contains any blocks that are not relatively optimal. The approximate method is essentially to choose the best move in the BlockInsert_n neighborhood. Körte and Oberhofer (1971) do have dynamic programs for computing some quantities, but come up short of constant time per block transposition, which Section 2.3.3 provides. This may be why the block transpositions have not seen further use in search for the linear ordering problem until now.

The VLSN counterpart of BlockInsert_n, BlockInsert_n^*, is the same as the twisted tree neighborhoods of Deĭneko and Woeginger (2000). The novel contribution of this chapter is the dynamic program to compute the scores (2.18), which reduces the asymptotic runtime from the $\Theta(n^5)$ of Bompadre and Orlin (2005) to $\Theta(n^3)$.

2.7.3 Metaheuristics

While this chapter measures LOP search performance using random restarts for initialization, Schiavinotto and Stützle (2004) used memetic algorithms for their search. Memetic algorithms are a variant of genetic algorithms. In addition to the usual mutation and cross-over operations of GAs, MAs have a local search operation. Whenever a new member of the population is created—a permutation in the case of MAs for the LOP—local search is performed to climb to a local maximum.

An important question is what neighborhood would perform best as the local search component of such memetic algorithms. Block $\mathcal{LS}_f$ may not continue to outperform $\mathcal{LS}_f$ when the permutations used for initialization are not random, but generated according to a memetic algorithm, where mutations and cross-over are expected to produce reasonably good starting permutations, given that they operate on existing permutations that are already local maxima.

Nareyek (2003) explored a variety of heuristic updates for weighting local search strategies. This approach could be adapted to the hybrid search setting, where many different neighborhoods are available with different size/time tradeoffs. Instead of choosing the neighborhood to search in a fixed hierarchical order, it could decide the next neighborhood probabilistically and then reward or punish the method chosen based on some trade-off between the quality of the improvement it proposed and the amount of time it took to find that improvement. One of the search options would be to restart at a random permutation, rather than continuing search from the current one.

2.7.4 Stochastic Search

Search need not always be greedy. The neighborhoods of this chapter are also compatible with stochastic search methods. Simulated annealing (Martin and Otto, 1994) is a good example, though Johnson and McGeoch (1997) found that hillclimbing outperformed it for the traveling salesman problem. Simulated annealing would take a random walk in the permutation space, sampling the next permutation from the neighborhood of each permutation it visits, while gradually sharpening the probability distrubtion towards its mode.

This requires several capabilities, including computing the total unnormalized weight of all permutations in the neighborhood, and sampling from the neighborhood. The methods of this chapter and Chapter 4 provide all of these tools. Chapter 4 will introduce a probability distribution over permutations and use it for various learning procedures in Section 4.11. These methods require the normal forms of Section 2.9 to avoid counting permutations multiple times. The next section works toward this goal by introducing grammars.

2.8 Grammars

In order to specify neighborhoods precisely and unambiguously, this section introduces context-free grammars. It will be convenient to think of grammars as generating sets of permutations given an initial permutation π. That is, the language represented by the CFG corresponding to neighborhood $\mathcal{N}$ is the set of permutations $\mathcal{N}(\pi)$.

Binary permutation trees will represent derivations from these grammars. These permutation trees will use two types of internal nodes, as illustrated in Figure 2.20—*in-order* nodes and *reverse* nodes. In-order nodes generate their left child subpermutations before their right child subpermutations, while reverse nodes generate their right children first.

Because the display of a reverse node shows the left child on the left, the permutation displayed at the leaves of the tree is the same regardless of the tree—Figures 2.22, 2.25, and 2.28 show example permutation trees and indicate the implied permutations.

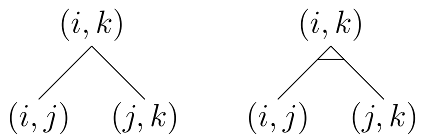

Figure 2.20: Internal nodes of binary permutation trees. The unmarked node, on the left, represents a derivation of a non-terminal spanning (i,k) that generates the span (i,j) before the span (j,k). The marked node, on the right, represents the opposite—a non-terminal spanning (i,k) that generates (j,k) first, then (i,j).

$$\begin{aligned}
S_i &\rightarrow \pi_i\ S_{i+1} \\
S_i &\rightarrow \pi_{i+1}\ \pi_i\ S_{i+2},\ \forall i \in (0, n-2) \\
\\
S_{n-1} &\rightarrow \pi_{n-1}\ S_n \\
S_{n-1} &\rightarrow \pi_n\ \pi_{n-1} \\
S_n &\rightarrow \pi_n
\end{aligned}$$

Figure 2.21: A grammar for $\text{Trans}_n^*(\pi)$.

The next few sections give grammars for each of the very large-scale neighborhoods of Section 2.4.

2.8.1 A Grammar for Transpositions

The first grammar formally describes the Trans_n^* neighborhood by generating $\text{Trans}_n^*(\pi)$. Let S_1 be the start symbol. Then the grammar is as shown in Figure 2.21.

There are special rules for S_{n-1} and S_n because the sequence ends at n. (Defining $S_{n+1} \rightarrow \epsilon$, would obviate the special rule for S_{n-1}.) Figure 2.22 shows an example derivation.

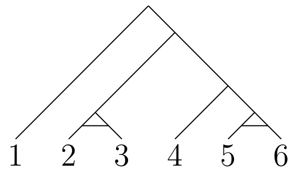

Figure 2.22: An example derivation from the transpositions grammar of Figure 2.21. The tree is drawn as though the grammar had been binarized by replacing the rule $S_i \rightarrow \pi_{i+1}\ \pi_i\ S_{i+2}$ with two rules: $S_i \rightarrow R_i\ S_{i+2}$ and $R_i \rightarrow \pi_{i+1}\ \pi_i$. The tree shows a derivation of the permutation 1 3 2 4 6 5 in the neighborhood $\text{Trans}^*(1\ 2\ 3\ 4\ 5\ 6)$.

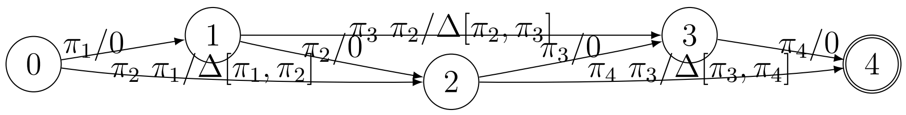
0
1
2
3
4
$\pi_1/0$
$\pi_2\ \pi_1/\Delta[\pi_1, \pi_2]$
$\pi_2/0$
$\pi_3\ \pi_2/\Delta[\pi_2, \pi_3]$
$\pi_3/0$
$\pi_4\ \pi_3/\Delta[\pi_3, \pi_4]$
$\pi_4/0$

Figure 2.23: A weighted FSA representing the Trans_4^* neighborhood.

$$\begin{array}{rcl} S_{i,k} & \rightarrow & S_{i,j}\ S_{j,k} \\ S_{i,k} & \rightarrow & S_{j,k}\ S_{i,j},\ \forall (i,j,k) \in \binom{n+1}{3} \\ \\ S_{i-1,i} & \rightarrow & \pi_i,\ \forall i \in (0,n) \end{array}$$

Figure 2.24: A grammar for $\mathrm{BlockInsert}_n^*(\pi)$.

This grammar, as given, is unweighted. Computing the highest-scoring neighbor requires adding weights to the grammar. Because computing $B(\pi)$ from scratch is $\Theta(n^2)$, the grammar will compute relative scores, as in Section 2.3.1, to keep search $\Theta(n)$. Therefore, the weight of each permutation $\pi' \in \mathrm{Trans}_n^*(\pi)$ will be $B(\pi') - B(\pi)$.

The weight of each context-free rule that transposes a pair of neighboring items π_i and π_{i+1} is $\Delta[\pi_i, \pi_{i+1}]$ as defined in (2.16). The weight of any other rule is zero.

The grammar of Figure 2.21 is not just context-free, but regular, meaning the neighborhood can also be represented as a regular expression, or as a finite-state automaton. Figure 2.23 shows a weighted finite-state acceptor for the neighborhood $\mathrm{Trans}_4^*(\pi)$.

Section 2.10 will point to the absence of spurious ambiguity in this grammar, and use the first pair of rules to derive the recurrence relation for the size of the neighborhood, and the latter rules to provide the base case.

2.8.2 A Grammar for Block Insertions

Moving on to a grammar for $\mathrm{BlockInsert}_n^*(\pi)$, let $S_{0,n}$ be the start symbol, and create a non-terminal $S_{i,k}$ for all $\binom{n+1}{2}$ pairs of positions such the $0 \leq i < k \leq n$. The base cases—pre-terminals—of the grammar are $S_{i-1,i}$. The grammar is given in Figure 2.24, which uses $\binom{n+1}{3}$ as shorthand for the set $\{(i,j,k) \mid 0 \leq i < j < k \leq n\}$ as in (2.8). Figure 2.25 shows an example derivation under this grammar.

Because the size of this grammar is $\Theta(n^3)$, it is efficient to compute the actual

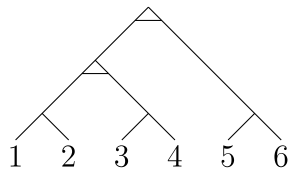

Figure 2.25: An example derivation from the block insertions grammar of Figure 2.24. The tree shows a derivation of the permutation 5 6 3 4 1 2 in the neighborhood BlockInsert$^*_6$(1 2 3 4 5 6).

$$\begin{aligned}
\overrightarrow{\gamma}_{i,i,k} &= 0 \\
\overrightarrow{\gamma}_{i,k,k} &= 0 \\
\overrightarrow{\gamma}_{i,j,k} &= \overrightarrow{\gamma}_{i,j,k-1} + \overrightarrow{\gamma}_{i+1,j,k} - \overrightarrow{\gamma}_{i+1,j,k-1} + B[\pi_{i+1}, \pi_k] \\
\\
\overleftarrow{\gamma}_{i,i,k} &= 0 \\
\overleftarrow{\gamma}_{i,k,k} &= 0 \\
\overleftarrow{\gamma}_{i,j,k} &= \overleftarrow{\gamma}_{i,j,k-1} + \overleftarrow{\gamma}_{i+1,j,k} - \overleftarrow{\gamma}_{i+1,j,k-1} + B[\pi_k, \pi_{i+1}]
\end{aligned}$$

Figure 2.26: A dynamic program for computing weights of grammar rules. $\overrightarrow{\gamma}_{i,j,k}$ is the weight of any rule, such as $S_{i,k} \to S_{i,j}\ S_{j,k}$, that generates (i,j) and (j,k) in order. $\overleftarrow{\gamma}_{i,j,k}$ is the weight of any rule, such as $S_{i,k} \to S_{j,k}\ S_{i,j}$, that generates (j,k) before (i,j). Preterminals, such as $S_{i-1,i} \to \pi_i$, always have weight 0.

score of each permutation $\pi' \in \text{BlockInsert}_n^*(\pi)$. The grammar weights are as follows:

$$\overrightarrow{\gamma}_{i,j,k} \overset{\text{def}}{=} \sum_{\ell=i+1}^{j} \sum_{r=j+1}^{k} B[\pi_\ell, \pi_r] \tag{2.24}$$

$$\overleftarrow{\gamma}_{i,j,k} \overset{\text{def}}{=} \sum_{\ell=i+1}^{j} \sum_{r=j+1}^{k} B[\pi_r, \pi_\ell] \tag{2.25}$$

Figure 2.26 gives a dynamic program for computing these weights efficiently and describes the rules to which they apply. Like the grammar, this dynamic program has $\Theta(n^3)$ items, each of which can be computed from earlier items in constant time.

Theorem 2.1 *Let T be a binary permutation tree expressing the permutation π. If each in-order node in T combining (i, j) and (j, k) to span (i, k) receives the score (2.24), and each reverse node in T combining (i, j) and (j, k) to span (i, k) receives the score (2.25), then the sum of the scores of all nodes in the tree is $B(\pi)$.*

Proof: The proof is by induction on the length of the permutation, n. It assumes that the total grammar score is correct for all permutations of length $m < n$, and shows that it is also correct for permutations of length n.

For the base case, $n = 1$, $B(\pi)$ is always zero. Likewise, the total grammar cost is zero, because the trivial tree has no interior nodes.

For the induction case, consider the node spanning $(0, n)$ that combines two subtrees spanning $(0, k)$ and (k, n), for some $0 < k < n$. First, observe that $(0, k)$ and (k, n) both have length less than n, so the induction hypothesis implies that the scores of the two sub-permutations equal the sums of the grammar scores in the two subtrees.

Now, every pair of items $\ell \prec r$ in π falls into one of three cases:

1. Both ℓ and r are in the left span $(0, k)$,
2. Both ℓ and r are in the right span (k, n), or
3. Exactly one of ℓ and r is in the left span, and the other is in the right span.

In the first case, $B[\ell, r]$ is included in the score of the left subtree, by the induction hypothesis. In the second case, $B[\ell, r]$ is included in the score of the right subtree, also by the induction hypothesis. Therefore, we need only consider those pairs split between the subtrees.

If the node spanning $(0, n)$ is in-order, then ℓ must fall in $(0, k)$ and r in (k, n) in order to achieve $\ell \prec r$. The grammar score is $\overrightarrow{\gamma}_{0,k,n}$, which includes the score $B[\ell, r]$.

If the node spanning $(0, n)$ reverses, then ℓ must fall in (k, n) and r in $(0, k)$ in order to achieve $\ell \prec r$. In this case, the grammar score is $\overleftarrow{\gamma}_{0,k,n}$, which includes the score $B[\ell, r]$.

Start symbol and base cases $S_{i-1,i}$ as in Figure 2.24, but

$$\begin{array}{lcl} S_{i,k} & \rightarrow & S_{i,i+1}\ S_{i+1,k} \\ S_{i,k} & \rightarrow & S_{i+1,k}\ S_{i,i+1} \\ S_{i,k} & \rightarrow & S_{i,k-1}\ S_{k-1,k} \\ S_{i,k} & \rightarrow & S_{k-1,k}\ S_{i,k-1} \end{array}$$

Or, equivalently,

$$\begin{array}{lcl} S_{i,k} & \rightarrow & \pi_{i+1}\ S_{i+1,k} \\ S_{i,k} & \rightarrow & S_{i+1,k}\ \pi_{i+1} \\ S_{i,k} & \rightarrow & S_{i,k-1}\ \pi_{k} \\ S_{i,k} & \rightarrow & \pi_{k}\ S_{i,k-1} \end{array}$$

Figure 2.27: Two equivalent grammars for $\text{Insert}_n^*(\pi)$.

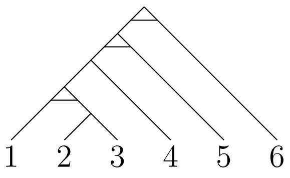

Figure 2.28: An example derivation from the insertions grammars of Figure 2.27. The tree shows a derivation of the permutation 6 5 2 3 1 4 in the neighborhood $\text{Insert}^*(1\ 2\ 3\ 4\ 5\ 6)$.

Further, these grammar scores for $(0, n)$ include *only* those B scores for ℓ and r in separate subtrees. This proves that the induction hypothesis holds for all $n \geq 1$. □

This grammar suffers from spurious ambiguity, which, along with its size, will be addressed in following sections.

2.8.3 A Grammar for Insertions

The $\text{Insert}_n^*(\pi)$ neighborhood can be generated by a subset of the previous grammar, where j is limited to the set $\{i + 1, k - 1\}$. See Figure 2.27. Note that when $k - i = 2$, the four rules only have two unique right-hand sides. Figure 2.28 shows an example permutation tree from these grammars.

The weights from Figure 2.26 apply to this grammar as well. Only those weights for which grammar rules exist need to be computed, meaning a size $\Theta(n^2)$ subset.

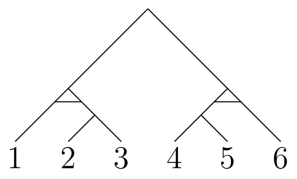

Figure 2.29: An example derivation from the insertions grammar with a special case. In addition to the rules of Figure 2.27, the grammar has rules of the form $S_{0,k} \rightarrow S_{0,j}\ S_{j,k}$ and $S_{0,k} \rightarrow S_{j,k}\ S_{0,j}$ for all $0 < j < k \leq n$. This tree shows a derivation of the permutation 2 3 1 6 4 5 in the neighborhood of 1 2 3 4 5 6. This permutation is not a member of Insert*(1 2 3 4 5 6).

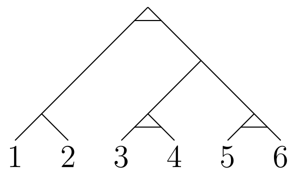

Figure 2.30: An example derivation from the insertions grammar with a different special case. In addition to the rules of Figure 2.27, the grammar has rules of the form $S_{i,n} \rightarrow S_{i,j}\ S_{j,n}$ and $S_{i,n} \rightarrow S_{j,n}\ S_{i,j}$ for all $0 \leq i < j < n$. This tree shows a derivation of the permutation 4 3 6 5 1 2 in the neighborhood of 1 2 3 4 5 6. This permutation is not a member of Insert*(1 2 3 4 5 6) nor of the neighborhood described in Figure 2.29.

Allowing j to take on additional values, e.g. $j \in \{i+1, \ldots, i+w, k-w, \ldots, k-1\}$, for some width w, where values outside $(i+1, k-1)$ are obviously disallowed when $k - i < 2w + 1$, results in the width-limited block insertion neighborhoods. The example tree in Figure 2.25 is compatible with a grammar of this type for $w = 2$.

Further, special cases can be allowed. For example, if $i = 0$, allow any $j \in (i+1, k-1)$, but otherwise limit j. Figure 2.29 shows an example permutation tree derived from a grammar of this sort. Figure 2.30 gives an example with $k = n$ as a special case.

2.8.4 Computation with Grammars

The important thing to understand is that, for computing with the grammar, the runtime is proportional to the number of rules in the grammar—the *grammar constant*, as it is called in the parsing community. All of these grammars have the property that the weights of the rules can be computed in constant time per rule using dynamic programming. Some will require modifications of the rules from Section 2.2. Those cases receive special mention when they arise.

For Trans$^*_n(\pi)$, the grammar has $2n - 1$ rules, so finding the best neighbor of π

$$
\begin{aligned}
\overrightarrow{\gamma}_{0,j,k} &= \overrightarrow{\gamma}_{0,j,k-1} + \overrightarrow{\gamma}_{0,j-1,k} - \overrightarrow{\gamma}_{0,j-1,k-1} + B[\pi_j, \pi_k] \\
\overleftarrow{\gamma}_{0,j,k} &= \overleftarrow{\gamma}_{0,j,k-1} + \overleftarrow{\gamma}_{0,j-1,k} - \overleftarrow{\gamma}_{0,j-1,k-1} + B[\pi_k, \pi_j]
\end{aligned}
$$

$$
\begin{aligned}
\overrightarrow{\gamma}_{i,j,n} &= \overrightarrow{\gamma}_{i+1,j,n} + \overrightarrow{\gamma}_{i,j+1,n} - \overrightarrow{\gamma}_{i+1,j+1,n} + B[\pi_{i+1}, \pi_{j+1}] \\
\overleftarrow{\gamma}_{i,j,n} &= \overleftarrow{\gamma}_{i+1,j,n} + \overleftarrow{\gamma}_{i,j+1,n} - \overleftarrow{\gamma}_{i+1,j+1,n} + B[\pi_{j+1}, \pi_{i+1}]
\end{aligned}
$$

Figure 2.31: Dynamic programs for special cases. The $(0, j, k)$ rules must be computed with j ranging from left to right, because $\gamma_{0,j,k}$ depends on $\gamma_{0,j-1,k}$. The (i, j, n) rules must be computed right to left, because $\gamma_{i,j,n}$ depends on $\gamma_{i,j+1,n}$.

Grammar	**Name**	**Size**
Transpositions (Figure 2.21)	G_T	$\Theta(n)$
Insertions (Figure 2.27)	G_I	$\Theta(n^2)$
Special case $i = 0$ (Figure 2.29)	$G_{i=0}$	$\Theta(n^2)$
Special case $k = n$ (Figure 2.30)	$G_{k=n}$	$\Theta(n^2)$
Insertions with blocks of size $\leq w$	$G_{B\leq w}$	$\Theta(wn^2)$
Block insertions (Figure 2.24)	G_B	$\Theta(n^3)$

Table 2.4: Number of rules in various grammars. The special case grammars $G_{i=0}$ and $G_{k=n}$, consisting only of the rules explicitly given in Figure 2.29 and Figure 2.30, respectively, are incomplete on their own. Figure 2.29 shows derivations from $G_I \cup G_{i=0}$ and Figure 2.30 from $G_I \cup G_{k=n}$. The size of the united grammars is bounded above by the sums of the sizes, so both remain $\Theta(n^2)$.

can be done in $\Theta(n)$ time.

For Insert$^*_n(\pi)$, there are $4\binom{n+1}{2} - 3n = \Theta(n^2)$ rules. Allowing $j \in \{i+1, \ldots, i+w, k-w, \ldots, k-1\}$ increases this to approximately $4w\binom{n+1}{2} = \Theta(wn^2)$. If w is a small constant, this is asymptotically the same as Insert$^*_n(\pi)$.

Allowing any j when $i = 0$ means $\Theta(n)$ rules for each non-terminal $S_{0,k}$ alone, but there are only $\Theta(n)$ of these non-terminals, making $\Theta(n^2)$ rules of this type. Likewise if $k = n$ is a special case. These rules can therefore be added to the Insert$^*_n(\pi)$ grammar without changing its asymptotic runtime.

These cases require different dynamic programming, however, because when considering $S_{0,k}$, for example, the corresponding weight of the $S_{1,k}$ rule may never have been computed. Figure 2.31 shows the new dynamic programs for the two cases $i = 0$ and $k = n$.

Finally, BlockInsert$^*_n(\pi)$ has $2\binom{n+1}{3} - n$ grammar rules, making it $\Theta(n^3)$. Table 2.4 summarizes this information and names the grammars.

$$\begin{aligned}&\gamma(S_{i,m} \rightarrow S_{j,k}\ S_{\ell,m}\ S_{i,j}\ S_{k,\ell})\\ &\quad= \underbrace{\overleftarrow{\gamma}_{i,j,m} - \overleftarrow{\gamma}_{i,j,\ell} + \overleftarrow{\gamma}_{i,j,k}}_{S_{j,k}\prec S_{i,j},\ S_{\ell,m}\prec S_{i,j}} + \underbrace{\overrightarrow{\gamma}_{i,j,\ell} - \overrightarrow{\gamma}_{i,j,k}}_{S_{i,j}\prec S_{k,\ell}} + \underbrace{\overrightarrow{\gamma}_{j,k,m}}_{S_{j,k}\prec S_{k,\ell},\ S_{j,k}\prec S_{\ell,m}} + \underbrace{\overleftarrow{\gamma}_{k,\ell,m}}_{S_{\ell,m}\prec S_{k,\ell}}\end{aligned}$$

Figure 2.32: A dynamic program for computing the weight of an inside-out grammar rule. This is significantly more complex than the rules in Figure 2.26, but is also constant time per rule, given the needed γ values.

2.8.5 Grammars for More Complex Neighborhoods

Most of these grammars, and more, were covered in Bompadre and Orlin (2005), though without the dynamic programs for efficiently computing grammar weights. From there it is easy to see how to avoid, for example, the "inside-out" permutations from the ITG literature—2 4 1 3 and 3 1 4 2—by introducing rules of the form $S_{i,m} \rightarrow S_{j,k}S_{\ell,m}S_{i,j}S_{k,\ell}$, and its reverse, but at the cost of a grammar with $\Theta(n^5)$ rules, since there are $\binom{n+1}{5}$ quintuples (i,j,k,ℓ,m) for which to form such rules.

Illustration of such permutations unfortunately cannot be accomplished with binary permutation trees, because the derivation rules necessarily have more than two non-terminals on their right-hand sides.

Computing the weights of such rules is also necessarily more complicated, though it can still be done in constant time per rule given the weights of the rules from the BlockInsert$^*(\pi)$ grammar. See Figure 2.32.[6]

The differences between the scores of these inside-out rules and the rule that keeps the four subsequences in order shows how these rules eliminate some of the local maxima of the BlockInsert$^*_n$ neighborhood.

$$\begin{aligned}&\gamma(S_{i,m} \rightarrow S_{j,k}\ S_{\ell,m}\ S_{i,j}\ S_{k,\ell}) - \gamma(S_{i,m} \rightarrow S_{i,j}\ S_{j,k}\ S_{k,\ell}\ S_{\ell,m}) && (2.26)\\ &\qquad = \Delta(i,j,m) - \Delta(i,j,\ell) + \Delta(i,j,k) + \Delta(k,\ell,m), \text{ and}\\ &\gamma(S_{i,m} \rightarrow S_{k,\ell}\ S_{i,j}\ S_{\ell,m}\ S_{j,k}) - \gamma(S_{i,m} \rightarrow S_{i,j}\ S_{j,k}\ S_{k,\ell}\ S_{\ell,m}) && (2.27)\\ &\qquad = \Delta(i,k,\ell) + \Delta(j,k,m) - \Delta(j,k,\ell).\end{aligned}$$

The presence of the negative terms, namely $\Delta(i,j,\ell)$ in (2.26) and $\Delta(j,k,\ell)$ in (2.27), implies that the differences can be positive even when none of the individual block moves are.

Two more complex neighborhoods result from composition of the finite-state

[6] No concerted attempt has been made to provide a dynamic program with the minimum number of lookups. A simpler set of grammar weights than those given may exist.

Trans_n^* neighborhood with the context-free BlockInsert_n^* neighborhood, namely

$$\text{Trans}_n^*(\text{BlockInsert}_n^*(\pi))$$
$$\text{and}$$
$$\text{BlockInsert}_n^*(\text{Trans}_n^*(\pi)).$$

These neighborhoods are not the same and probably have different sizes. Both are clearly larger than $\text{BlockInsert}_n^*(\pi)$ alone. They include some of the inside-out permutations handled by the previous grammar.

Composition of a weighted context-free language with a weighted regular relation results in another weighted context-free language, so both of these composite neighborhoods are context-free.[7] $\text{Trans}_n^*(\text{BlockInsert}_n^*(\pi))$ can be handled using the techniques of the next chapter—Section 3.9—by implementing the Trans_n^* transducer as the A model there. This transducer is quite different from the one shown in 2.23. It has a distinguished initial state q_I and an additional state q_i for each $i \in \binom{n}{1}$, and the following arcs:

$$\begin{aligned} q_I &\xrightarrow{i:i/0} q_I && \forall i \in \tbinom{n}{1} \\ q_I &\xrightarrow{i:\epsilon/0} q_i && \forall i \in \tbinom{n}{1} \\ q_i &\xrightarrow{j:j\ \ i/\Delta[i,j]} q_I && \forall i,j \in \tbinom{n}{1},\ j \neq i. \end{aligned}$$

The first type of arc is a self-loop at the initial state that leaves items in order. The second and third types create loops through an intermediate state that swap neighboring items. Because the transducer will be projected to its input language for use with the search algorithms, it doesn't matter that some of the arcs have multiple output symbols.

It is possible to convert the grammar $G_B(\pi)$ from Figure 2.24 into a grammar for the $\text{BlockInsert}_n^*(\text{Trans}_n^*(\pi))$ neighborhood by augmenting each non-terminal with two binary variables—one for its left endpoint and one for its right—indicating whether those endpoints transpose with their neighbors *outside* the span. A rule that combines (i, j) and (j, k) into (i, k) is only valid in this grammar if the right annotation on (i, j) matches the left annotation on (j, k), possibly indicating transposition of π_j and π_{j+1}.

This dissertation does not consider any of these complex neighborhoods further, though it would be interesting to derive normal forms and determine their sizes.

2.8.6 Parsing with Permutation Grammars

"Parsing" with permutation grammars is straightforward. Viterbi chart parsing, as in the weighted CKY algorithm, finds the best way of deriving each gram-

[7]Technically, all the grammars of this chapter are finite-state, because the languages they express are finite. The crucial distinction, then, is that the context-free grammars that result from these compositions have a polynomial number of rules, whereas the enumeration of the language as a regular expression would require exponential size.

mar non-terminal starting at each position $(i, k) \in \binom{n+1}{2}$ in the chart. Viterbi permutation parsing, on the other hand, simply finds the best permutation (of a sub-sequence) starting from each non-terminal—the non-terminals are necessarily position-dependent. It can use the same kind of chart, and the same algorithms, including many known optimizations, some of which arise in Chapter 3.

As an example, consider finding the best neighbor of π in $\text{Trans}^*(\pi)$ using the grammar of Figure 2.21. Compute the weight $\beta(N)$ of the best permutation generated by each non-terminal N in the grammar, in bottom-up order, starting with S_n. The permutation itself can be retrieved from back-pointers. S_n only has one derivation—π_n—and the weight of that rule—not given in the grammar—is 0, so

$$\beta(S_n) = 0. \tag{2.28}$$

There are two possible derivations of S_{n-1}, and

$$\beta(S_{n-1}) = \max\left(0,\ \Delta[\pi_{n-1}, \pi_n]\right). \tag{2.29}$$

For each remaining non-terminal, use the rule

$$\beta(S_i) = \max\left(\beta(S_{i+1}),\ \Delta[\pi_i, \pi_{i+1}] + \beta(S_{i+2})\right). \tag{2.30}$$

$\beta(S_1)$ then gives $B(\hat{\pi}) - B(\pi)$, where $\hat{\pi}$ is the neighborhood's best permutation. Compare this to the "forward" dynamic program (2.19) from Section 2.4.1.

Figure 2.33 shows pseudocode for computing the score of the best permutation in $\text{BlockInsert}^*_n(\pi)$ using the grammar G_B. It considers spans (i, k) in narrow-to-wide order, and includes the dynamic program for computing the grammar weights from Figure 2.26. The pseudocode dispenses with back pointers for simplicity, but it is easy to see how to include them.

2.9 Normal Forms

Using grammars and parsing to explore permutations has a side effect. Neighborhoods, as in Definition 2.3, are *sets* of permutations, meaning that each permutation is present in or absent from the neighborhood. The permutations themselves are the objects of interest, but the procedural descriptions of the neighborhoods, and the grammars of the previous section, may admit more than one way of arriving at the same permutation. For example, in the simple neighborhood Insert_n, moving the element at position i one to the right results in the same permutation (a transposition), as moving the element at $i+1$ one to the left. In the case of grammars, multiple different permutation trees express the same permutation. For the mathematical perspective of sets of permutations, multiple distinct *derivations* leading to the same permutation is *spurious ambiguity.*

```
procedure BlockInsert$^*(B, \pi)$
    $n \leftarrow |\pi|$
    for $i \leftarrow 0$ to $n-1$ do
        $\beta[i, i+1] \leftarrow 0$
        for $k \leftarrow i+1$ to $n$ do
            $\overrightarrow{\gamma}[i,i,k] \leftarrow \overleftarrow{\gamma}[i,i,k] \leftarrow 0$
            $\overrightarrow{\gamma}[i,k,k] \leftarrow \overleftarrow{\gamma}[i,k,k] \leftarrow 0$
        end for
    end for
    for $w \leftarrow 2$ to $n$ do
        for $i \leftarrow 0$ to $n-w$ do
            $k \leftarrow i+w$
            $\beta[i,k] \leftarrow -\infty$
            for $j \leftarrow i+1$ to $k-1$ do
                $\overrightarrow{\gamma}[i,j,k] \leftarrow$ Compute$(\overrightarrow{\gamma}, i, j, k)$ $+B[\pi_{i+1}, \pi_k]$
                $\beta[i,k] \leftarrow \max(\beta[i,k], \overrightarrow{\gamma}[i,j,k] + \beta[i,j] + \beta[j,k])$
                $\overleftarrow{\gamma}[i,j,k] \leftarrow$ Compute$(\overleftarrow{\gamma}, i, j, k)$ $+B[\pi_k, \pi_{i+1}]$
                $\beta[i,k] \leftarrow \max(\beta[i,k], \overleftarrow{\gamma}[i,j,k] + \beta[i,j] + \beta[j,k])$
            end for
        end for
    end for
    return $\beta[0,n]$
end procedure

function Compute$(\gamma, i, j, k)$
    return $\gamma[i,j,k-1] + \gamma[i+1,j,k] - \gamma[i+1,j,k-1]$
end function
```

Figure 2.33: Pseudocode for parsing with G_B. This procedure returns the score of the best permutation in the neighborhood, rather than that permutation, to avoid the complicated notation required to keep back pointers.

During search, this spurious ambiguity is not a problem. The parse forest may include many permutation trees that express the same permutation, but, crucially, each of these trees will compute the same score for the permutation, and search can choose arbitrarily among the derivations since the next step of search will throw away the tree and work only with the new permutation. In fact, it may be beneficial to allow spurious ambiguity, because the grammar may be smaller, and search faster, as a result.

There are, however, circumstances in which a one-to-one correspondence between trees and permutations is necessary:

- The simplest case is counting the number of permutations in a neighborhood using the grammar. It is straightforward to compute the number of trees in the parse forest, but unless each expresses an unique permutation, this will only be an upper bound on the size, in permutations, of the neighborhood, and not a tight one.

- Simulated annealing (Section 2.7.4), and other stochastic methods such as the Metropolis-Hastings algorithm, which will come up in Section 4.11.1, require a distribution over permutations rather than over derivations thereof.

- Several of the learning methods in Chapter 4, including k-best MIRA (Section 4.9), constrastive likelihood (Section 4.11.2), and expected loss minimization (Section 4.11.3), will also require normal-form.

How does spurious ambiguity arise, and how can it be eliminated? These are the central questions this section addresses. To begin, it shows necessary and sufficient conditions for spurious ambiguity. Second, it considers colorings of the nodes in the permutation trees, and corresponding grammar rules, that eliminate the ambiguity and result in a forest with a one-to-one correspondence between trees and permutations. Section 2.10 then considers the question of neighborhood size for several particular neighborhoods discussed above.

2.9.1 Spurious Ambiguity

A sufficient condition for spurious ambiguity of a grammar arises when it allows distinct trees with more than two consecutive spans that combine sequentially either all in order, or all reversed. Let the first three such spans be (i, j), (j, k), and (k, ℓ), for example. (i, j) can combine with (j, k) first, and the result with (k, ℓ), or (j, k) with (k, ℓ) first, and the result with (i, j). Either way, the same permutation is expressed in two different ways (see Figure 2.34). With m such constituents, the number of trees expressing the same permutation increases as the Catalan number C_{m-1}. For example, the identity permutation corresponds to any derivation that contains no exchanges, and there are an exponential number of these.

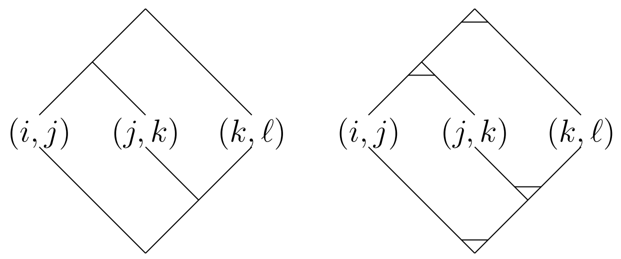

Figure 2.34: Spurious ambiguity in permutation trees: both trees on the left preserve the permutation (i,j) (j,k) (k,ℓ), and both trees on the right reverse the permutation to (k,ℓ) (j,k) (i,j).

In fact, refining the requirement above just a little will lead to a *necessary* condition for spurious ambiguity. The additional requirement is that the spans to combine are not themselves already combinations of the same orientation. That is, when combining the spans in order, they must consist of either single items or reversed sub-spans, and when combining them in reverse, they must consist of single items or in-order sub-spans. Call this the *more-than-two* condition.

Theorem 2.2 *The* more-than-two *condition is necessary for spurious ambiguity.*

Proof: Assume there exist distinct trees, T_1 and T_2, that express the same permutation π' of length n. In order to arrive at a contradiction, assume also that T_1 and T_2 do not satisfy the requirement described above that they share a sequence of more than two spans that either remain in sequence or exactly reverse their sequence.

Let (i,k) be the span of the first node in T_1 that differs from T_2 in a depth-first traversal starting at the root. Because T_1 and T_2 are assumed different, this necessarily exists. Call the nodes N_1 and N_2 respectively. Clearly N_1 and N_2 have the same orientation, because T_1 and T_2 must express the same relative ordering of π_{i+1} and π_k.

Therefore, N_1 and N_2 must differ in their split point. Let j_1 be the split point of N_1, so that it combines (i,j_1) and (j_1,k), and let j_2 be the split point of N_2, so that it combines (i,j_2) and (j_2,k). Assume without loss of generality that $j_1 < j_2$.

Now, consider the three spans (i,j_1), (j_1,j_2), and (j_2,k). To see that they must satisfy the more-than-two condition, consider that T_1 implies that (j_1,j_2) is adjacent to (j_2,k) in π', because (j_1,k) is a constituent of T_1. Likewise, T_2 implies that (j_1,j_2) is adjacent to (i,j_1) in π'.

This contradiction implies that two trees cannot differ and still express the same permutation without satisfying the more-than-two condition. □

The next concern is to eliminate spurious ambiguity from the parse forest to allow computation of the neighborhood size. The essential insight here derives from the fact, established above, that the more-than-two condition is the only source of spurious

$$\begin{array}{rcl}
S_{i-1,i} & \rightarrow & \pi_i \\
\overrightarrow{S}_{i-1,i} & \rightarrow & \pi_i \\
\overleftarrow{S}_{i-1,i} & \rightarrow & \pi_i, \ \forall i \in \binom{n}{1} \\
\\
S_{i,k} & \rightarrow & \overrightarrow{S}_{i,k} \\
S_{i,k} & \rightarrow & \overleftarrow{S}_{i,k}, \ \forall (i,k) \in \binom{n+1}{2}, \ k-i \geq 2 \\
\\
\overrightarrow{S}_{i,k} & \rightarrow & S_{i,j} \ \ \overleftarrow{S}_{j,k} \\
\overleftarrow{S}_{i,k} & \rightarrow & \overrightarrow{S}_{j,k} \ \ S_{i,j}, \ \forall (i,j,k) \in \binom{n+1}{3}
\end{array}$$

Figure 2.35: A normal-form grammar for the dynamic programming block insertion neighborhood. The second set of rules explicitly excludes the case $k = i+1$, which is handled by the first set of rules, to avoid ambiguity in the grammar. The alternative would be a more complicated third set of rules, with special handling for width-one children on either or both sides—six additional rules. The given grammar has only one way of generating $\pi_i \ \pi_{i+1}$, that is by labelling π_i with $S_{i-1,i}$ and π_{i+1} with $\overleftarrow{S}_{i,i+1}$. Analogously, there is only one way to generate $\pi_{i+1} \ \pi_i$.

ambiguity. In order to eliminate spurious ambiguity, it will suffice to allow only one of the trees that combines more than two sub-constituents of the opposite type.

2.9.2 Normal Form for Block Insertions

In the case of the BlockInsert$^*_n$ neighborhood, eliminating spurious ambiguity is simple, straightforward, and well studied (Eisner, 1996; Zens and Ney, 2003). Allow either left-to-right or right-to-left combination (which to allow is a decision that can be made independently for in-order and reversed combinations) with only a simple change to the grammar.

The full grammar of Figure 2.24 allowed any pair of constituents to be combined in order. To eliminate spurious ambiguity, require that the right constituent not already be combined in order. Likewise with reversed combinations. (The left constituent would work just as well.) The new grammar, G_B^{NF}, is given in Figure 2.35. The left-branching trees shown above the permutations in Figure 2.34 are normal under this grammar, while the right-branching trees shown below the permutations are not.

Normal form does not come at no cost. G_B^{NF} has $\Theta(n^2)$ more rules than G_B. Instead of computing a single value $\beta(S_{i,k})$ for each span (i,k) during search, normal

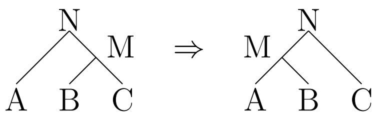

Figure 2.36: Converting in-order nodes to normal form.

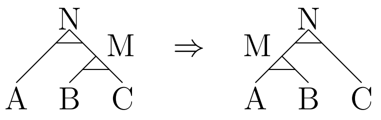

Figure 2.37: Converting swap nodes to normal form.

form requires computation of three values: $\beta(S_{i,k})$, $\beta(\overrightarrow{S}_{i,k})$, and $\beta(\overleftarrow{S}_{i,k})$.

$$\beta(S_{i,k}) = \max(\beta(\overrightarrow{S}_{i,k}),\ \beta(\overleftarrow{S}_{i,k}))$$

is cheap to compute, though. Since G_B requires $\Theta(n^3)$ to parse, an extra factor of $\Theta(n^2)$ is only a small change in efficiency.

Theorem 2.3 *The normal-form grammar G_B^{NF} expresses the same permutations as the grammar G_B: $L(G_B^{\text{NF}}) \equiv L(G_B)$.*

Proof: If T is a non-normal tree, then it has a node N whose right child M is the same type. Let A be the left child of N, and B and C be the left and right children of M, respectively.

If N is an in-order node, then the following achieves the same permutation, namely $\pi(A)\ \pi(B)\ \pi(C)$:

- change M to join A and B,
- make M the *left* child of N, and
- promote C to be N's right child.

Both N and M remain in-order nodes. An illustration of this transformation appears in Figure 2.36.

If N is a swap node, then the same procedure achieves the correct permutation, namely $\pi(C)\ \pi(B)\ \pi(A)$, with N and M remaining swap nodes. See Figure 2.37.

If T has multiple such non-normal constructions, eliminating them in a depth-first order is guaranteed to preserve the permutation and result in a normal tree. $\square$

Theorem 2.4 *The normal-form grammar G_B^{NF} is unambigous: each permutation in $L(G_B^{\text{NF}})$ has a single derivation under the grammar.*

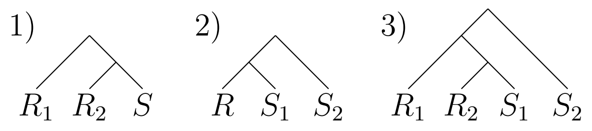

Figure 2.38: Alternate trees to the normal-form tree

Proof: Overloading notation and additionally treating each sub-tree as the set of indices at its leaves, then given an in-order node N with left and right children A and B respectively,

$$\forall a \in A, \forall b \in B, a \prec b, \tag{2.31}$$

meaning a precedes b in the permutation. Whereas if N is a swap node, then

$$\forall a \in A, \forall b \in B, b \prec a. \tag{2.32}$$

A permutation is fully determined by the set of precedence relations that such a tree encodes.

Consider any normal-form tree or subtree T whose topmost node is in-order, and whose children nodes are R and S. This tree, and any that might contain it, clearly has $R \prec S$, where this overloading is meant to imply the relationship in (2.31). Whenever a reverse node occurs, it permanently changes the order of the sets that its two children comprise. Therefore, any tree that expresses $R \prec S$ must combine any subset of R with any subset of S in order. Assume, for the sake of a contradiction, that there is a normal-form tree, T', different from T, that encodes the same permutation.

There are only three possibilities,[8] depicted in Figure 2.38:

1. T' combines R_1 with $R_2 \cup S$, where $R = R_1 \cup R_2$,
2. T' combines $R \cup S_1$ with S_2, where $S = S_1 \cup S_2$, or
3. it combines R_1 and S_2 separately with $R_2 \cup S_1$, where $R = R_1 \cup R_2$ and $S = S_1 \cup S_2$.

Consider each in turn: In (1), both T' and its right child must be in-order, which is non-normal. In (2), the implication is that the current S is an in-order node, since it puts $S_1 \prec S_2$. This would make T non-normal, because T is in-order and so is its right child S—a contradiction. In (3), S must be an in-order node (to keep S_1 before S_2) or it contradicts $R_2 \prec S$, but this leads to the same contradiction as (2). Therefore normal-form in-order nodes express unique permutations.

Reverse nodes have the same three cases. Separately putting two parts of R—R_1 and R_2—after S can only be accomplished by a non-normal tree. Separately putting two parts of S—S_1 and S_2—before R requires that $S_2 \prec S_1$ in the permutation,

[8]If R or S is a leaf node, then some of these scenarios are impossible, because leaves cannot be split. The argument stands.

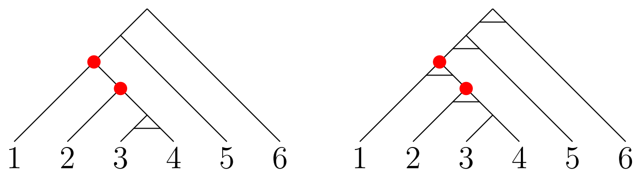

Figure 2.39: Normal-form trees from the dynamic programming insertion neighborhood. A red circle, ●, marks red nodes of either orientation. The tree on the left corresponds to the permutation 1 2 4 3 5 6, while the tree on the right corresponds to 6 5 3 4 2 1.

implying that S itself is a swap node, a contradiction. Finally, if part of R (R_2) combines with part of S (S_1) first, that must be a reverse node. Then T' must further put S_2 before S_1R_2 to achieve $S_2 \prec R_2$, implying again that S swaps to achieve $S_2 \prec S_1$. This is the same contradiction as in the second case.

The only conclusion is that T' does not exist, implying that normal-form trees under G_B^{NF} express unique permutations. $\square$

2.9.3 Normal Form for Insertions

For the limited-width insertion neighborhoods expressed by the grammars G_I and $G_{B\leq w}$, normal form is not as simple, and has not previously been studied. This section introduces a novel normal form for any $G_{B\leq w}$ neighborhood, including $G_I \equiv G_{B\leq 1}$.

In $G_{B\leq w}$, one of the constituents combined at each node in the tree must have size $\leq w$. The more-than-two condition is still necessary and sufficient, but its description can be refined in this case. At most one of the many spans can have size more than w, and each tree must successively combine smaller constituents with the one large one. There are nonetheless many possible combination orders in any case except when the large constituent is left- or rightmost.

There is one further possibility that the above paragraph misses, which is that some of the smaller constituents could combine first to form slightly larger constituents that are still no wider than w, assuming $w \geq 2$. This doesn't change the analysis, however, because such trees could never be normal.

Decree that the normal-form tree is the one that combines the large constituent first with all the small constituents to its left, and then with those to its right—necessarily in inside-out order. In order to enforce this, the grammar must allow only trees of this type. For this purpose, it will be necessary to introduce a new label for nodes. Let "white" refer to nodes that combine constituents in order in a left-branching structure, and "black" to those that reverse constituents, also left-branching. Nodes that combine constituents right to left, in a right-branching structure, of either type, will be colored "red". Finally, leaf nodes can freely participate

$$
\begin{aligned}
S_{i-1,i} &\rightarrow \pi_i \\
\overrightarrow{S}_{i-1,i} &\rightarrow \pi_i \\
\overleftarrow{S}_{i-1,i} &\rightarrow \pi_i,\ \forall i \in \binom{n}{1}
\end{aligned}
$$

$$
S_{i,k} \rightarrow \overrightarrow{S}_{i,k} \text{ or } \overleftarrow{S}_{i,k} \text{ or } \overset{\text{red}}{S}_{i,k},\ \forall (i,k) \in \binom{n+1}{2},\ k-i \geq 2
$$

$$
\begin{aligned}
\overrightarrow{S}_{i,k} &\rightarrow S_{i,j}\ \overleftarrow{S}_{j,k} \\
\overleftarrow{S}_{i,k} &\rightarrow \overrightarrow{S}_{j,k}\ S_{i,j},\ \forall (i,j,k) \in \binom{n+1}{3},\ k-w \leq j < k
\end{aligned}
$$

$$
\begin{aligned}
\overset{\text{red}}{S}_{i,k} &\rightarrow \overleftarrow{S}_{i,j}\ \left(\overleftarrow{S}_{j,k} \text{ or } \overset{\text{red}}{S}_{j,k}\right) \\
\overset{\text{red}}{S}_{i,k} &\rightarrow \left(\overrightarrow{S}_{j,k} \text{ or } \overset{\text{red}}{S}_{j,k}\right)\ \overrightarrow{S}_{i,j},\ \forall (i,j,k) \in \binom{n+1}{3},\ i < j \leq \min(i+w,\ k-w-1)
\end{aligned}
$$

Figure 2.40: A normal-form grammar $G^{\text{NF}}_{B\leq w}$. $\overrightarrow{S}$ is "white", $\overleftarrow{S}$ is "black", and $\overset{\text{red}}{S}$ is "red".

in any combinations and have no color.[9]

One example of a normal-form grammar is sketched as follows. If the right constituent to be combined has width $\leq w$, use these rules:

- White $\rightarrow$ Any Non-White (in order)
- Black $\rightarrow$ Any Non-Black (reversed)

If, on the other hand, the right constituent has width $> w$ but the left constituent has width $\leq w$, use these rules:

- Red $\rightarrow$ Non-White Non-White (in order)
- Red $\rightarrow$ Non-Black Non-Black (reversed)

The first pair of rules are identical to those in the grammar G^{NF}_{B}. The second pair are invoked only when combining a wider constituent with narrower constituents to its left. A formal definition of this grammar appears in Figure 2.40. Call this grammar $G^{\text{NF}}_{B\leq w}$, and let $G^{\text{NF}}_{I} = G^{\text{NF}}_{B\leq 1}$.

Theorem 2.5 *The normal-form grammar $G^{\text{NF}}_{B\leq w}$ expresses the same permutations as the grammar $G_{B\leq w}$: $L(G^{\text{NF}}_{B\leq w}) \equiv L(G_{B\leq w})$.*

[9] Or let "green" be a color that substitutes for any of the others.

Proof: It suffices to show that for any tree T compatible with $G_{B\leq w}$ that is not compatible with $G^{\mathrm{NF}}_{B\leq w}$, there is a normal form tree T' expressing the same permutation.

If T is not normal, then the more-than-two condition indicates that there is some sequence of at least three spans that T combines in sequential order. At most one of those spans has width $> w$. Call this (i, j) if it exists.

There exists a normal-form tree that combines the same spans and results in the same permutation. If there is no span (i, j) such that $j - i > w$, then T' combines the spans with a left-branching tree. If (i, j) exists, then T' combines the spans to its left with it in a right-branching tree, then combines that tree with the spans to the right of (i, j) with a left-branching structure. In either case, T' exists given T. □

Theorem 2.6 *The normal-form grammar $G^{\mathrm{NF}}_{B\leq w}$ is unambiguous: each permutation in $L(G^{\mathrm{NF}}_{B\leq w})$ has a single derivation under the grammar.*

Proof: Repeat the argument from the proof of Theorem 2.2. Let T_1 be compatible with $G^{\mathrm{NF}}_{B\leq w}$ and let T_2, assumed different, express the same permutation. It suffices to show that T_2 is not normal.

As in the prior proof, let N_1 and N_2, both spanning some (i, k) be the first difference between the trees. If N_1 has no wide children, then the proof of Theorem 2.4 applies, because the grammars are identical for narrow spans. Therefore, consider only the case where N_1 has a wide child span.

If j_1 and j_2, $j_1 < j_2$ are the two split points, regardless of which pertains to which tree, then the proof of Theorem 2.2 argues that the three spans (i, j_1), (j_1, j_2), and (j_2, k) combine in both trees either all in order or all in reverse.

First, let j_1 pertain to T_1, so N_1 combines (i, j_1) with (j_1, k). N_1's right child must therefore have the same orientation as N_1. Along with the assumption that T_1 is normal, this implies that N_1 and its right child are both red. N_1 therefore has a largest wide right corner with the opposite orientation, whose left endpoint is necessarily beyond j_2. Call its span (j_3, k). T_2 must have a node with the same orientation covering (j_3, k), because the relative order of π_{j_3+1} and π_k is fixed. Therefore N_2's right child is wide, and N_2 is also red. But, because T_2 combines (i, j_2) before attaching that to (j_2, k), N_2's left child must have the same orientation as N_2, which the red rules disallow. Therefore, T_2 cannot be normal.

Second, let j_2 pertain to T_1, so N_1 combines (i, j_2) and (j_2, k). Again, (i, j_2) must be wide by the assumptions. Either of its children may be wide, or it may be the first wide node. If its left child is wide, then T_2 violates normal form by combining (j_1, j_2) and (j_2, k) before attaching them to (i, j_1). If N_1's right child is wide, then T_2 combines to the right before it combines to the left, and if neither child is wide, then T_2 violates the left-branching condition. In any case, T_2 cannot be normal. □

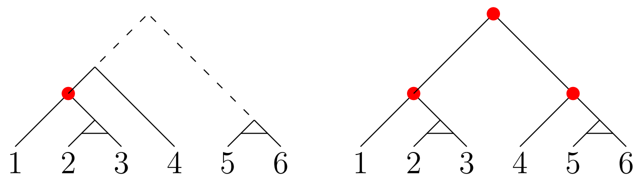

Figure 2.41: Two left-anchored trees expressing the same permutation, 1 3 2 4 6 5. The tree on the left is not normal according to $G^{\text{NF}}_{i=0}$ because it tries to combine two wide children with opposite orientations, indicated by dotted lines. The tree on the right is normal because both children are red.

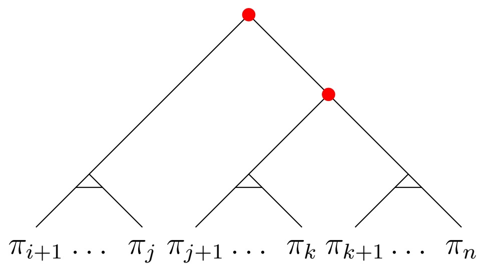
$\pi_{i+1} \ldots \pi_j$ $\pi_{j+1} \ldots \pi_k$ $\pi_{k+1} \ldots \pi_n$

Figure 2.42: An example right-anchored tree. Three black constituents can be combined in-order at the right anchor using the right-branching rules. The in-order nodes would be colored red.

2.9.4 Normal Forms for Special Cases

Now, consider the neighborhood that allows two wide constituents to combine in the special case when the left-most endpoint is zero using the rules of $G_{i=0}$, introduced in Section 2.8.3. These combinations always take place in a left-to-right manner, obviously. But, the fact that the right constituent may be wide means that in many cases it will be colored red, which means its type is unknown. Therefore it won't work to indiscriminately use the usual left-to-right rules.

Fortunately, because the $i = 0$ constraint already enforces a left-branching structure, the grammar can treat constituents with two wide children as red and use the right-branching rules instead, with a slight modification. The $G_{i=0}$ rules that build in-order nodes then combine only red and black nodes, and the rules that build reversed nodes combine only red and white nodes. The result is a left-branching structure anchored at 0, consisting of right-branching subconstituents. Figure 2.45 shows the modified grammar rules, and Figure 2.41 shows an example.

Symmetrically, consider the case when the right-most endpoint is n, the length of the permutation. Now the combinations are necessarily right-branching. Unfortunately, left-branching rules are inadequate for this case. Consider three wide constituents colored black that must combine in order, the rightmost endpoint at n, as shown in Figure 2.42. These could only combine in a right-branching way. However,

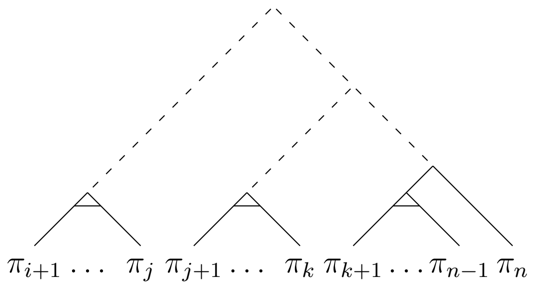

Figure 2.43: Another example right-anchored tree. Three black constituents followed by a leaf at n cannot be combined using the right-branching rules, because the in-order node spanning (k, n) would be colored white.

the left-branching rules don't allow this—they forbid the right constituent from being white.

Right-branching rules are also inadequate. They would allow the given instance—form a red node from the two rightmost, then combine with the leftmost using the same rule. But consider the case of three wide black nodes followed by a leaf node at n, as in Figure 2.43. To combine these, the rightmost black node must combine with the leaf first (two wide nodes can't combine unless they are anchored at n). That creates a white node using the left-branching rules. Neither pair of rules allows a black node to combine with a white node in order.

The problem, essentially, is that the grammar prefers left-branching structure but needs to accommodate a right-branching situation. The obvious, though somewhat expensive, solution is to introduce an additional color. A better solution is to use a right-branching version of the existing left-branching rules when both constituents are wide and $k = n$, such that

- White $\rightarrow$ Non-White Any (in order)
- Black $\rightarrow$ Non-Black Any (reversed)

This pair of rules, combined with those that already exist, would admit both of the examples described above.

This will only work properly if these grammar rules apply only when both constituents are wide, because otherwise these right-branching rules would coincide with the right-branching red rules. With this stipulation, narrow constituents must be incorporated into wide red constituents before they can be combined with other wide constituents ending at n.

Even if that is a solution for the $k = n$ case, what happens with $G_{i=0} \cup G_{k=n}$? That is, how do the two cases interact when $i = 0$ and $k = n$? The rules that apply when $i = 0$ cannot build constituents from 0 to n using wide right constituents of the same type built with right-branching rules, because those latter are not red. However, the same rules can apply from both directions if the right constituent is of

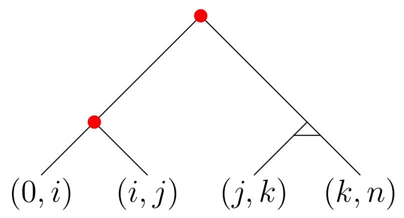

Figure 2.44: An example tree with children anchored at both 0 and n. All four spans shown are wider than w. $(0,j)$ is therefore built according to $G^{\text{NF}}_{i=0}$ and (j,n) according to $G^{\text{NF}}_{k=n}$. The node covering $(0,n)$ could be built according to $G^{\text{NF}}_{i=0}$, resulting in a red node, or according to $G^{\text{NF}}_{k=n}$, resulting in a white node. As Figure 2.45 indicates, the solution is to remove the $G^{\text{NF}}_{k=n}$ rule when $i=0$.

$$\begin{aligned}
\overset{\text{red}}{S}_{0,k} &\rightarrow \left(\overleftarrow{S}_{0,j} \text{ or } \overset{\text{red}}{S}_{0,j}\right) \left(\overleftarrow{S}_{j,k} \text{ or } \overset{\text{red}}{S}_{j,k}\right) \\
\overset{\text{red}}{S}_{0,k} &\rightarrow \left(\overrightarrow{S}_{j,k} \text{ or } \overset{\text{red}}{S}_{j,k}\right) \left(\overrightarrow{S}_{0,j} \text{ or } \overset{\text{red}}{S}_{0,j}\right), \ \forall (j,k) \in \tbinom{n}{2}, \ w < j < k - w \\
\\
\overrightarrow{S}_{i,n} &\rightarrow \left(\overleftarrow{S}_{i,j} \text{ or } \overset{\text{red}}{S}_{i,j}\right) S_{j,n} \\
\overleftarrow{S}_{i,n} &\rightarrow S_{j,n} \left(\overrightarrow{S}_{i,j} \text{ or } \overset{\text{red}}{S}_{i,j}\right), \ \forall (i,j) \in \tbinom{n+1}{2}, \ i + w < j < n - w
\end{aligned}$$

Figure 2.45: Normal-form grammar rules for special cases $G_{i=0}$ and $G_{k=n}$, with implicit argument w. These rules apply in addition to the rules of $G^{\text{NF}}_{B\leq w}$. If both $G_{i=0}$ and $G_{k=n}$ are in use, then disallow the case $i=0$ from the special $k=n$ rules.

the opposite type. Say there is a red constituent, connecting two wide constituents in order, spanning $(0,j)$, and a black constituent, also connecting two wide constituents, spanning (j,n). Then both the $i=0$ red rule and the $k=n$ white rule apply to build an in order constituent spanning $(0,n)$.

This is not the same spurious ambiguity encountered before. This is a new kind of spurious ambiguity created by the colored grammar rules that were introduced to prevent the other kind. The two trees built in this case would be identical, but they would nonetheless be treated separately within the grammar. A simple solution is to explicitly prevent the $k=n$ rules from use when $i=0$ and the latter rules are in effect.

Figure 2.45 gives additional grammar rules for $G^{\text{NF}}_{i=0}$ and $G^{\text{NF}}_{k=n}$, for use in addition to those of some $G^{\text{NF}}_{B\leq w}$.

2.10 Neighborhood Size

Given the normal form grammars described or introduced in the previous section, it is possible to compute the number of distinct permutations in the corresponding neighborhoods. This section does so for a few of the neighborhoods. For Trans_n^*, there was no spurious ambiguity in the original grammar, therefore G_T suffices. The other computations use the normal-form grammars G_B^{NF}, G_I^{NF}, and $G_{B\leq 2}^{\text{NF}}$.

2.10.1 Size of the Transpositions Neighborhood

The size of Trans_n^*, $T(n)$, follows from the recurrence equation:

$$T(n) = T(n-1) + T(n-2), \tag{2.33}$$

which derives directly from the grammar G_T of Figure 2.21. Some neighbors of π leave π_1 unchanged, in which case there are $n-1$ remaining items to permute. Others transpose π_1 and π_2, leaving $n-2$ others to permute.

Including the identity permutation in the neighborhood means $T(1) = 1$ and $T(2) = 2$. Therefore $T(n)$ is a Fibonacci number, and $T(n) = \Theta(\phi^n)$ where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The neighborhood has exponential size, as promised, despite linear search time.

2.10.2 Size of the Block Insertions Neighborhood

The size of the BlockInsert_n^* neighborhood is not as simple to compute. Let $s(n)$ be the number of normal-form trees for permutations of length n whose top-most node is in-order, or white. Observe that, because of symmetry, $s(n)$ also counts the number of trees with top-most reversal, colored black. Let $s(1) = 1$, so that leaf nodes count as both black and white. Now observe that the total number of normal-form trees for permutations of length n is $S(n) = 2s(n)$, except for the special case $S(1) = 1$. The following recurrence derives directly from the grammar G_B^{NF}:

$$s(n) = \sum_{k=1}^{n-1} S(k)s(n-k). \tag{2.34}$$

A tree spanning n items can be formed by combining any tree on the left spanning k items and a tree of the opposite type on the right spanning $n-k$ items.

Zens and Ney (2003) studied the size of this neighborhood before, and cited Shapiro and Stephens (1991), who showed that this $S(n)$ is a large Schröder number—$s(n)$ is a *small* Schröder number—after Schröder's second problem (Schröder, 1870). They also gave the following approximation to $S_n \equiv S(n-1)$, from Knuth (1973a):

$$S_n \approx \frac{1}{2}\sqrt{\frac{3\sqrt{2}-4}{\pi}}\left(3+2\sqrt{2}\right)^n n^{-\frac{3}{2}}. \tag{2.35}$$

This sequence also satisfies the more concise recurrence (Shapiro and Stephens, 1991):

$$nS(n) = (6n-9)S(n-1) - (n-3)S(n-2). \tag{2.36}$$

It is instructive to compare (2.35) to the number of trees allowed by the non-normal from grammar G_B. The number of binary-branching trees with n leaf nodes is the Catalan number C_{n-1}, and for any such tree, each internal node can be either of two colors—white or black, order-preserving or -reversing. The number of such trees is therefore $T_{n-1} = C_{n-1}2^{n-1}$, giving

$$T_n \approx \left(\frac{4^n}{n^{\frac{3}{2}}\sqrt{\pi}}\right)2^n = \frac{1}{\sqrt{\pi}}8^n n^{-\frac{3}{2}}. \tag{2.37}$$

2.10.3 Size of the Insertions Neighborhood

Consider the grammar G_I^{NF}. Let w_n, b_n, r_n, and t_n be respectively the number of white, black, red, and total normal-form trees with n leaves. Arbitrarily call a leaf node red, so that $r_1 = t_1 = 1$ and $w_1 = b_1 = 0$. Now, observe that $w_n = b_n = t_{n-1}$ because there is only one normal rule of each type—the right-branching rule. There are two rules with red left-hand sides, so

$$\begin{aligned} r_n &= (t_{n-1} - w_{n-1}) + (t_{n-1} - b_{n-1}) && (2.38)\\ &= 2(t_{n-1} - t_{n-2}). && (2.39)\end{aligned}$$

The total number of trees with n leaves is

$$\begin{aligned} t_n &= r_n + w_n + b_n && (2.40)\\ &= r_n + 2t_{n-1}. && (2.41)\end{aligned}$$

Rearranging (2.41) gives $r_n = t_n - 2t_{n-1}$. Therefore,

$$\begin{aligned} t_n - 2t_{n-1} &= 2t_{n-1} - 2t_{n-2} && (2.42)\\ t_n &= 4t_{n-1} - 2t_{n-2}. && (2.43)\end{aligned}$$

Assume that t_n has the form cx^n for some unknowns c and x. Then (2.43) leads to the quadratic equation

$$x^2 - 4x + 2 = 0, \tag{2.44}$$

which has solutions $x = 2 \pm \sqrt{2}$. Therefore,

$$t_n = c_1(2+\sqrt{2})^n + c_2(2-\sqrt{2})^n. \tag{2.45}$$

Invoking the base cases $t_1 = 1$ and $t_2 = 2$ gives two equations with two unknowns, the solution to which is

$$\begin{aligned} c_1 &= \frac{2-\sqrt{2}}{4}, && (2.46)\\ c_2 &= \frac{2+\sqrt{2}}{4}. && (2.47)\end{aligned}$$

Finally, substituting these back into (2.45) and simplifying gives

$$t_n = \frac{(2+\sqrt{2})^{n-1} + (2-\sqrt{2})^{n-1}}{2}. \tag{2.48}$$

This is sequence A006012 in the On-Line Encyclopedia of Integer Sequences.[10]

2.10.4 Size of the Width-2 Block Insertions Neighborhood

The analysis of $G_{B\leq 2}^{\mathrm{NF}}$ is similar to that of G_I^{NF} from the previous section. It makes use of the same symbols. Observe, now, that $w_n = b_n = t_{n-1} + t_{n-2}$, and

$$\begin{aligned} r_n &= 2(t_{n-1} - w_{n-1}) + 2(t_{n-2} - w_{n-2}) && (2.49)\\ &= 2(t_{n-1} - t_{n-2} - t_{n-3}) + 2(t_{n-2} - t_{n-3} - t_{n-4}) && (2.50)\\ &= 2(t_{n-1} - 2t_{n-3} - t_{n-4}). && (2.51) \end{aligned}$$

The total number of trees with n leaves is

$$\begin{aligned} t_n &= r_n + w_n + b_n && (2.52)\\ &= r_n + 2(t_{n-1} + t_{n-2}). && (2.53) \end{aligned}$$

Combining these two expressions for r_n gives

$$\begin{aligned} t_n - 2(t_{n-1} + t_{n-2}) &= 2(t_{n-1} - 2t_{n-3} - t_{n-4}) && (2.54)\\ t_n &= 4t_{n-1} + 2t_{n-2} - 4t_{n-3} - 2t_{n-4}. && (2.55) \end{aligned}$$

With the assumption that $t_n = cx^n$ this leads to the quartic equation

$$x^4 - 4x^3 - 2x^2 + 4x - 2 = 0, \tag{2.56}$$

whose largest solution is an algebraic constant $x \approx 4.2227433$. Therefore this $t_n = \Theta((\approx 4.2227433)^n)$.

Table 2.5 summarizes the sizes of many of the neighborhoods from this chapter, including the single-change neighborhoods. It includes the size for the case $n = 250$, of interest because of the XLOLIB benchmarks from Section 2.6. Table 2.6 shows the first ten sizes for a number of the neighborhoods from this chapter. The Trans_n^* neighborhood has exponential size, but the base of the exponent is relatively small, and its growth much slower than the others.

[10] http://www.research.att.com/~njas/sequences/A006012

Neighborhood	**Size**	$\boldsymbol{n = 250}$
Trans_n	$n-1$	249
Insert_n	$(n-1)^2$	62,001
BlockInsert_n	$\binom{n+1}{3}$	2,604,125
Trans^*_n	$\Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$	1.277652×10^{52}
Insert^*_n	$\Theta\left(\left(2+\sqrt{2}\right)^n\right)$	3.078578×10^{132}
$\text{BlockInsert}^*_n, w \leq 2$	$\Theta\left((\approx 4.2227433)^n\right)$	1.655187×10^{155}
BlockInsert^*_n	$\Theta\left(\left(3+\sqrt{8}\right)^n n^{-\frac{3}{2}}\right)$	8.598488×10^{186}
Π_n	$n!$	3.232856×10^{492}

Table 2.5: Neighborhood sizes

n	G_T	G_I^{NF}	$G_{B\leq 2}^{\text{NF}}$	$G_{B\leq 3}^{\text{NF}}$	G_B^{NF}	Π_n
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	6	6	6	6	6
4	5	20	22	22	22	24
5	8	68	90	90	90	120
6	13	232	376	394	394	720
7	21	792	1,584	1,806	1,806	5,040
8	34	2,704	6,684	8,316	8,558	40,320
9	55	9,232	28,220	38,396	41,586	362,880
10	89	31,520	119,160	177,672	206,098	3,628,800

Table 2.6: The first few terms in the recurrences for several neighborhood sizes.

2.11 Path Length

Each of the neighborhoods considered in this chapter induces a graph on the set of permutations Π_n. The graph has a vertex for each permutation $\pi \in \Pi_n$, and an edge connecting $\pi \to \pi'$ for each $\pi' \in \mathcal{N}(\pi)$. Because each of the neighborhoods considered is symmetric, i.e. $\pi' \in \mathcal{N}(\pi)$ implies $\pi \in \mathcal{N}(\pi')$, an undirected graph suffices.

This section gives several interesting results related to search trajectories in these induced graphs. First, though, it will consider the special case of the linear ordering problem under a total order.

2.11.1 Total Orders

What makes the linear ordering problem NP-hard in general is the possibility that the ordering preferences expressed in the matrix B are mutually inconsistent. For example, the matrix might express a preference for all three of $i \prec j$, $j \prec k$, and $k \prec i$. If no such inconsistent preferences exist in B, then the corresponding linear ordering problem is not hard at all—it reduces to sorting.

Definition 2.5 *A relation R is a* **total order** *if and only if*

1. *R is antisymmetric: iRj implies $\neg jRi$,*

2. *R is transitive: iRj and jRk implies iRk, and*

3. *R is total: iRj or jRi, for all i, j.*

Definition 2.6 *A LOP matrix B is* **consistent** *if there exists some total order relation R on $\{1, 2, \dots, n\}$ such that iRj if and only if $B[i, j] > B[j, i]$.*

The following theorem will prove necessary to demonstrate the usefulness of consistent LOP matrices for search.

Theorem 2.7 *If R is a total order relation and $\pi^* \in \Pi_n$ is the corresponding ordered permutation—$\forall i < j$, $\pi^*_i R \pi^*_j$—, then for any $\pi \in \Pi_n$, $\pi \neq \pi^*$, there must be some adjacent pair in π that is out of order, i.e. there exists an i, $1 \leq i < n$ such that $\pi_{i+1} R \pi_i$.*

Proof: Assume, without loss of generality, that π^* is the identity permutation $1\ 2\ \dots\ n$. Renaming the items in π^* with their positions always achieves this. No out-of-order adjacent pairs in π then means the items in π are increasing.

If the items in π are increasing, then n must occupy π_n, because if it occupied any other position, it would be immediately followed by something smaller. Further, $n - 1$ must occupy π_{n-1}, because $\pi_n = n$ and n is no longer available to follow $n - 1$ anywhere else. Applying this argument recursively leads to the conclusion that π is the identity permutation. Therefore, no $\pi \neq \pi^*$ with the property in question exists. $\square$

Theorem 2.7 implies that search using the Trans_n neighborhood will always find the true permutation π^* if the matrix B is consistent. Any superset of Trans_n, including each of the neighborhoods discussed in this chapter, has the same property.

2.11.2 Rank Correlation

One particularly useful consistent LOP matrix $B^{\tau}_{\pi^*}$ is defined as follows for some target permutation $\pi^* \in \Pi_n$:

$$\begin{aligned} B^{\tau}_{\pi^*}[i,j] &= 1 \quad \text{if} \quad i \prec j \in \pi^*, \\ B^{\tau}_{\pi^*}[i,j] &= 0 \quad \text{if} \quad j \prec i \in \pi^*. \end{aligned}$$

The matrix is called B^{τ} because of its relationship to Kendall's τ (Kendall, 1938), a measure of rank correlation. $B^{\tau}_{\pi^*}(\pi)$ counts the number of pairs in π that are in-order with respect to π^*, and is related to the rank correlation by

$$\tau(\pi^*, \pi) = \frac{2B^{\tau}_{\pi^*}(\pi)}{\binom{n}{2}} - 1. \tag{2.57}$$

The factor of $\binom{n}{2}$ normalizes the count to a fraction in the interval $[0, 1]$, and the multiplication by two and subtraction of one makes it a correlation in the interval $[-1, 1]$.

Altering $B^{\tau}_{\pi^*}(\pi)$ by making it grow larger as π becomes more different from π^* converts it into a loss function. Charon and Hudry (2007) called this loss function the *symmetric difference distance* δ:

$$\delta(\pi^*, \pi) \overset{\text{def}}{=} \binom{n}{2} - B^{\tau}_{\pi^*}(\pi). \tag{2.58}$$

Chapter 4 will make extensive use of this loss function, particularly for the guided learning procedures in Section 4.8 and expected loss minimization in Section 4.11.3.

2.11.3 Graph Diameter

Section 2.11.1 mentioned that the linear ordering problem with a consistent matrix is the same as sorting. This section makes that relationship explicit in order to measure the diameters of the graphs induced on the permutations by two of the neighborhoods.

An analogy to bubble sort shows that the diameter of the graph induced by the Trans^*_n neighborhood is $O(n)$. Actually, this differs slightly from bubble sort, because bubble sort can make multiple swaps of the same element in a single pass through the list, bubbling the largest element all the way to the end on the first pass. However, this fact doesn't affect the asymptotic runtime. Figure 2.46 shows an example.

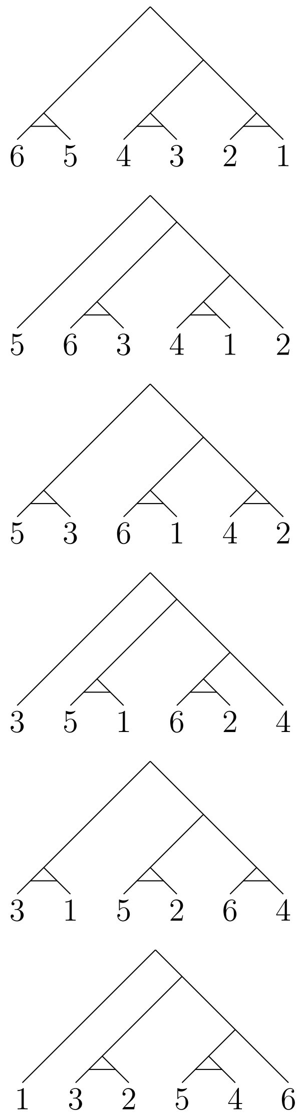

Figure 2.46: Bubble sort using transpositions. The result of the last permutation tree is the identity permutation 1 2 3 4 5 6. The distance between the permutations 6 5 4 3 2 1 and 1 2 3 4 5 6 in the graph induced by G_T is therefore no more than 6.

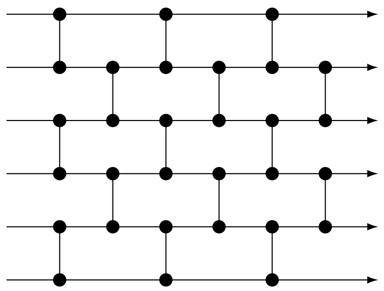

Figure 2.47: An odd-even sorting network. Horizontal lines represent *wires*, while vertical line segments represent *comparisons*. The result of a comparison is that the bottom wire holds the min, and the top wire the max, of the two inputs. For any n, the network contains n parallel layers and $\binom{n}{2}$ total comparisons.

Theorem 2.8 *The diameter of the graph induced on* Π_n *by the* Trans_n^* *neighborhood is at most* n.

Proof: The proof uses the *odd-even sorting network*, a sorting network that uses only adjacent transpositions (Knuth, 1973b; Cormen, Leiserson, Rivest, and Stein, 2001). The network consists of n layers, each with approximately $\frac{n}{2}$ comparisons. Odd-numbered layers compare all pairs of wires $(2i-1, 2i)$ for $1 \leq i \leq \frac{n}{2}$, and even-numbered layers compare pairs $(2i, 2i+1)$ for $1 \leq i < \frac{n}{2}$, where wires are numbered 1–n. Figure 2.47 shows an example. Compare to Figure 2.46.

Given the fact that odd-even transposition networks *are* sorting networks, meaning they sort their inputs, the proof of this theorem is trivial. Trans_n^* can simulate the layers of the odd-even network in sequence. Therefore every permutation is within n steps of the identity permutation. This suffices, because the neighborhood is invariant to renaming. □

A similar analogy to quick sort shows that the diameter of the graph induced by $L(G_I \cup G_{i=0})$ is $O(\log n)$. Because larger neighborhoods have all the edges of this neighborhood, as well as additional edges, this result applies to them as well. Figure 2.48 shows an example.

Theorem 2.9 *The diameter of the graph induced on* Π_n *by the neighborhood* $L(G_I \cup G_{i=0})$ *is at most* $\lceil \log_2 n \rceil$.

Proof: It suffices to show that every permutation π is within $\lceil \log_2 n \rceil$ steps of the identity permutation, because the neighborhood is invariant to renaming.

Given a permutation π, let m be a median element of π. Construct the normal-form permutation tree that joins first each element to the left of m, then each element to the right of m, onto the subtree that governs m. Among elements to the left of m,

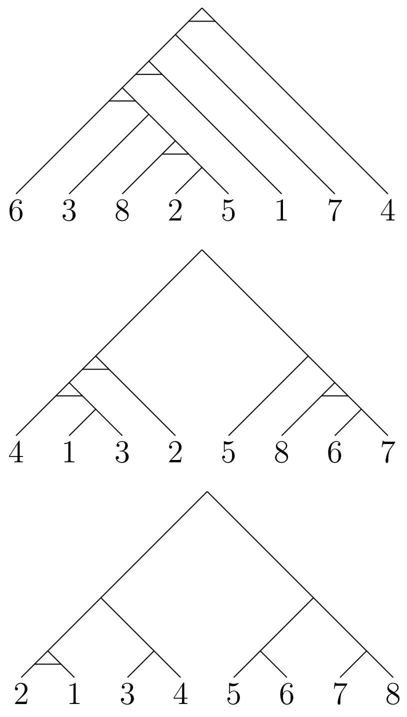

Figure 2.48: Quick sort using insertions. In the first tree, 5 serves as the pivot. In the second tree, 3 and 7 do. In the third tree, the sequences are size 2 each, so the pivot is irrelevant. The result of the third tree is the identity permutation 1 2 3 4 5 6 7 8, only three steps away from any other permutation in the graph induced by the neighborhood $L(G_I \cup G_{i=0})$.

those smaller than m stay in-order and those larger move to the other side. Among elements to the right, smaller move and larger stay. The result is a permutation with m in the middle, all smaller items to its left, and all larger items to its right. Call this $\pi^{(1)}$. That is,

- $\pi^{(1)}$'s span $(0, \lfloor \frac{n}{2} \rfloor)$ contains the items $\{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$, and
- $\pi^{(1)}$'s span $(\lceil \frac{n}{2} \rceil, n)$ contains the items $\{\lceil \frac{n}{2} \rceil, \ldots, n\}$.

The next step recursively considers the two halves of $\pi^{(1)}$ and performs the same procedure, then joins the two halves with an in-order node anchored at $i = 0$. The result is $\pi^{(2)}$, which has four subsequences, each an unsorted collection of the items in the appropriate subsequence.

By $\pi^{(\lceil \log_2 n \rceil)}$, this procedure reaches the identity permutation. □

2.11.4 Neighborhood Membership

An important question related to neighborhood permutation graphs is whether π and $\pi' \in \Pi_n$ are neighbors. That is, is $\pi' \in \mathcal{N}(\pi)$. Because all these neighborhoods are symmetric, whether $\pi \in \mathcal{N}(\pi')$ has the same answer.

It is straightfoward to answer this question using parsing algorithms, given a grammar corresponding to the neighborhood in question. Let $G(\pi)$ be the grammar. Then parse π' under $G(\pi)$ using a standard chart parsing algorithm, such as CKY. If parsing succeeds, then the two are neighbors, and if it fails, they are not. Because of the size of the grammar, using this method to determine if $\pi' \in L(G_B(\pi))$ requires $\Theta(n^3)$ time. This can be improved upon, however. A shift-reduce parser can decide the question in $\Theta(n)$ time.[11]

The shift-reduce parser takes advantage of contiguity. If $\pi' \in L(G_B(\pi))$, then every subsequence of π' that includes items from a contiguous subsequence of π, including individual items—subsequences of length one—, must be adjacent in π' to another such subsequence, to which it is also adjacent in π.

This description of the shift-reduce parser assumes, again without loss of generality, that π is always the identity permutation $1\ 2\ \ldots\ n$.

- The shift step removes one item i from the front of the permutation π' and pushes the span $(i-1, i)$ onto the stack.
- The reduce step looks at the top two spans on the stack: (i, j) and (k, ℓ). There are two possible reductions:
  - If $j = k$, then the two spans are contiguous and in-order, and they are replaced on the stack with the new span (i, ℓ).

[11]The parser as given answers the question for the BlockInsert$^*_n$ neighborhood. However, a simple modification, not presented here, makes it parse Insert$^*_n$, or other subsets of BlockInsert$^*_n$, instead.

$$\begin{array}{rcl}
 & \rightarrow & 3\ 1\ 2\ 4 \\
(2,3) & \rightarrow & 1\ 2\ 4 \\
(2,3)\ (0,1) & \rightarrow & 2\ 4 \\
(2,3)\ (0,1)\ (1,2) & \leftarrow & 4 \\
(2,3)\ (0,2) & \leftarrow & 4 \\
(0,3) & \rightarrow & 4 \\
(0,3)\ (3,4) & \leftarrow & \\
(0,4) & &
\end{array}
\qquad
\begin{array}{rcl}
 & \rightarrow & 3\ 1\ 4\ 2 \\
(2,3) & \rightarrow & 1\ 4\ 2 \\
(2,3)\ (0,1) & \rightarrow & 4\ 2 \\
(2,3)\ (0,1)\ (3,4) & \rightarrow & 2 \\
(2,3)\ (0,1)\ (3,4)\ (1,2) & &
\end{array}$$

Figure 2.49: Two examples of shift-reduce parsing for permutations of length 4. The stack is on the left, the permutation on the right, $\rightarrow$ indicates a shift step, and $\leftarrow$ a reduction. The parse on the left shows that 3 1 2 4 is in $L(G_B(1\ 2\ 3\ 4))$, while the failed parse on the right shows that 3 1 4 2—one of the inside-out permutations—is not.

  - If $i = \ell$, then the two spans are contiguous but out-of-order, and they are replaced on the stack with the new span (k, j).

The parser tries to reduce as long as the stack contains at least two spans. Whenever the stack is too small or no reduction is possible, it performs one shift step and tries to reduce again. Once the parser is finished, if the stack contains the single span $(0, n)$, then parsing is successful and π' is a member of the neighborhood. Otherwise, parsing fails, because π' is not in the neighborhood.

Figure 2.49 shows the steps of the shift-reduce parser on two example permutations of length 4. One is a member of the neighborhood, the other is not.

Theorem 2.10 *The shift-reduce parser described above runs in $\Theta(n)$ time.*

Proof: The shift step is performed exactly n times. Each shift is followed by a reduce step. The reduce step can succeed at combining two spans at most $n-1$ times, because each such combination reduces the number of available spans by one, and there are exactly n primitive spans. Each time reduce is called, it returns by failing exactly once. Each of the operations described above is constant time. Therefore, the total time is $\Theta(n)$. $\square$

Aurenhammer (1988) presented an algorithm for sorting *twisted sequences* in linear time. The definition of twisted sequences is slightly different from the neighborhood here, but constitutes the same set of permutations. The presentation is quite different, using a geometric interpretation. That algorithm is capable of sorting sequences other than permutations of known objects, making it more general.

Length	**Sentences**	**CDF**
0	71,196	0.095
1	332,160	0.540
2	279,514	0.914
3	60,037	0.994
4	4,099	1.000
5	82	1.000

Table 2.7: Path lengths from source to target ordering of greedy search on German-English parallel sentences. This table aggregates the data from Figure 2.50 over all sentence lengths. More than half of the reordered German sentences are in the neighborhood of the source German, and more than 90% are within two steps.

2.11.5 Shortest Paths

Section 2.11.4 showed how to determine whether two permutations π and π' are neighbors in a given neighborhood. This section takes the question one step further. If π and π' are not neighbors in the graph, how far apart are they? That is, what is the shortest path in the neighborhood permutation graph from π' to π? Short of brute force enumeration, no way of answering this question exactly is known for the neighborhoods described in this chapter. However, this section provides an empirical upper bound, often smaller than the graph diameter—a trivial upper bound itself.

Starting at π' and using the LOP matrix B^{τ}_{π} for search, Section 2.11.2 argued that, for any of the neighborhoods under consideration, the search is guaranteed to arrive at π. Because search takes steps in the induced permutation graph, the number of steps it requires to reach π is obviously an upper bound on the distance between them.

Figure 2.50 shows upper bounds on shortest paths generated from German-English translation alignment data described in Chapter 3. Empirical path lengths vary between zero—meaning the source and target permutations are identical—and five. The data consist of 747,088 sentence pairs, and the total breakdown of path lengths is given in Table 2.7.

It is worth pointing out, though it may be obvious, that when search in the very large-scale neighborhoods arrives at the desired permutation in one or two steps, then the bound is exact, but more than two steps is really an upper bound. For example, if the search takes three steps, that does not necessarily imply that there is no path of exactly two steps.

The smallest n for which the length of the search path is greater than two for *any* permutation in Π_n, is $n = 7$. There are 88 such permutations requiring three steps in the search path. One example is 1 4 6 2 7 3 5. The minimum loss neighbor is 1 2 4 6 3 5 7, which contains an inside-out permutation of loss 3 in its middle—4 6 3 5. This then requires two additional steps to untangle. But the permutation

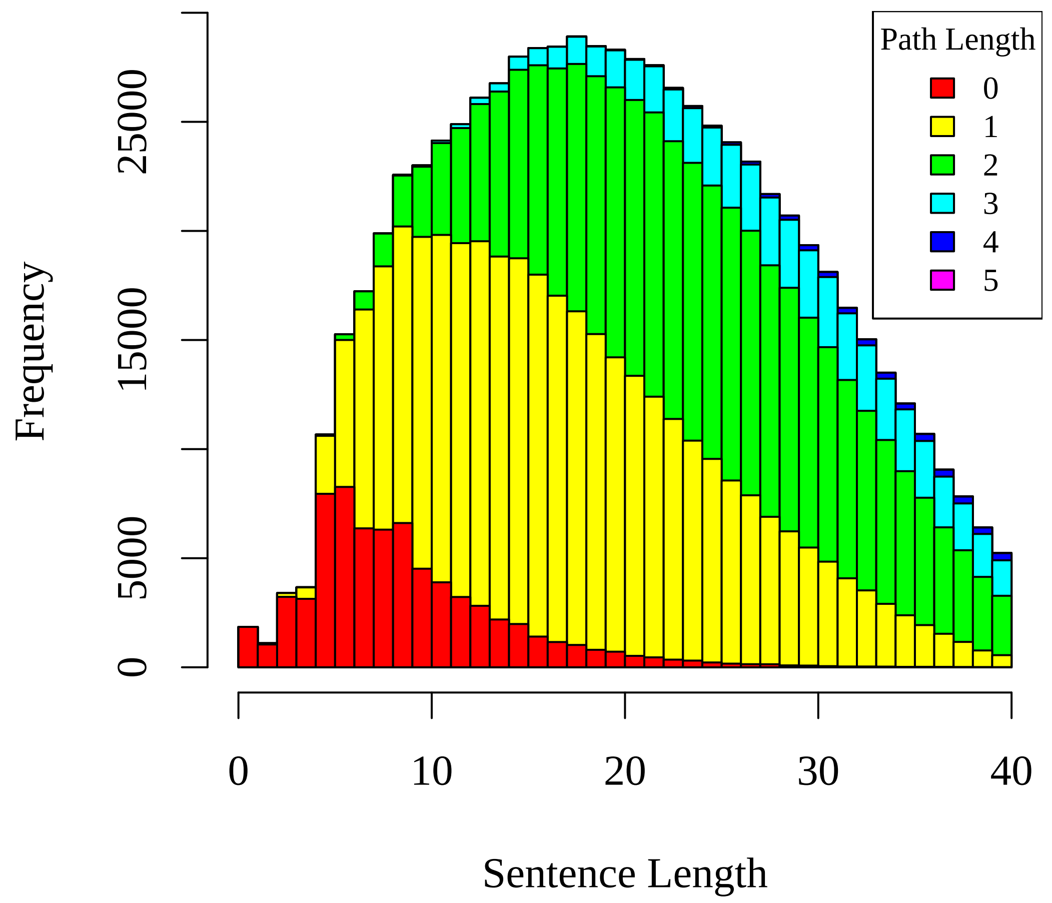

Figure 2.50: Path length from source to target ordering of greedy search on German-English parallel sentences. Length 0 means the source and target ordering are the same. Lengths greater than two are upper bounds on the true path length, which may be as small as two.

1 4 5 6 7 2 3, which has a loss of 8, is only one step from the identity.

This section ends with an open question: Is there a different LOP matrix that is better than B_π^τ? That is, can a different matrix reduce the average empirical search path length between arbitrary pairs of permutations?

2.12 Summary

This chapter presents a unified view of neighborhood search for the Linear Ordering Problem. Many of the neighborhoods, as well as other results herein, are of prior provenance, but this chapter fills in gaps where they occur, and extends prior work in completely new directions as well.

The Block $\mathcal{LS}_f$ shortcut search procedure introduced here is a state-of-the-art local search neighborhood for the XLOLIB benchmarks. It may prove of use for other LOP instances as well. The novel dynamic program for computing the costs of block transpositions in constant time per neighbor is essential to this result.

This chapter also attempts the first application of very large-scale neighborhood search to the LOP. Although the VLSNs do not outperform $\mathcal{LS}_f$ on XLOLIB benchmarks, the following chapters will show that these neighborhoods are nonetheless tremendously useful in their own right. Search for a good sentence reordering for machine translation will use the BlockInsert$_n^*$ neighborhood as a constraint to improve performance. Machine learning methods will make use of the VLSNs to approximate the partition function, which is intractable to compute over all $n!$ permutations.

To the extent that the VLSNs are useful, normal form grammars for them are necessary as well, to ensure a one-to-one correspondence between derivations and permutations. The three-color normal form for Insert$_n^*$ and limited-width block insertions is therefore also an important contribution. The normal forms also allow computation of neighborhood sizes, which are interesting in their own right. Finally, the graph diameter results contribute to understanding the structure that the VLSNs induce on the set of permutations.

Chapter 3

Reordering and Machine Translation

This chapter introduces a model of reordering for machine translation, including the Linear Ordering Problem of Chapter 2 as a component. It proposes handling word-to-word translation monotonically using a weighted finite-state transducer, and uses a trick involving FST projection to convert the entire translation problem, including a target N-gram language model, into a combinatorial optimization problem, reordering the source language sentence.

In order to reach that point, it introduces the Machine Translation problem, focusing heavily on the reordering component in the exposition. Section 3.3 reviews a number of prior reordering models, and discusses their strengths and weaknesses relative to a model based on the LOP. Section 3.4 describes aspects of the German language relevant to translation, and particularly the reordering necessary to translate German to English.

Section 3.5 gives a brief tutorial introduction to weighted finite-state transducers, then describes how to encode some well known models of translation using WFSTs. Section 3.6 then combines those finite-state automata with the Linear Ordering Problem and another novel function of permutations to produce the translation model that is this chapter's primary contribution. Section 3.7 describes the cyclic cost function, which scores triples of items in the permutation the same way the LOP scores pairs.

Section 3.8 provides the main empirical results of the dissertation. It demonstrates the utility of models based on the LOP for word reordering as a preprocessing step. This provides a fully automatic way to systematically rearrange German sentences into an order that is more conducive to translation into English.

The remainder of the chapter is dedicated to proposals of methods for decoding the entire model of Section 3.6. Because of the complexity of VLSN search in the presence of the finite-state automaton—polynomial, but with an unattractive exponent—these methods have not yet yielded empirical translation results. However, there are several interesting possibilities for further investigation.

Before proposing this chapter's translation model, it is appropriate to lay out the context into which it fits.

3.1 Machine Translation

Machine translation—the problem of translating among human languages automatically with computers—is as old as artificial intelligence (AI). Proposed as early as 1949 by Warren Weaver, it is now one of the major subfields of natural language processing. It is generally posed as a sentence-by-sentence problem—i.e. given a single sentence in one *source* language, often called f, translate that sentence into the *target* language, often called e.[1] Ultimately, in order to capture all the information available in the source language, machine translation will have to move beyond treating each sentence independently, but the complexity of the task as it is currently posed is already great.

For much of its history, machine translation was carried out in the tradition of *symbolic* AI. This thesis will not concern itself with this tradition, except to briefly mention a connection in Section 4.4 on features for learning.

In 1990, researchers at IBM published a seminal paper (Brown, Cocke, Della Pietra, Della Pietra, Jelinek, Lafferty, Mercer, and Roossin, 1990) introducing *statistical* machine translation. A later paper (Brown, Della Pietra, Della Pietra, and Mercer, 1993) proposed five increasingly sophisticated models of word-to-word translation now commonly known as IBM Models 1–5. Their Model 4 appears in Section 3.5.1, part of a discussion of encoding translation models as finite-state transducers. These models are now used primarily for the task of word alignment—see Section 3.8.1—rather than for the full translation task. They have been replaced by phrase-based and syntactic translation models.

Knight (1999) showed that decoding under such word-based translation models is NP-complete by separate reductions from the Hamiltonian Circuit Problem and the Minimum Set Cover Problem. The latter reduction points to the complexity in choosing multi-word translations to cover the source language sentence. The former reduction—more important to this thesis—demonstrates that another source of complexity is the ordering of the words. Section 3.6 will introduce a slightly different, more general, model of translation in terms of two NP-complete ordering problems—the Linear Ordering Problem of Chapter 2, and a generalization of the famous Traveling Salesman Problem.

Word-based statistical translation models have largely been surpassed by the more general *phrase-based* models described in detail in Och and Ney (2004). Phrase-based decoders segment the source sentence into contiguous chunks and use a phrase dictionary to translate entire chunks into the target language. The segmentation is non-deterministic, and the decoder searches simultaneously for the best segmentation,

[1]Some authors use a *noisy channel* formulation and reverse the *source* and *target* labels.

phrase translations, and target language phrase ordering. Section 3.3 discusses several reordering models proposed in the literature.

Several increasingly popular alternatives to phrase-based translation models fall under the generic label of *syntax-based* statistical machine translation. The simplest such models use unlabeled binary trees much like the permutation trees described in Section 2.8. The difference is that the translation models, such as inversion-transduction grammar (ITG) (Wu, 1997), use *synchronous* grammars—grammars that generate multiple strings simultaneously.

The full battery of syntax-based translation models is beyond the scope of this dissertation, but because of the relationship between synchronous grammars and the permutation trees used here, much of the work on decoding with such models is relevant to this work. This chapter invokes several instances of such work.

3.2 Resources

Because the machine translation problem is so well established, the field has developed numerous tools for evaluation of different models and decoding systems. These include large corpora of parallel sentences, automatic evaluation measures, aligners, decoders, and more. This chapter makes use of the following:

- The Europarl corpus (Koehn, 2005) consists of hundreds of thousands of parallel sentences in many languages from the proceedings of the European parliament. The experiments this chapter describes are carried out using the German-English portion of this corpus.

- GIZA++ (Och and Ney, 2003) is an alignment tool that implements all of the IBM models (Brown et al., 1993) and the HMM alignment model of Vogel, Ney, and Tillmann (1996).

- BerkeleyAligner (Liang, Taskar, and Klein, 2006b; DeNero and Klein, 2007) is an alternative to GIZA++. It is standard practice to train GIZA++ using both channel directions—source-to-target and target-to-source. Combining the two sets of alignments usually produces a better result than using either alignment alone. BerkeleyAligner shares information between these tasks during parameter estimation to improve alignment quality.

- Moses (Koehn, Hoang, Birch, Callison-Burch, Federico, Bertoldi, Cowan, Shen, Moran, Zens, Dyer, Bojar, Constantin, and Herbst, 2007) is a complete machine translation decoder. It uses beam search for phrase-based translation, incorporating lexicalized reordering models and target language models.

- Fsa (Kanthak and Ney, 2004) is a weighted finite-state transducer toolkit, used to implement the finite-state algorithms.

- BLEU score (Papineni, Roukos, Ward, and Zhu, 2002) is an automatic translation evaluation measure. Section A.1 describes it in full.
- METEOR (Lavie, Sagae, and Jayaraman, 2004; Banerjee and Lavie, 2005; Lavie and Agarwal, 2007) is another translation evaluation measure, described in Section A.3.
- TER (Snover, Dorr, Schwartz, Micciulla, and Makhoul, 2006) is still a third translation evaluation measure, described in Section A.4.

3.3 Reordering

As Section 3.1 suggests, choosing the word order in the target language is an important source of complexity in the machine translation task. This section further demonstrates that the reordering model is essential for good translation performance, and that the language model, though a good source of ordering information, is inadequate by itself.

This chapter and the next make extensive use of the concepts of *search error* and *model error*.

Definition 3.1 *A* **search error** *occurs when the answer returned by the search procedure differs from the answer that is optimal according to the model.*

Definition 3.2 *A* **model error** *occurs when the answer that is optimal according to the model differs from the true answer.*

3.3.1 Target Language Models

An early throw-away experiment in the work of this dissertation tried to recover English sentences from “bags of words” using an n-gram language model and the VLSN search methods of Chapter 2.[2] Greedy search using these neighborhoods rarely returned the correct English sentence, but not simply because of search error. Rather, model error was also to blame. The language model rarely prefers the true English sentence to all of its possible permutations.

While this problem might be mitigated to some extent with higher-order n-gram models, it will never go away completely. For example, a corpus consisting of the two sentences “John loves Mary.” and “Mary loves John.” would always defeat the language model on one of the two.

These unreported experiments do not stand alone. Al-Onaizan and Papineni (2006) reported substantially similar results. They defined what they called a *word*

[2]The language model component complicates the grammars of Section 2.8. Section 3.6 introduces a general model that handles this case, and Figure 3.6 on page 118 gives a grammar.

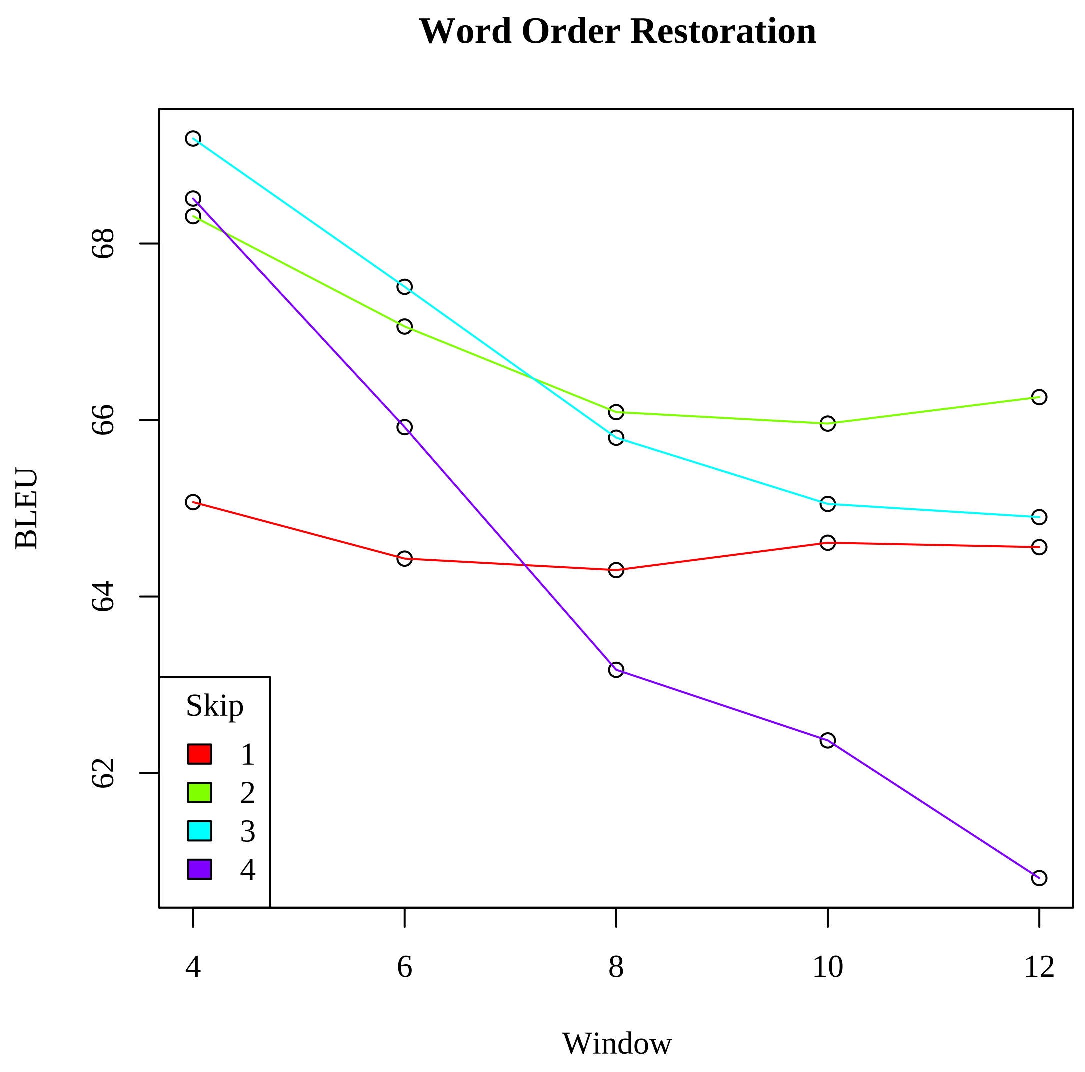

Figure 3.1: The word order restoration results of Al-Onaizan and Papineni (2006). The source is English words in Arabic word order, and the target is the original English. Arabic word order has a BLEU score of 0.5617. Each ordering is scored using only a trigram language model. Performance degrades for all maximum skip sizes as the window size increases, allowing words to move farther and farther from their source-language position.

order restoration task. English words start out in, in their case, Arabic word order, and the task is to restore them to English word order. Figure 3.1 shows a summary of their results using only a trigram language model to assign a score to each ordering. They varied two parameters to the search, as in Tillmann and Ney (2003):

Window is the maximum number of words allowed from the left-most uncovered source word to the right-most covered word. For a word to move to its right, it must remain uncovered while other words are covered. For a word to move to its left, it must be covered before its predecessors. In either case, the window parameter limits the extent of its movement.

Skip is the maximum number of uncovered source words allowed to the left of the right-most covered word. It determines how many words can move simultaneously and also limits movement to the left.

It is interesting to consider these constraints as forming neighborhoods in the sense of Chapter 2. When both the window and skip parameters are set to 1, the result is exactly the Trans_n^* neighborhood. However, more permissive settings result in new neighborhoods. Zens and Ney (2003) showed that when the window is unconstrained, resulting in the so-called IBM constraints (Berger, Brown, Della Pietra, Della Pietra, Kehler, and Mercer, 1996), the size of the neighborhood for skip parameter s is

$$r_n = \begin{cases} s^{n-s} \cdot s! & \text{if } n > k, \\ n! & \text{if } n \leq k. \end{cases} \tag{3.1}$$

The task is limited to reordering, so the BLEU scores reported always have perfect unigram precision—Section 4.6, starting on page 138, discusses such monolingual BLEU scores in more detail.

While these results indicate that the language model should help find a better target order than monotonic decoding, they also show that the language model alone suffers from a great deal of model error—even *given* the correct words. The *window* and *skip* parameters introduce systematic search errors, forcing the decoder to choose from among a subset of the possible permutations, in order to limit the effect of model error.

The most obvious problem with the language model is that it completely ignores the source language sentence. It looks exclusively at the target language words *after* translation of the source. The models that follow, including the one Section 3.6 proposes, take the source ordering into account in a variety of ways, modeling reordering as though it occurs *before* translation.

3.3.2 Distance Models

It is important to make a distinction, with phrase-based models, between the reordering that occurs *within* phrases, and reordering *between* phrases. Within-phrase reordering occurs without an explicit model, as part of the phrase translation process—phrase-table entries, such as the one depicted in Figure 3.2 on page 99, may contain arbitrary reordering of concepts. The models described below are thus limited to between-phrase reordering.

Och and Ney (2004) employed a very simple reordering model, defined in terms of an alignment feature h_{AL}:

$$h_{\mathrm{AL}}(e_1^I, f_1^J, \pi_1^K, z_1^K) = \sum_{k=1}^{K+1} \left| j_{\pi_k - 1} - j_{\pi_{k-1}} \right| , \tag{3.2}$$

where e and f are the sentences, z is the sequence of alignment templates, and π is the sequence of phrase alignments. $j_{\pi_k - 1}$ is the position before the start of phrase k, while $j_{\pi_{k-1}}$ is the last position of the previous phrase. If phrase translation is entirely monotonic, $h_{\mathrm{AL}} = 0$, and it is larger than 0 if any reordering occurs.

Incorporated into a log-linear model, this implies a geometric distribution with parameter $\exp(\lambda_{\mathrm{AL}})$, where λ_{AL} is the weight of feature h_{AL}. Assuming $\lambda_{\mathrm{AL}} < 0$, this model always assigns the highest weight to the monotonic ordering, and penalizes long-distance movement exponentially more than short-distance movement.

While this model is somewhat appropriate for some language pairs, it is a poor model in general. For example, translation between German and English has to move verbs from SOV order to SVO order. This distance can become arbitrarily long as the object becomes increasingly complex. Section 3.4 describes German for the purposes of translating to English.

3.3.3 Lexicalized Models

Brown et al. (1993) introduced "distortion" in Models 3 and 4 to model the permutation of words in the source language (theirs is a noisy channel model). Model 5 addresses the deficiency in Model 4, which allows "orderings" that are not permutations. These models include sophisticated conditioning, such as on classes of words. So-called *lexicalized* reordering models follow in this tradition by conditioning reordering probability on the words in the reordered phrases.

Tillmann (2004) introduced a simple "unigram" model that predicted whether the translation of a given phrase would occur to the right or to the left of its predecessor's translation in the target language ordering. It is a unigram model because the prediction is conditioned only on the given phrase, not on its predecessor. Tillmann showed that this model improved translation performance over the target language model alone.

Kumar and Byrne (2005) proposed a *jump*-based reordering model where the random variable in question, b_1^K, is the sequence of differences between each phrase's position in the source language phrase sequence and its position in the target language phrase sequence. The probability of each b_k is conditioned on the probability of b_{k-1}

and on the phrase table entry. It is thus a bigram jump model, but, like Tillmann's model, only considers phrase unigrams.

Al-Onaizan and Papineni (2006) proposed a related, but more sophisticated, model consisting of three parts:

Outbound Distortion predicts the position in the reordering of the next word relative to the position of the current word given the *current* word.

Inbound Distortion predicts the relative position given the *next* word.

Pairwise Distortion predicts the relative position given *both* the current and next words.

The first two are different ways of backing off from the third, which models the relative order of pairs of words. This model differs from the model this chapter proposes in two important ways:

- It predicts relative *position* of pairs of words, rather than just relative *ordering*—a binary variable.
- It only predicts the relative position of words that are adjacent in the source sentence, rather than all pairs of words.

Kuhn, Yuen, Simard, Paul, Foster, Joanis, and Johnson (2006) proposed "segment choice models", where *segment* is a synonym for *phrase* in the non-linguistic sense of phrase-based translation. They assumed the segmentation of the source language sentence into phrases was given, and modeled the relative order of those phrases in the target language. The probability of a phrase order decomposes into a sequence of decisions about the next phrase to include in a left-to-right ordering. They used decision trees (Breiman, Friedman, Olshen, and Stone, 1984) to assign probabilities. Their "features"—the questions the decision trees can ask—concern the relative positions of the phrases in the source sentence, their lengths, and the words they contain.

Xiong, Liu, and Lin (2006) proposed a constituent reordering model for a bracketing transduction grammar (BTG) (Wu, 1995). Their model predicts the probability that a pair of subconstituents will remain in order or reverse order when combined to form a new constituent. The features of their model look only at the *first* source and target words of each constituent. This could be considered a sparse version of the reordering model that this chapter proposes. The details of training and applying the models are quite different, however.

The model of Chang and Toutanova (2007) is altogether different. It predicts the position of each child relative to its parent using global features of a target dependency tree, obtained via projection from the source language. Their features consider target language words, source and target language parts of speech, and the displacements of target language words relative to the source sentence.

The default lexicalized reordering model for Moses, described in Koehn, Axelrod, Mayne, Callison-Burch, Osborne, and Talbot (2005), is similar to Tillmann's model. It distinguishes three different cases:

Monotone phrases are adjacent and in the same order in both the source and target languages.

Swap phrases are adjacent in both languages, but in opposite orders.

Discontinuous phrases are adjacent in one language but not the other.

The model assigns probability to each of the three possibilities given the phrase-table entries, and Moses uses the model in both the source-to-target and target-to-source directions.

This model is certainly an improvement over the distance-based distortion models of the previous section, and that improvement is reflected in improved performance, but it remains a *unigram* model. The probability that the current phrase occurs in monotone, swap, or discontinuous order relative to the previous phrase is independent of the identity of the previous phrase. The model that Section 3.6 introduces addresses this shortcoming.

3.4 German

Prior to introduction of this chapter's reordering model, it is appropriate to have a concrete notion of the types of reordering that the machine translation task is confronted with. This section describes aspects of the German language, with an emphasis on phenomena that relate to ordering differences from English.

English is typologically characterized as an SVO language, meaning the subject precedes the verb, and the object follows it. German, on the other hand, is SOV, meaning the verb comes after the object. This is the greatest typological distinction between the two languages, which in many respects remain quite similar. German's SOV label requires some refinement, however.

First, German retains case markings that English has lost, except on pronouns and possessives. English uses SVO order to indicate subject and object. German, on the other hand, has explicit nominative case marking to indicate the subject, and accusative and dative case markings to indicate different types of objects. As a result, word order is relatively less constrained in German. Sometimes the object appears before the subject of the same clause, as in

(1)	Das	muß	ich	erst	einmal	klären.
	That	must	I	first	once	resolve.
	"I will have to look into that."					

Second, the main finite verb appears in *second* position in declarative sentences. The element that appears before that verb may be a noun phrase, a prepositional phrase, an adverb, etc. It may be the subject of the main verb, its object, or neither. Besides the main finite verb, other parts of the verb, including separable prefixes, infinitives, past participles, etc., appear at the end of the clause. Everything else comes in between.

(2)

Wir	werden	das	überprüfen.
We	will	that	review.

"We will consider the matter."

(3)

Vielleicht	könnten	die	Kommission	oder	Sie	mir	einen	Punkt	erläutern.
Maybe	can	the	Commission	or	you	me	a	point	explain.

"Perhaps the Commission or you could clarify a point for me."

In general, subordinate clauses differ from the main clause in the lack of this second position requirement. The entire verb appears in the final position. It is in this sense that German is typologically verb-final.

(4)

Ich	kann	nicht	erkennen,	was	das	mit	dem	Protokoll	zu	tun	hat.
I	can	not	discern,	what	that	with	the	Minutes	to	do	has.

"I cannot see what that has to do with the Minutes."

Durrell (1997) gives the following basic word order plan for German:

1. *Vorfeld*—first position
2. Bracket^1—the first part of the verb
3. Pronouns
4. Noun subject
5. Dative noun
6. Most adverbials
7. Accusative noun
8. *nicht*—negative adverb
9. Adverbials of manner
10. Complements—genitive or prepositional objects
11. Bracket^2—the rest of the verb

In imperative sentences and questions, the *Vorfeld* is empty, and in subordinate clauses, Bracket^1 is. This shows that adverbs, including negation, will often require reordering to English as well.

(5) Ich kann diesen Vorschlag nicht nachvollziehen.
I can this proposal not comprehend.
"I cannot support this proposal."

Another important phenomenon is that of *separable prefix verbs*. The dictionary form is often written *prefix+verb*, and the prefix is usually orthographically identical to a preposition. For example, the infinitive *zustimmen*, meaning *to agree*, breaks down into the prefix *zu* and the verb *stimmen*, which alone means *to vote*. When inflected, the prefix separates from the verb and moves to the end of the clause. Effectively, these constitute compound words whose two parts can move arbitrarily far apart. Putting them back together is almost essential for correct translation. The last sentence in Table 3.9 on page 116 shows an example use of *zustimmen*.

Prepositional phrases and noun phrases are substantially similar between German and English. Prepositions precede determiners, which are followed by adjectives and then nouns. However, whereas English puts possessors before their possessions, and indicates the relationship with a clitic *'s*, German puts the possessor after, and uses genitive case marking, except with proper nouns. This is less problematic for reordering models for two reasons. First, the movement is not over a long distance. Second, English can preserve order by translating with *of*, rather than a possessive.

(6) Wir kommen nun zur Festsetzung des Arbeitsplans.
We come now to the determination of the work plan.
"The next item is the order of business."

Finally, German is notorious for compounding nouns. English also has noun-noun compounds, but often keeps them as separate words, especially with newly coined terms. Because German removes spaces, and the prefix noun often undergoes morphological changes, segmenting nouns into their components is non-trivial—cf. Schmid (2005). This is not a particular problem for reordering, but it is an essential part of translation from German. Many compound words in the test set are likely to be absent from the vocabulary of the phrase table. The methods presented here do not address this problem.

3.5 Translation with Finite-State Transducers

Finite-state automata are a computation-theoretic abstraction that have made the jump to practical algorithms because of their convenient properties. Finite-state acceptors (FSAs) have a one-to-one correspondence with regular expressions.[3] One of their most important properties is that, unlike context-free grammars, FSAs are closed under intersection.

Finite-state transducers (FSTs) are automata representing regular *relations*. They express not just one regular language, but two—an input and an output language. They can be thought of as accepting ordered pairs of strings, or as mapping one string to another, in either direction. Because of closure under intersection, so-called *composition* of transducers is possible.[4] This intersects the output language of one transducer with the input language of another, producing a third transducer that maps a subset of the first input language to a subset of the second output language.

Both FSAs and FSTs can be weighted without altering many of their closure properties, including intersection/composition, though some of the resulting algorithms require significantly more sophistication.[5] The resulting weighted finite-state transducers (WFSTs) represent weighted regular relations, where each pair of strings has a corresponding weight, which can be interpreted as a cost, a probability, etc., depending on the semiring (see Definition 3.3 ahead).

It is often convenient to devise complex WFSTs as *cascades* of simpler transducers that can be combined successively using composition. The intermediate languages can make use of abstract marker symbols that do not appear in either the ultimate input or output languages. The cascade of transducers that Section 3.5.1 constructs is an excellent example.

Formal presentation of WFSTs begins with two definitions.

Definition 3.3 *A* **semiring** *is a 5-tuple* $\langle K, \oplus, \otimes, \bar{0}, \bar{1}\rangle$ *with the following properties (Mohri, Pereira, and Riley, 2000):*

1. $\langle K, \oplus\rangle$ *is a commutative monoid with identity element* $\bar{0}$*,*

2. $\langle K, \otimes\rangle$ *is a (possibly non-commutative) monoid with identity element* $\bar{1}$*,*

3. $\otimes$ *distributes over* $\oplus$*, and*

4. $\bar{0}$ *is an annihilator, meaning* $\bar{0} \otimes w = w \otimes \bar{0} = \bar{0},\ \forall w \in K$.

Examples of semirings include:

- the real semiring $\langle \mathbb{R}, +, \cdot, 0, 1\rangle$,

[3]Meaning regular expressions of the computation-theoretic variety, consisting of the union, concatenation, and closure operations. Regular expressions as implemented in popular programming languages, such as perl, often have additional operators that make them non-finite-state.

[4]FSTs are *not* closed under intersection, meaning intersection of their regular relations. For example, if one transducer maps the language a^n to the language $a^n b^*$ while another maps a^n to $a^* b^n$, then their intersection maps a^n to $a^n b^n$, which is not regular.

[5]Cf. Mohri, Pereira, and Riley (1996); Mohri (2000, 2001); Eisner (2003).

- the log semiring $\langle \mathbb{R} \cup -\infty, \text{log+}, +, -\infty, 0 \rangle$, where

$$\text{log+}(v, w) \overset{\text{def}}{=} \log\left(e^v + e^w\right), \tag{3.3}$$

- the tropical semiring $\langle \mathbb{R} \cup \infty, \min, +, \infty, 0 \rangle$, and
- the expectation semiring (Eisner, 2001, 2002).

The expectation semiring will prove useful in Section 4.11.3 of the next chapter. For details, see Definition 4.2 on page 152.

Definition 3.4 *A* **weighted finite-state transducer** *(WFST) (Mohri, 1997) is a 7-tuple $\langle Q, \Sigma, \Delta, K, \delta, i, F \rangle$:*

1. *Q is the finite set of states.*
2. *Σ is the finite* **input** *or* **upper** *alphabet.*
3. *Δ is the finite* **output** *or* **lower** *alphabet.*
4. *$\langle K, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ is a semiring. The $\oplus$ operator is sometimes called* **collect***, because it aggregates the weights of multiple paths, and the $\otimes$ operator* **extend***, because it aggregates the weights along a single path.*
5. *$\delta : Q \times Q \times (\Sigma \cup \epsilon) \times (\Delta \cup \epsilon) \mapsto K$ is the transition function. The symbol ϵ indicates the empty string, which matches any position in a string without consuming any symbols. If $\delta(q, r, a, b) = w$ then there is an arc from state q to state r with input symbol a, output symbol b, and weight w. This is sometimes written*
$$q \xrightarrow{a:b/w} r. \tag{3.4}$$
If $\delta(q, r, a, b) = \bar{0}$ then there is no arc from q to r that accepts $a : b$.
6. *$i \in Q$ is a distinguished* **initial** *state.*
7. *$F : Q \mapsto K$ is the state* **final** *weight function. The set of* **final states** *is $\{q \in Q \mid F(q) \neq \bar{0}\}$.*

A weighted finite-state acceptor (WFSA) is defined identically, except it lacks the second alphabet Δ and δ has one parameter fewer. *Projection* converts a WFST into a WFSA by eliminating one of the alphabets, and the corresponding arc labels. A WFST can be projected onto either its input or its output language.

A WFST can be thought of as a function $T : \Sigma^* \times \Delta^* \mapsto K$ from pairs of strings to weights. Likewise, a WFSA can be thought of as a function $A : \Sigma^* \mapsto K$ from strings to weights. For $x \in \Sigma^*$, A *accepts* x if $A(x) \neq \bar{0}$. In general, there may be multiple paths in the transducer matching $x : y$, or in the acceptor matching x. $T(x, y)$ and

Symbol	**Name**	**Meaning**
$\mathbf{e}$	English	the source sentence, in the noisy channel formulation
$\mathbf{f}$	French	the target sentence, in the noisy channel formulation, which is given
$\boldsymbol{\phi}$	fertility	the number of French words to which each English word aligns
$\boldsymbol{\tau}$	tableau	the list of French words to which each English word aligns
$\boldsymbol{\pi}$	permutation	the French word order
$\mathbf{a}$	alignment	the combination of τ and π

Table 3.1: Random variables from IBM Model 4.

$A(x)$ compute the semiring sum of the weights of all matching paths. In the tropical semiring, this is simply the weight of the best path.

Composition of two transducers T_1 and T_2 produces a third transducer $T_1 \circ T_2$, such that

$$(T_1 \circ T_2)(x, z) = \bigoplus_{y \in \Delta^*} T_1(x, y) \otimes T_2(y, z), \tag{3.5}$$

where Δ is both the output alphabet of T_1 and the input alphabet of T_2.[6]

The remainder of this section explores WFST implementations of standard translation models. The formulations described here sometimes include a reordering component. Section 3.6, in contrast, introduces a model where the WFST only performs monotonic translation. Search for a good ordering takes place outside the automaton, and incorporates scores from other models of reordering, including the linear ordering problem.

3.5.1 IBM Model 4

Knight and Al-Onaizan (1998) first proposed translation with transducers. They discussed how to formulate IBM Models 1–3 as WFSTs. They also proposed alternatives to Model 3 with simpler reordering models. This section extends that work by encoding IBM Model 4 as a WFST. Section 3.5.2 turns to phrase-based translation models.

IBM Model 4 (Brown et al., 1993) models the random variables shown in Table 3.1 using the following formulae and assumptions, and the parameters from Table 3.2. The exposition uses *French* and *English* to respectively name the target and source languages of the noisy channel, following their precedent. The model is obviously not

[6]In general, the output alphabet of T_1 and the input alphabet of T_2 may differ, so long as they share some symbols. In that case, Δ is the union of the two alphabets.

Notation	**Meaning**
$n(\phi \mid e)$	fertility probability
p_0, p_1	NULL fertility probabilities
$t(f \mid e)$	word translation probability
$d_1(\Delta j \mid \mathcal{A}, \mathcal{B})$	distortion probability for the first word aligned to each English word
$d_{>1}(\Delta j \mid \mathcal{B})$	distortion probability for other words aligned to each English word

Table 3.2: Parameters of IBM Model 4.

limited to this language pair.

$$
\begin{aligned}
P_\theta(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}) &= P_\theta(\boldsymbol{\phi} \mid \mathbf{e}) P_\theta(\boldsymbol{\tau} \mid \boldsymbol{\phi}, \mathbf{e}) P_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \boldsymbol{\phi}, \mathbf{e}), && (3.6)\\
P_\theta(\mathbf{f}, \mathbf{a} \mid \mathbf{e}) &= \sum_{(\boldsymbol{\tau}, \boldsymbol{\pi}) \in \langle \mathbf{f}, \mathbf{a} \rangle} P_\theta(\boldsymbol{\tau}, \boldsymbol{\pi} \mid \mathbf{e}), && (3.7)\\
P_\theta(\boldsymbol{\phi} \mid \mathbf{e}) &= n_0 \left(\phi_0 \mid \sum_{i=1}^{l} \phi_i \right) \prod_{i=1}^{l} n(\phi_i \mid e_i), && (3.8)\\
P_\theta(\boldsymbol{\tau} \mid \boldsymbol{\phi}, \mathbf{e}) &= \prod_{i=0}^{l} \prod_{k=1}^{\phi_i} t(\tau_{ik} \mid e_i), && (3.9)\\
P_\theta(\boldsymbol{\pi} \mid \boldsymbol{\tau}, \boldsymbol{\phi}, \mathbf{e}) &= \frac{1}{\phi_0!} \prod_{i=1}^{l} \prod_{k=1}^{\phi_i} p_{ik}(\pi_{ik}), && (3.10)\\
n_0(\phi_0 \mid m') &= \binom{m'}{\phi_0} p_0^{m'-\phi_0} p_1^{\phi_0}, && (3.11)\\
p_{ik}(j) &= \begin{cases} d_1(j - c_{\rho_i} \mid \mathcal{A}(e_{\rho_i}), \mathcal{B}(\tau_{i1})) & \text{if } k = 1,\\ d_{>1}(j - \pi_{ik-1} \mid \mathcal{B}(\tau_{ik})) & \text{if } k > 1, \end{cases} && (3.12)\\
\rho_i &= \max_{i' < i} \{ i' : \phi_{i'} > 0 \}, && (3.13)\\
c_\rho &= \left\lceil \phi_\rho^{-1} \sum_{k=1}^{\phi_\rho} \pi_{\rho k} \right\rceil . && (3.14)
\end{aligned}
$$

In order to map this generative process onto finite-state transducers, it will be convenient to further subdivide it. The resulting cascade of transducers can combine through composition to produce a single model. The formulation given below assumes that the French sentence is given. This assumption is necessary because of the distributions and dependencies used in Model 4, particularly (3.10), (3.11), and (3.12), which require counts.

The first step, because (3.8) depends on it, is to choose fertilities for all of the

English words. To model fertility, let each English word transduce to itself an appropriate number of times. For later use, preface any English word with fertility greater than zero with a special bracket symbol indexed by the class of that word. Thus, the sequence

$$e_1 \; e_2 \; e_3 \; e_4$$

would become

$$[_{\mathcal{A}(e_1)} \; e_1 \; e_1 \; [_{\mathcal{A}(e_2)} \; e_2 \; [_{\mathcal{A}(e_4)} \; e_4 \; e_4 \; e_4$$

with probability

$$n(2 \mid e_1) \cdot n(1 \mid e_2) \cdot n(0 \mid e_3) \cdot n(3 \mid e_4).$$

The structure of this transducer depends on the structure of each $n(\cdot \mid \cdot)$ distribution. If the distributions are geometric, then the transducer needs only one state for each word. If they are arbitrary multinomial distributions, then the transducer needs one state for each word/fertility combination. The number of states mirrors the number of parameters of the distributions.[7]

To complete (3.8), compose the first transducer with one that optionally inserts the symbol $e_0 = \text{NULL}$ after each instance of an English word with probability p_1, and doesn't do so with probability p_0. This would, for example, turn

$$[_{\mathcal{A}(e_1)} \; e_1 \; e_1 \; [_{\mathcal{A}(e_2)} \; e_2 \; [_{\mathcal{A}(e_4)} \; e_4 \; e_4 \; e_4$$

into

$$[_{\mathcal{A}(e_1)} \; e_1 \; \text{NULL} \; e_1 \; [_{\mathcal{A}(e_2)} \; e_2 \; [_{\mathcal{A}(e_4)} \; e_4 \; e_4 \; \text{NULL} \; e_4$$

with probability

$$p_1 \cdot p_0 \cdot p_0 \cdot p_0 \cdot p_1 \cdot p_0.$$

This accounts for most of (3.11). In order to handle the $\binom{m'}{\phi_0}$ term, it suffices to use the states to count ϕ_0, since $m' = m - \phi_0$, where m is the given total number of French words. Using the notation of Definition 3.4, the final weight of state q_ϕ then is $F(q_\phi) = \binom{m-\phi}{\phi}$.

The next step is to translate the English words, including NULL, according to (3.9). A one-state transducer accomplishes this, and augments the French word symbols with their positions in the French string, information that the next step will use. So, for example, this might transduce

$$[_{\mathcal{A}(e_1)} \; e_1 \; \text{NULL} \; e_1 \; [_{\mathcal{A}(e_2)} \; e_2 \; [_{\mathcal{A}(e_4)} \; e_4 \; e_4 \; \text{NULL} \; e_4$$

[7]This is not true of discrete probability distributions in general. For example, the Poisson distribution, with the single parameter λ,

$$P(X = x) = e^{-\lambda}\frac{\lambda^x}{x!}, \tag{3.15}$$

requires an arbitrary number of states to correctly compute the denominator. Fortunately, the number of English words is bounded by the number of French words, so this is still finite-state *given* the French sentence.

into

$$[_{\mathcal{A}(e_1)}\ f_5\ \textsc{null}(f_8)\ f_6\ [_{\mathcal{A}(e_2)}\ f_1\ [_{\mathcal{A}(e_4)}\ f_4\ f_2\ \textsc{null}(f_3)\ f_7$$

with probability

$$t(f_5 \mid e_1) \cdot t(f_8 \mid \textsc{null}) \cdot t(f_6 \mid e_1) \cdot t(f_1 \mid e_2) \cdot t(f_4 \mid e_4) \cdot t(f_2 \mid e_4) \cdot t(f_3 \mid \textsc{null}) \cdot t(f_7 \mid e_4).$$

Note that this transducer does not account for the positions of the French words, it merely assigns them. In practice, the French words and their positions are given, it is the English words that are unknown. This transducer can therefore map each French word position to all non-zero English entries in the translation table for that word.

Next, score the permutation implied by the indices on the French words (namely, the one that would result in the sequence $f_1\ f_2\ \dots\ f_8$) according to the distortion probability (3.10). First, there is the $\frac{1}{\phi_0!}$ term that accounts for the many possible permutations of those French words generated by NULL. This is why the mark-up of NULL words is needed. The example sequence remains unchanged by this step but is assigned probability $\frac{1}{2}$ since there are 2 instances of NULL. This automaton is such that the kth instance of NULL gets probability $\frac{1}{k}$, and needs as many states as there are possible NULL-aligned French words.[8]

Another acceptor incorporates the $d_{>1}$ probabilities. This assigns the example sequence probability

$$d_{>1}(1 \mid \mathcal{B}(f_6)) \cdot d_{>1}(-2 \mid \mathcal{B}(f_2)) \cdot d_{>1}(5 \mid \mathcal{B}(f_7)).$$

It does so by ignoring NULL-generated words and skipping over word boundaries separated by a bracket.

The last step incorporates the d_1 probabilities of the words immediately following brackets. It assigns the example sequence probability

$$d_1(5 \mid \mathcal{A}(?), \mathcal{B}(f_5)) \cdot d_1(-5 \mid \mathcal{A}(e_1), \mathcal{B}(f_1)) \cdot d_1(3 \mid \mathcal{A}(e_2), \mathcal{B}(f_4)).$$

This is the most challenging part for finite-state modeling, because the distribution d_1 depends on c_{ρ_i}. In words, ρ_i is the index of the previous English word that generates at least one French word, and c_ρ is the average position of French words aligned to e_ρ. To accomplish this computation, the automaton must count the non-NULL words between the brackets and sum their positions.

Index the states of the automaton with an ordered triple (ℓ, s, a), where ℓ counts the number of words since the most recent bracket, s counts the sum of word positions, and a is the class of the most recent English word. The arc that matches French word f_j from state (ℓ, s, a) goes to state $(\ell+1, s+j, a)$. Self-loops match NULL words. Arcs that match brackets $[_{a'}$ go from state (ℓ, s, a) to a special state labeled $(\lceil \frac{s}{\ell} \rceil, a, a')$. Each special state (c, a, a') has an arc matching each f_j with the appropriate weight

[8]Model 4 allows at most *half* the French words to be aligned to NULL.

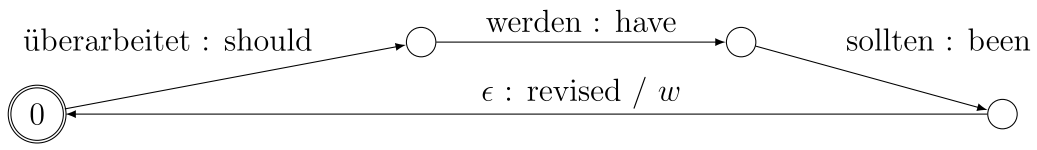

Figure 3.2: One path of a transducer representing the phrase table. Each phrase table entry forms a weighted loop from the distinguished state 0. Arcs without explicit weights have weight zero. Note that, because of within-phrase reordering, there is no lexical correspondence along the arcs. The alignment within the phrase is *überarbeitet—revised*, *werden—have been*, *sollten—should*.

$d_1(j - c \mid a, \mathcal{B}(f_j))$ that goes to state $(1, j, a')$. This automaton needs $O(m^3 |\mathcal{A}| + m |\mathcal{A}|^2)$ states, where m is the number of French words.

Finally, remove the brackets and the NULL annotations to produce the reordered French string annotated with its word positions. Notice that this cascade of transducers does *not* generate the French string in French order. That is the job of the reordering search, as Section 3.6 will explain.

3.5.2 Phrase-Based Translation

Kumar and Byrne (2003) showed how to implement the Alignment Template Translation Model of Och and Ney (2004) with finite-state transducers. Their implementation consists of a cascade of six automata:

1. Source sentence: a straight-line FSA with $n + 1$ states and n arcs.
2. Source segmentation model: a WFST that maps from the alphabet of words to an alphabet that has a different symbol for each phrase.
3. Phrase permutation model: a WFST that allows a subset of the possible permutations of the available phrases.
4. Template sequence model: a WFST that maps source language phrases in target language order to alignment templates.
5. Phrasal translation model: a WFST that converts alignment templates to target language words. It is itself a composition of several simpler automata.
6. Target language model: a WFSA that computes the probability of the target sentence.

Koehn, Och, and Marcu (2003) proposed a simpler phrase-based model, consisting of four parts:

1. a phrase translation table,

2. a distortion model,

3. a word penalty, and

4. a language model.

The translation table can be implemented simply after the example of Figure 3.2. The WFST has a single final state—the initial state 0. Each phrase table entry corresponds to a path from state 0 back to state 0 that accepts the appropriate source and target language word sequences with the appropriate weight from the table. The experiments reported here use five features for each phrase table entry:

- source-to-target phrase translation probability,
- target-to-source phrase translation probability,
- source-to-target word translation probabilities,
- target-to-source word translation probabilities, and
- phrase count—each entry has value 1 for this feature.

The word penalty, which biases the model to produce longer or shorter output, can also be incorporated into the phrase table weight, as each phrase table entry knows how many target words it generates.

Conveniently, a simple distance-based distortion model can be implemented as an n-state WFSA without even encoding the phrase segmentation. There is a transition from each state i to each other state j with label j and weight $\alpha^{|j-i-1|}$, so that if $j = i + 1$ the weight is one. Every source word sequence that will be used as a phrase will thus incur no cost within the phrase. The only costs come at non-contiguous phrase boundaries.[9]

Finally, the N-gram language model can be implemented as a WFSA in the standard way (Mohri, 1997). The experiments in later sections use finite-state models built according to this description. The weights of each of these models is tuned on the development data set using minimum error-rate training (MERT) (Och, 2003).

3.6 An ABC Ordering Model

This section describes a novel reordering model that combines three different sources of information.[10] Although the model is formulated here exclusively as a

[9]Looking forward to Section 3.6, this WFSA would be composed onto the cascade in *front* of the FST W that maps positions to words. It is also not unreasonable to throw away this distortion model and rely on B and C to provide adequate models of reordering.

[10]This model, and some of the other work of the dissertation, appeared in Eisner and Tromble (2006).

reordering model, it can also be considered a translation model, because the finite-state component can be a transducer. The components are:

A: a weighted finite-state acceptor (WFSA),

B: a linear ordering problem matrix, and

C: a three-dimensional cost matrix scoring triples of indices, analogous to the linear ordering problem, and detailed in Section 3.7.

Each of these models assigns a value to every permutation $\pi \in \Pi_n$, where n is the length in words[11] of the source sentence. The score of a permutation under this ABC model is $A(\pi) + B(\pi) + C(\pi)$.

The weighted acceptor A has $\Sigma = \{1, 2, \ldots, n\}$. It would be awkward and inefficient to limit A to accepting Π_n—assigning weight $\bar{0}$ to $\Sigma^* \setminus \Pi_n$. Instead, it is practical to allow A to accept any string and to restrict the search algorithms to consider only permutations.

Section 3.5 describes how to encode translation models as weighted finite-state transducers (WFSTs). In order to convert such a transducer into a finite-state acceptor whose language is Π_n, compose on the source side with an unweighted one-state transducer W that maps source language positions to the words that occupy those positions, then project the resulting transducer to its (weighted) input language to get A. The weight A assigns to a permutation π is then identical to the weight of the best path through the transducer that has $W(\pi)$ as its input.[12]

To perform translation, first search for the best ordering π^*—finding the best permutation is NP-complete in general under each of these models *separately*, so some non-optimal search, such as Section 2.5 described, is necessary for efficiency. Given some good permutation $\hat{\pi}$, find the best transduction of $\hat{\pi}$ by the WFST from which A was derived.

The trick here is that the projection of the WFST onto A, assuming a max-based semiring, assigns the same weight to a permutation π as the WFST assigns to the pair of strings $W(\pi)$ and its best translation. In a sense, A looks ahead to the transduction step without actually performing it.

The problem of finding the best permutation according to $A(\pi)$ is related to classical path algorithms. If A has the structure of a bigram language model, then there is a one-to-one correspondence between the *states* of A and the items in the permutation. In this case, because a permutation necessarily visits each state once, finding the best permutation is a weighted Hamiltonian path problem, a close relative of the traveling salesman problem.

[11] For some languages, the *word* may not be the appropriate unit for reordering. This dissertation uses *word* for transparency, where *token* or some other concept might be more appropriate.

[12] Here, $W(\pi)$ means the unique string to which the unweighted transducer W maps the permutation π.

In general, however, the automaton A has unlabeled nodes and only *edges* labeled with indices, corresponding to the words of the source sentence. The problem is to find a path through A that traverses exactly one edge labeled with each index, and an arbitrary number of edges labeled ϵ, the empty string.

The B model is the primary focus of this dissertation. The linear ordering problem is the topic of Chapter 2, while Chapter 4, particularly Section 4.4, describes models for generating LOP matrices from source language sentences. Both the B model and specific instances of the A model have been extensively studied on their own, but never before in combination.

The C model remains speculative—the experiments this chapter reports do not use it at all. Still, it is worth considering here briefly. Section 3.7 discusses it.

3.6.1 Discussion

The A model, because it is finite-state, is best suited to modeling the local structure of the permutation. It is the logical place to incorporate an n-gram target language model, as well as noisy-channel reordering models like those presented in Section 3.3. That is, most of those models predict what happens to adjacent words or phrases, and the A model sees what is adjacent in the *target* language, therefore reordering models that predict source given target are appropriate.

The B model, on the other hand, pays no attention to adjacency, but only considers relative ordering. It can incorporate some of the reordering models from Section 3.3 in the source-to-target direction, though not those that predict specific distances. B can incorporate such predictions not just for adjacent words in the source sentence, but for all pairs of words.

One limitation of the B model is that it doesn't pay attention to phrases. If a phrase segmentation is given, as with the segment choice model of Kuhn et al. (2006), then the phrases can act as the units of movement, instead of words. However, if the phrase segmentation is non-deterministic, as in standard phrase-based translation, then B leaves phrase coherency to the A model.

It is possible, though, to introduce additional indices into the permutation corresponding to, e.g., phrase brackets in the source sentence. Eisner and Tromble (2006) proposed symbols such as $[_{\mathrm{NP4}}$ and $]_{\mathrm{NP4}}$ for indicating the endpoints of the fourth noun phrase in the sentence. The following additions to the B matrix would be appropriate:

- The score for moving the right bracket before its corresponding left bracket should be a large negative value, even $-\infty$.

- Likewise the score for moving a word from within the phrase either before the left bracket or after the right bracket. However, this need not be a hard constraint.

- Words outside the phrase that should reorder with respect to it can use the phrase brackets as reference points—moving before the phrase means moving before the left bracket, and moving after the phrase means moving after the right bracket.

Unless this system can accommodate some nondeterminism, it is just a more expensive way of accomplishing the word-to-phrase transformation mentioned above. Therefore, allowing words to move outside their phrases at some cost is a reasonable possibility. That cost should derive from features of the phrase, including the confidence of the parser that bracketed it.

One potentially important difference between the B model and many of the reordering models of Section 3.3 is that the B model is capable of assigning a distinct score to every possible permutation of the source sentence. Some of those models can accomplish the same thing by considering the probability in both source-to-target and target-to-source directions. Nonetheless, if a pair of words or phrases is adjacent in neither the source nor the target ordering, then most of the models described above only "care" about that pair via second-order effects. The B model considers their relative order *directly*.

For example, the verb and its object noun may be separated in both the source and target sentence by other words, including adverbs or other parts of the noun phrase, such as adjectives. Nonetheless, English order will usually require the verb to precede that noun. The B model can account for this with an appropriate matrix entry.

3.7 Cyclic Costs

This section details the C model, novel to Eisner and Tromble (2006). Given an $n \times n \times n$ table C, define $C(\pi)$ as follows:

$$C(\pi) \overset{\text{def}}{=} \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} C[\pi_i, \pi_j, \pi_k]. \tag{3.16}$$

This is an analog of the linear ordering problem definition (2.2), scoring ordered triples instead of pairs.

Remarkably, the efficient dynamic programming of Chapter 2 goes through for the C model if the table satisfies a special *cyclic* structure:[13]

$$C[\pi_i, \pi_j, \pi_k] = C[\pi_j, \pi_k, \pi_i] = C[\pi_k, \pi_i, \pi_j]. \tag{3.17}$$

While this is certainly not as general as an unconstrained table, it still provides a potentially useful distinction—*even* permutations of i, j, and k receive one score,

[13]The appendix of Eisner and Tromble (2006) works around this constraint by adding additional symbols to the permutation.

while *odd* permutations receive a different score. Further, in conjunction with the B model, there are a total of four score differences to manipulate for each triple. This is not as general as the six possible scores in an unconstrained C model, but the efficiency gained may be worth the compromise.

The four score differences for each triple i, j, k are:

$$\begin{aligned} C[\pi_i, \pi_j, \pi_k] - C[\pi_k, \pi_j, \pi_i]&, \\ B[\pi_i, \pi_j] - B[\pi_j, \pi_i]&, \\ B[\pi_i, \pi_k] - B[\pi_k, \pi_i]&, \text{ and} \\ B[\pi_j, \pi_k] - B[\pi_k, \pi_j]&. \end{aligned}$$

These four differences allow $(B+C)(\pi)$ to differentiate between different even and odd permutations of π_i, π_j, and π_k. For example, the difference between $(B+C)(\pi_i\ \pi_j\ \pi_k)$ and $(B+C)(\pi_k\ \pi_i\ \pi_j)$ is

$$(B[\pi_i, \pi_k] - B[\pi_k, \pi_i]) + (B[\pi_j, \pi_k] - B[\pi_k, \pi_j])\,.$$

Because each B model difference is shared by $\Theta(n)$ triples, though, combining B with a cyclicly-constrained C model is still far from approximating an unconstrained C model, in general.

The reason for the cyclicity requirement is that, at the time that the span (i, j) is combined with the span (j, k), the relative order of every $\ell \in (i, j)$ and every $r \in (j, k)$ is known. (This was the insight for the linear ordering problem.) The relative order of every $o \in (0, n) \setminus (i, k)$ with respect to ℓ and r, on the other hand, is unknown—it could be $o \prec \ell, r$ or $\ell, r \prec o$. However, if the table has the cyclic property (3.17), then those scores are the same.

The grammar weights associated with the C score are defined as follows:

$$\overrightarrow{\gamma}_{i,j,k} \stackrel{\text{def}}{=} \sum_{\ell=i+1}^{j} \sum_{r=j+1}^{k} \left(\sum_{o=1}^{i} C[\pi_o, \pi_\ell, \pi_r] + \sum_{o=k+1}^{n} C[\pi_\ell, \pi_r, \pi_o] \right) \tag{3.18}$$

$$\overleftarrow{\gamma}_{i,j,k} \stackrel{\text{def}}{=} \sum_{\ell=i+1}^{j} \sum_{r=j+1}^{k} \left(\sum_{o=1}^{i} C[\pi_o, \pi_r, \pi_\ell] + \sum_{o=k+1}^{n} C[\pi_r, \pi_\ell, \pi_o] \right) \tag{3.19}$$

Figure 3.3 shows a modification of the dynamic program from Figure 2.26 on page 48 that computes the value $C(\pi)$ instead of $B(\pi)$. Include the $B[\cdot, \cdot]$ terms from Figure 2.26 to compute $B(\pi) + C(\pi)$ simultaneously.

This dynamic program appears to increase the complexity of the one for B—it computes $\Theta(n^3)$ quantities, and each appears to require three $\Theta(n)$ sums to compute its value. Fortunately, the three sums on the right hand side are analogous to B matrix entries—there are only $\Theta(n^2)$ of them. Each can be computed in $\Theta(n)$ time during preprocessing, and the results stored in a matrix. This optimization keeps the total runtime at $\Theta(n^3)$.

$$\begin{aligned}
\vec{\gamma}_{i,i,k} &= 0 \\
\vec{\gamma}_{i,k,k} &= 0 \\
\vec{\gamma}_{i,j,k} &= \vec{\gamma}_{i,j,k-1} + \vec{\gamma}_{i+1,j,k} - \vec{\gamma}_{i+1,j,k-1} \\
&\quad - \sum_{m=i+2}^{k-1} C[\pi_{i+1}, \pi_m, \pi_k] + \sum_{\ell=1}^{i} C[\pi_\ell, \pi_{i+1}, \pi_k] + \sum_{r=k+1}^{n} C[\pi_{i+1}, \pi_k, \pi_r] \\
\overleftarrow{\gamma}_{i,i,k} &= 0 \\
\overleftarrow{\gamma}_{i,k,k} &= 0 \\
\overleftarrow{\gamma}_{i,j,k} &= \overleftarrow{\gamma}_{i,j,k-1} + \overleftarrow{\gamma}_{i+1,j,k} - \overleftarrow{\gamma}_{i+1,j,k-1} \\
&\quad - \sum_{m=i+2}^{k-1} C[\pi_k, \pi_m, \pi_{i+1}] + \sum_{\ell=1}^{i} C[\pi_\ell, \pi_k, \pi_{i+1}] + \sum_{r=k+1}^{n} C[\pi_k, \pi_{i+1}, \pi_r]
\end{aligned}$$

Figure 3.3: A dynamic program for computing grammar rule weights under the C model. These equations rely on the cyclic property (3.17) for correctness.

Theorem 3.1 *Let T be a binary permutation tree expressing the permutation π. If each in-order node in T combining (i,j) and (j,k) to span (i,k) receives the score (3.18), and each reverse node in T combining (i,j) and (j,k) to span (i,k) receives the score (3.19), then the sum of the scores of all nodes in the tree is $C(\pi)$, assuming C has the cyclic property (3.17).*

The proof is more complex than the corresponding proof for the LOP Theorem 2.1 because the grammar scores include items outside their spans.

Proof: $C(\pi)$ is the sum over all ordered triples $\ell \prec m \prec r$ in π of $C[\ell, m, r]$. It suffices to show that each such table entry is included in exactly one grammar score.

Each internal node in the tree covers some span (i,k) and combines two child spans (i,j) and (j,k). If (i,j) and (j,k) are kept in order, then the grammar score for (i,k) is $\vec{\gamma}_{i,j,k}$, which includes all table costs of the form $C[\ell \in (i,j), m \in (j,k), r \notin (i,k)]$. If (i,j) and (j,k) are reversed, then the grammar score is $\overleftarrow{\gamma}_{i,j,k}$, which includes all table costs of the form $C[\ell \in (j,k), m \in (i,j), r \notin (i,k)]$. It further suffices to show that for each triple $\ell \prec m \prec r$, there is exactly one node in the tree such that one of ℓ, m, r is in the span (i,j) of its left child, one is in the span (j,k) of its right child, and one is outside its span (i,k).

For every triple $\ell \prec m \prec r$ in π, there is a minimal node governing both ℓ and m—call it L—and a different minimal node governing both m and r—call it R. L and R *cannot* be the same node, because m falls between ℓ and r in the permutation. Therefore, either L governs R, or vice versa.

Suppose L governs R. Then call the span of R (i, k), the span of its left child (i, j), and the span of its right child (j, k). Either $m \in (i, j)$, $r \in (j, k)$, and R is in-order, or $m \in (j, k)$, $r \in (i, j)$, and R is reversed. In the first case, the score at R is $\vec{\gamma}_{i,j,k}$, which includes either $C[\ell, m, r]$ or $C[m, r, \ell]$, equal by the assumption of the cyclic property. In the second case, the score at R is $\overleftarrow{\gamma}_{i,j,k}$, which includes one of the same two scores.

Suppose R governs L. Then call the span of L (i, k), the span of its left child (i, j), and the span of its right child (j, k). Either $\ell \in (i, j)$, $m \in (j, k)$, and L is in-order, or $\ell \in (j, k)$, $m \in (i, j)$, and L is reversed. In the first case, the score at L is $\vec{\gamma}_{i,j,k}$, which includes either $C[r, \ell, m]$ or $C[\ell, m, r]$, equal by the assumption of the cyclic property. In the second case, the score at L is $\overleftarrow{\gamma}_{i,j,k}$, which includes one of the same two scores.

Therefore, there exists a unique node whose grammar score includes a score equal to $C[\ell, m, r]$, and the sum of all grammar scores in T is $C(\pi)$. □

3.8 Reordered Source Language

One prior approach to reordering for machine translation involves preprocessing the source language to make it easier to translate into the target language, but to continue to use standard tools for the actual translation step. Xia and McCord (2004) improved English to French translation using syntactic rewrite rules derived from Slot Grammar parses. Costa-jussà and Fonollosa (2006) improved Spanish-English and Chinese-English translation using a two-step translation process. The first step used the same translation models as the second step, but merely reordered the source language words.

Collins, Koehn, and Kučerová (2005) reported an improvement from 25.2% to 26.8% BLEU score on German-English translation using six different hand-written rules to reorder German sentences using automatically generated phrase-structure parse trees. This section reports the results of similar experiments using fully automatic reordering models that build instances of the linear ordering problem for each source language sentence based on automatically assigned part of speech tags.

3.8.1 Word Alignment

The first experiment uses an oracle English ordering for the German sentences derived from unweighted symmetrized alignments of parallel sentences in the corpus. That is, GIZA++ (Och and Ney, 2003) produced lexical alignments for both the German-English and the English-German translation direction, and those were combined using the "grow-diag-final" heuristic. Figure 3.4 shows an example of such an alignment.

The oracle English ordering was derived from these alignments as follows:

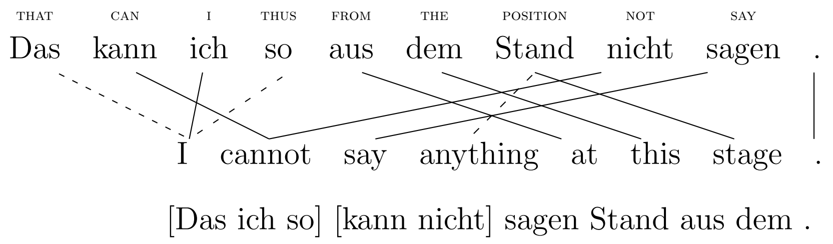
THAT CAN I THUS FROM THE POSITION NOT SAY
Das kann ich so aus dem Stand nicht sagen .
I cannot say anything at this stage .
[Das ich so] [kann nicht] sagen Stand aus dem .

Figure 3.4: An automatically-generated alignment. Links shown with dashed lines are poor alignments. The German′ derived under Oracle 1 for this sentence appears below the English. Square brackets indicate multiple German words with the same left-most English alignment.

- Each German word was mapped to the position of the leftmost English word to which it was aligned.
- If a German word was not aligned to any English word, it was mapped to the position to the left of all other German words.
- Ties were broken by preserving left-to-right order of the German, so that if two German words were both mapped to the same English position, the one that occurred earlier in the German sentence also occurred first in English order.

This is similar to the oracle ordering used by Al-Onaizan and Papineni (2006). In their scheme, which puts the target-language sentence in source-language order,

> If a target word is not aligned, then, we assume that it is aligned to the same source word that the preceding aligned target word is aligned to.

The second experiment used an oracle English ordering derived from alignments computed in only one direction, from English to German, with null alignments disallowed, so that the alignment was many-to-one in the German-to-English direction.

- Each German word was mapped to the position of the unique English word to which it was aligned.
- Ties were broken by preserving left-to-right order of the German words.

A third experiment used alignments computed using the BerkeleyAligner. This produced different alignments from GIZA++, but the procedure for constructing an English order was identical to Oracle 2. These alignments also disallowed nulls, so that every German word had at least one corresponding English word.

3.8.2 Corpus

The experiments in this section began with the training section of the German-English portion of the Europarl corpus (Koehn, 2005), consisting of 751,088 sentence pairs. From this were extracted two random samples of size 2000 sentence pairs each for use as a development corpus and an evaluation corpus, respectively. The remaining 747,088 sentences comprise the training set.

3.8.3 Method

Given a linear model, as in (2.3), from which to construct a linear ordering problem for each German sentence, the experiments of this section use the following procedure:

1. Reorder each training, development, and evaluation sentence using search for the linear ordering problem. Search was limited to the exponential-size neighborhood BlockInsert$^*_n(I_n)$, where I_n is the identity permutation. See Section 4.12.3 for an explanation of this constraint.

2. Align and extract phrases from the training portion of the reordered data.

3. Tune Moses parameters using minimum error-rate training on the development portion of the reordered data.

4. Decode the evaluation portion of the corpus using the tuned Moses parameters.

5. Use automatic evaluation to measure the resulting translation performance.

Unless otherwise specified, Moses used standard settings, including a bi-directional lexicalized reordering model and a distortion limit of 6. Some of the experiments used a distance-based reordering model.

3.8.4 Results

Table 3.3 shows the phrase table size and translation performance for different German orderings. The first section is the original German, the second section used models learned via the techniques of Chapter 4, and the remainder are oracle orderings as described in Section 3.8.1. The BLEU score improvements between the original German and reordering using the LOP model, though apparently small, are statistically significant according to the tests described in Section A.2 of Appendix A.

Additionally, it is worth observing that translating LOP POS 1 with the simple distance-based reordering within Moses outperforms the translation of unreordered German using a lexicalized reordering model in Moses. This BLEU score difference is also significant according to both the sign and paired permutation tests.

The phrase table statistics are interesting in their own right. Moses extracts a phrase consisting of German words g_i–g_j and English words e_k–e_ℓ only if all the

Source Order	**|Phrase Table|**	**Reordering**	**BLEU**	**METEOR**	**TER**
German	32,746,913	Distance	25.27	54.03	60.60
		Lexical	25.55	54.18	59.76
LOP POS 1	35,948,638	Distance	26.03	54.62	59.39
		Lexical	26.25	54.49	59.67
LOP POS 2	35,991,596	Lexical	26.35	54.66	59.39
LOP POS+Word 2	36,888,806	Lexical	26.44	54.61	59.23
LOP POS+Word 3	36,417,828	Lexical	26.10	54.42	59.68
Oracle 1	44,463,584	Distance	32.96	57.67	50.72
		Lexical	32.86	57.61	51.05
Oracle 2	44,457,955	Lexical	32.16	57.40	52.23
Oracle 3	47,476,788	Lexical	34.70	57.88	53.96

Table 3.3: German-English translation performance and other statistics of several systematic reorderings. Higher BLEU and METEOR and lower TER is better. For the LOP models, the second part of the name is the feature sets, and the number corresponds to the oracle used for training.

alignments of words in the German side are to words in the English side, and vice versa. The closer the alignments become to monotone, the more phrases satisfy this requirement. Hence the more phrases, by *token*, Moses finds to extract.

The sizes of the tables as given, though, is in terms of *types*—the number of *distinct* phrase pairs. To some extent, normalization of word order might be expected to *reduce* the number of phrase types. It is not clear whether this happens as well. Figure 3.5 shows phrase lengths for several reorderings.

Oracle 2 was intended to overcome the problem of null-aligned words faced by Oracle 1. Table 3.3 shows that this oracle ordering is not as good as the original oracle, across all three evaluation measures. This is probably related to the fact that the alignments are only computed in one direction. However, the LOP model trained using Oracle 2 rather than Oracle 1 performs better, again according to all three measures. This suggests that though Oracle 1 is itself a better ordering, it is harder to learn than Oracle 2, which is closer to the original German ordering, according to monolingual BLEU.

The difference between the BLEU scores of Oracle 1 and Oracle 2 using the lexicalized reordering model, from 32.86 to 32.16, is significant at $p < 0.01$ according to both the paired permutation test and the sign test. The difference between the LOP models, from 26.25 to 26.35, is not significant at $p < 0.05$ according to either test.

Table 3.4 shows the five sentences that improve the BLEU score most when the translated German is replaced by translated German′ from the LOP POS 2 model. Table 3.6 shows the corresponding German, and the reordered German′. Similarly, Table 3.7 shows the five with the largest negative effect, and Table 3.9 shows their corresponding German.

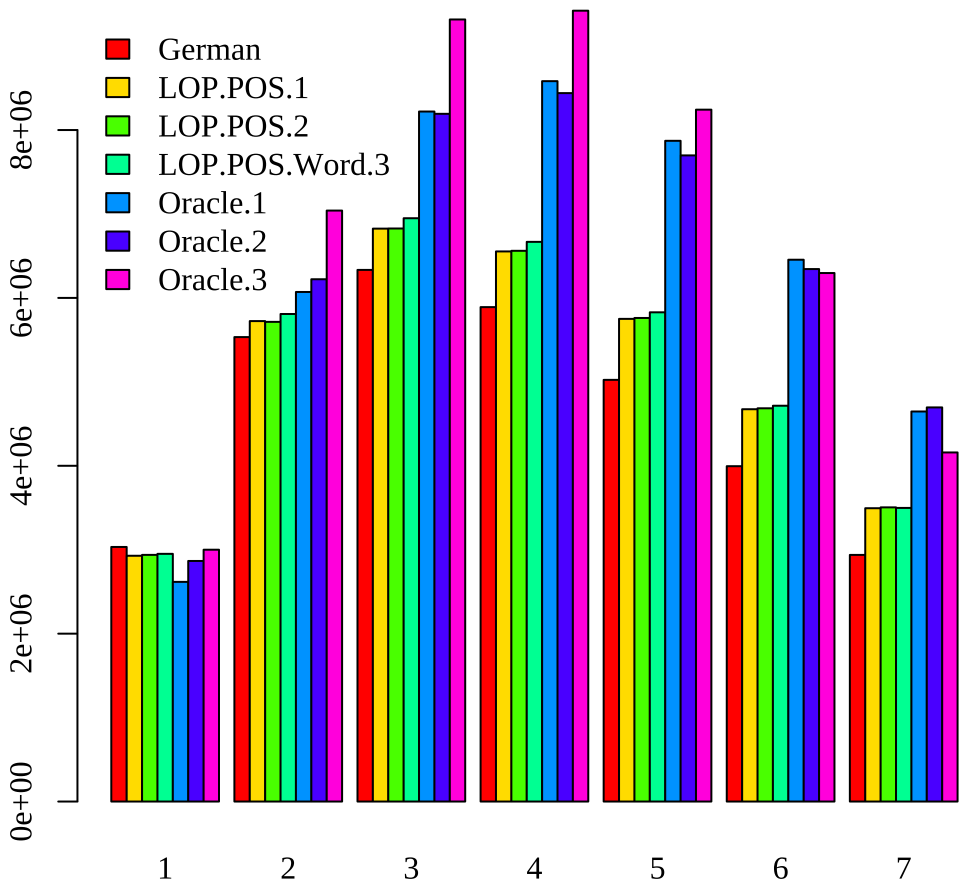

Figure 3.5: Count of phrases by length in phrase tables derived from different reorderings.

German	the commission has made several attempts to improve the coordination between the european union and the monetary union made to the issue of external representation within the international monetary fund to resolve.
German′	the commission has made several attempts to improve the coordination between the european union and the monetary union done to resolve the problem of the external representation of the international monetary fund.
Reference	the commission has made several attempts to improve the coordination of the european union and the monetary union in order to resolve the problem of the external representation of the international monetary fund.
German	mr president, i would like to begin my satisfaction on the report on the stability and convergence programmes, which mr katiforis has drawn up on the initiative of parliament.
German′	mr president, i would first like to express my satisfaction with the report on the stability and convergence programmes, mr katiforis has drawn up on the initiative of the european parliament.
Reference	mr president, i would first like to express my satisfaction with the report, initiated by parliament, on the stability and convergence programmes prepared by mr katiforis.
German	in accordance with rule 137 (1) (1) of the rules of procedure)
German′	(explanation of vote cut short pursuant to rule 137 (1) of the rules of procedure)
Reference	(explanation of vote cut short pursuant to rule 137 (1) of the rules of procedure)
German	therefore, we must welcome the support and proposals put forward by the committee on budgets for an increase in the budget for prince.
German′	that is why we support and welcome the proposals tabled by the committee on budgets for an increase in the budget for prince.
Reference	that is why we support and welcome wholeheartedly the proposals tabled by the committee on budgets for an increase in the prince budget.
German	mr president, i believe that this is not just about to refute the idea that, in the member states is the poor from brussels and that good a work of the government.
German′	i believe, mr president, that it is not only to refute the idea that in the member states is the poor from brussels and that is a good work of the government.
Reference	i believe, mr president, that it is not only a matter of refuting the idea that everything bad in the member states comes from brussels and everything good comes from the governments.

Table 3.4: The five best prepocessed sentences of LOP POS 2. See the text for a description of the selection criterion.

ART	NN	VAFIN	PIAT	NN	APPRART	NN	ART	NN	ART
Die	Kommission	hat	mehrere	Versuche	zur	Verbesserung	der	Koordinierung	der
THE	COMISSION	HAS	SEVERAL	ATTEMPTS	TO THE	IMPROVEMENT	OF THE	COORDINATION	OF THE

ADJA	NN	KON	ART	NN	VVPP	$,	KOUI	ART	NN	ART
Europäischen	Union	und	der	Währungsunion	unternommen	,	um	das	Problem	der
EUROPEAN	UNION	AND	OF THE	MONETARY UNION	UNDERTAKEN	,	ABOUT	THE	PROBLEM	OF THE

NN	APPRART	ADJA	NN	PTKZU	VVINF	$.
Außenvertretung	im	Internationalen	Währungsfonds	zu	lösen	.
EXTERNAL REPRESENTATION	IN THE	INTERNATIONAL	MONETARY FUND	TO	RESOLVE	.

NN	NN	$,	PPER	VMFIN	ADV	PPOSAT	NN	APPR	ART	NN	APPR	ART
Herr	Präsident	,	ich	möchte	zunächst	meine	Befriedigung	über	den	Bericht	zu	den
MR	PRESIDENT	,	I	WOULD LIKE	FIRST	MY	SATISFACTION	WITH	THE	REPORT	ON	THE

TRUNC	KON	NN	VVINF	$,	ART	NN	NE	APPR	NN
Stabilitäts-	und	Konvergenzprogrammen	ausdrücken	,	den	Herr	Katiforis	auf	Initiative
STABILITY	AND	CONVERGENCE PROGRAMS	EXPRESS	,	WHICH	MR	KATIFORIS	ON	INITIATIVE

ART	NN	VVPP	VAFIN	$.
des	Parlaments	ausgearbeitet	hat	.
OF THE	PARLIAMENT	PREPARED	HAS	.

$*LRB*	ADJA	NN	CARD	NN	CARD	VVFIN	ADJA	NN	APPRART
(	Gemäß	Artikel	137	Absatz	1	GO	gekürzte	Erklärung	zur
(	PURSUANT	RULE	137	SECTION	1	RULES OF PROCEDURE	CUT SHORT	EXPLANATION	OF THE

NN	$*LRB*
Abstimmung	)
VOTE	)

PROAV	VVFIN	KON	VVFIN	PPER	ART	APPRART	NN	ADJA
Deshalb	unterstützen	und	begrüßen	wir	die	vom	Haushaltsausschuß	eingebrachten
THUS	SUPPORT	AND	WELCOME	WE	THE	FROM THE	BUDGET COMMITTEE	TABLED

NN	APPR	ART	NN	ART	NN	APPR	CARD	$.
Vorschläge	für	eine	Aufstockung	des	Haushalts	für	Prince	.
PROPOSALS	FOR	AN	INCREASE	OF THE	BUDGET	FOR	PRINCE	.

PPER	VVFIN	$,	NN	NN	$,	KOUS	PPER	PTKNEG	ADV	PROAV	VVFIN	$,	ART	NN
Ich	glaube	,	Herr	Präsident	,	daß	es	nicht	nur	darum	geht	,	die	Vorstellung
I	BELIEVE	,	MR	PRESIDENT	,	THAT	IT	NOT	ONLY	A MATTER	IS	,	THE	IDEA

PTKZU	VVINF	$,	KOUS	APPR	ART	NN	ART	ADJA	APPR	NE	VVFIN
zu	widerlegen	,	daß	in	der	Mitgliedstaaten	das	Schlechte	aus	Brüssel	kommt
TO	REFUTE	,	THAT	IN	THE	MEMBER STATES	THE	BAD	FROM	BRUSSELS	COMES

KON	ART	NN	ART	NN	ART	NN	VAFIN	$.
und	das	Gute	ein	Werk	der	Regierung	ist	.
AND	THE	GOOD	AN	ACT	OF THE	GOVERNMENT	IS	.

Table 3.5: The original German of the five best preprocessed sentences, with automatically assigned parts of speech and English gloss.

German	die kommission hat mehrere versuche zur verbesserung der koordinierung der europäischen union und der währungsunion unternommen, um das problem der außenvertretung im internationalen währungsfonds zu lösen.
German′	die kommission hat mehrere versuche zur verbesserung der koordinierung der europäischen union und der währungsunion unternommen, um **zu lösen** das problem der außenvertretung im internationalen währungsfonds.
German	herr präsident, ich möchte zunächst meine befriedigung über den bericht zu den stabilitäts- und konvergenzprogrammen ausdrücken, den herr katiforis auf initiative des parlaments ausgearbeitet hat.
German′	herr präsident, ich möchte zunächst **ausdrücken** meine befriedigung über den bericht zu den stabilitäts- und konvergenzprogrammen, den herr katiforis **ausgearbeitet hat** auf initiative des parlaments.
German	(gemäß artikel 137 absatz 1 go gekürzte erklärung zur abstimmung)
German′	(**erklärung zur abstimmung** gemäß artikel 137 go gekürzte **absatz 1**)
German	deshalb unterstützen und begrüßen wir die vom haushaltsausschuß eingebrachten vorschläge für eine aufstockung des haushalts für prince.
German′	deshalb **wir** unterstützen und begrüßen die eingebrachten vorschläge **vom haushaltsausschuß** für eine aufstockung des haushalts für prince.
German	ich glaube, herr präsident, daß es nicht nur darum geht, die vorstellung zu widerlegen, daß in den mitgliedstaaten das schlechte aus brüssel kommt und das gute ein werk der regierung ist.
German′	ich glaube, herr präsident, daß es nicht nur darum geht, **zu widerlegen** die vorstellung, daß in den mitgliedstaaten das schlechte **kommt** aus brüssel und das gute **ist** ein werk der regierung.

Table 3.6: The German side of the five best preprocessed sentences. Bold indicates reordered words.

German	. (sv) we have today voted against the european parliament's report on the adoption of a new statute for members voted in favour, however, with 294 votes to 171 votes against and abstentions 59.
German′	- (sv) we have voted against the report by the european parliament voted for the adoption of a new statute for members, however, with the 294 votes to 171 votes against and abstentions 59, has been adopted.
Reference	. (sv) we have today voted against the european parliament's report on a new statute for members, which was however approved by 294 votes, with 171 against and 59 abstentions.
German	i support the commission's framework programme to combat trafficking in human beings, whose objective in the construction of an effective judicial cooperation in the european union.
German′	i support the framework programme the commission on combating trafficking in human beings, whose aim is to build an effective judicial cooperation in the european union.
Reference	i support the commission's framework programme to combat trafficking in human beings; the aim is to develop effective judicial cooperation across the european union.
German	let me come to a final point, the improved coordination between the development cooperation policy and the external aspects of the common fisheries policy is concerned.
German′	let me turn to a last point, the improved coordination between the policy of cooperation and the external aspects of the common fisheries policy.
Reference	let me come to a final point, which concerns improved coordination between development cooperation policy and the external aspects of the common fisheries policy.
German	i believe that the vast majority of the members of this parliament also wants the commission to appeal against this shocking decision.
German′	i believe that the great majority of the members of this parliament that the commission appointment procedure against this shocking decision.
Reference	i believe that the vast majority of members of this parliament would wish the commission to appeal against this shocking decision.
German	what eurobonds is concerned, we agree with the rapporteur's idea to exempt eurobonds with a high mindeststückelung, but we consider the proposed value of eur 50 000 is too low.
German′	as far as the eurobonds is concerned, we agree with the rapporteur's idea of eurobonds, with a high mindeststückelung exclude, but we feel that the proposed value of eur 50 000 for too low.
Reference	as regards eurobonds we agree with the rapporteur's idea to exempt eurobonds with a high minimum denomination but we consider that the level proposed - eur 50,000 - is too low.

Table 3.7: The five worst preprocessed sentences.

ART	$*LRB*	NN	$*LRB*	PPER	VAFIN	ADV	APPR	ART	NN	ART	ADJA	NN
.	(	SV	)	Wir	haben	heute	gegen	den	Bericht	des	Europäischen	Parlaments
.	(	SV	)	WE	HAVE	TODAY	AGAINST	THE	REPORT	OF THE	EUROPEAN	PARLIAMENT

APPRART	NN	ART	ADJA	NN	VVPP	$,	PRELS	ADV	APPR
zur	Annahme	eines	neuen	Abgeordnetenstatuts	gestimmt	,	der	allerdings	mit
ON THE	ADOPTION	OF A	NEW	MEMBER STATUTE	VOTED	,	WHICH	HOWEVER	WITH

CARD	NN	APPR	CARD	XY	KON	CARD	XY	VVPP	VAFIN	$.
294	Stimmen	bei	171	Gegenstimmen	und	59	Enthaltungen	angenommen	wurde	.
294	VOTES	WITH	171	AGAINST VOTES	AND	59	ABSTENTIONS	APPROVED	WAS	.

PPER	VVFIN	ART	NN	ART	NN	APPRART	NN
Ich	unterstütze	das	Rahmenprogramm	der	Kommission	zur	Bekämpfung
I	SUPPORT	THE	FRAMEWORK PROGRAM	OF THE	COMMISION	ON THE	COMBAT

ART	NN	$,	PRELAT	NN	APPRART	NN	ART	ADJA	ADJA
des	Menschenhandels	,	dessen	Ziel	im	Aufbau	einer	wirksamen	justiziellen
OF THE	HUMAN TRAFFICKING	,	WHOSE	AIM	IN THE	DEVELOPMENT	A	EFFECTIVE	JUDICIAL

NN	APPR	ART	ADJA	NN	VVFIN	$.
Zusammenarbeit	in	der	Europäischen	Union	besteht	.
COOPERATION	IN	THE	EUROPEAN	UNION	EXISTS	.

VVIMP	PPER	PRF	APPR	ART	ADJA	NN	VVINF	$,	PRELS	ART	ADJA
Lassen	Sie	mich	zu	einem	letzten	Punkt	kommen	,	der	die	verbesserte
LET	YOU	ME	TO	A	FINAL	POINT	COME	,	WHICH	THE	IMPROVED

NN	APPR	ART	NN	ART	NN	KON	ART	ADJA
Koordinierung	zwischen	der	Politik	der	Entwicklungszusammenarbeit	und	den	externen
COORDINATION	BETWEEN	THE	POLICY	OF THE	DEVELOPMENT COOPERATION	AND	THE	EXTERNAL

NN	ART	ADJA	NN	VVFIN	$.
Aspekten	der	gemeinsamen	Fischereipolitik	betrifft	.
ASPECTS	OF THE	COMMON	FISHERIES POLICY	CONCERNS	.

PPER	VVFIN	$,	ART	ADJA	NN	ART	NN	PDAT	NN	VVFIN	$,	KOUS
Ich	glaube	,	die	große	Mehrheit	der	Abgeordneten	dieses	Parlaments	wünscht	,	daß
I	BELIEVE	,	THE	VAST	MAJORITY	OF THE	MEMBERS	OF THIS	PARLIAMENT	WISH	,	THAT

ART	NN	NN	APPR	PDAT	ADJA	NN	VVFIN	$.
die	Kommission	Berufung	gegen	diese	schockierende	Entscheidung	einlegt	.
THE	COMMISSION	APPEAL$_1$	AGAINST	THIS	SHOCKING	DECISION	APPEAL$_2$	.

PWS	ART	NN	VVFIN	$,	ADV	VVFIN	PPER	ART	NN	ART	NN
Was	die	Eurobonds	betrifft	,	so	stimmen	wir	dem	Gedanken	des	Berichterstatters
WHAT	THE	EUROBONDS	REGARDS	,		AGREE$_1$	WE	THE	IDEA	OF THE	RAPPORTEUR

PTKVZ	$,	ADJD	APPR	ART	ADJA	NN	VVIZU	$,	KON	PPER	VVFIN
zu	,	Eurobonds	mit	einer	hohen	Mindeststückelung	auszunehmen	,	aber	wir	halten
AGREE$_2$	,	EUROBONDS	WITH	A	HIGH	MINIMUM DENOMINATION	EXEMPT	,	BUT	WE	HOLD

ART	ADJA	NN	CARD	CARD	NN	APPR	PTKA	ADJD	$.
den	vorgeschlagenen	Wert	50	000	Euro	für	zu	gering	.
THE	PROPOSED	VALUE	50	000	EURO	FOR	TOO	LOW	.

Table 3.8: The original German of the five worst preprocessed sentences, with automatically assigned parts of speech and English gloss.

German	. (sv) wir haben heute gegen den bericht des europäischen parlaments zur annahme eines neuen abgeordnetenstatuts gestimmt , der allerdings mit 294 stimmen bei 171 gegenstimmen und 59 enthaltungen angenommen wurde.
German′	**wir haben**. (sv) heute gegen den bericht des europäischen parlaments **gestimmt** zur annahme eines neuen abgeordnetenstatuts, der allerdings mit 294 stimmen bei 171 gegenstimmen und 59 enthaltungen angenommen wurde.
German	ich unterstütze das rahmenprogramm der kommission zur bekämpfung des menschenhandels, dessen ziel im aufbau einer wirksamen justiziellen zusammenarbeit in der europäischen union besteht.
German′	ich unterstütze das rahmenprogramm der kommission zur bekämpfung des menschenhandels, dessen ziel im aufbau **besteht** einer wirksamen justiziellen zusammenarbeit in der europäischen union.
German	lassen sie mich zu einem letzten punkt kommen, der die verbesserte koordinierung zwischen der politik der entwicklungszusammenarbeit und den externen aspekten der gemeinsamen fischereipolitik betrifft.
German′	lassen sie mich **kommen** zu einem letzten punkt, der die verbesserte koordinierung zwischen der politik **betrifft** der entwicklungszusammenarbeit und den externen aspekten der gemeinsamen fischereipolitik.
German	ich glaube, die große mehrheit der abgeordneten dieses parlaments wünscht, daß die kommission berufung gegen diese schockierende entscheidung einlegt.
German′	ich glaube, die große mehrheit **wünscht** der abgeordneten dieses parlaments, daß die kommission berufung **einlegt** gegen diese schockierende entscheidung.
German	was die eurobonds betrifft, so stimmen wir dem gedanken des berichterstatters zu, eurobonds mit einer hohen mindeststückelung auszunehmen, aber wir halten den vorgeschlagenen wert 50 000 euro für zu gering.
German′	was die eurobonds betrifft, so **wir** stimmen **zu** dem gedanken des berichterstatters, eurobonds mit einer hohen mindeststückelung auszunehmen, aber wir halten den vorgeschlagenen wert 50 000 euro für zu gering.

Table 3.9: The German side of the five worst preprocessed sentences.

Some consistent patterns appear in the German$'$:

- *wir* moves to precede neighboring verbs, and
- verbs such as *lösen*, *ausdrücken*, etc., move from clause final position into an appropriate SVO ordering, as does the separable prefix *zu* in the last sentence of Table 3.9, although it would be better prefixed to *stimmen*.

In fact, some of the reorderings in Table 3.9 are appropriate, despite the fact that they lead to lower BLEU.

This output suggests that, to a large extent, the models Chapter 4 is learning are doing the right thing, even if that doesn't always reflect in the automatic translation measures. There are many possible reasons for this, related to what happens to the reordered German once it is handed to Moses for translation.

- Short-distance reordering is unlikely to help, because the phrase table can handle it directly, particularly for common phrases where alignments are likely to be good.
- Reordering may lead to different lexical choice, which may sometimes incidentally result in lower BLEU scores.
- Some reordering may move the right words to the wrong places, resulting in better word order that is nonetheless incorrect.

The next section goes beyond preprocessing, proposing some methods for integrating the linear ordering problem into the decoding process.

3.9 Exact Algorithms

Figure 3.6 gives a grammar in terms of states in the automaton A to adapt VLSN search using permutation tree parsing to the ABC model of Section 3.6. $B(\pi)+C(\pi)$ can be handled using the grammars of Sections 2.8 and 3.7, but $A(\pi)$ complicates things. In addition to computing the score of the permutation, the grammar must ensure that the score comes from a valid path through the automaton, beginning at the distinguished initial state and ending at some final state.

Including $A(\pi)$ requires only two changes to the grammar scores:

- Rules of the form $S \rightarrow S_{0,n,i\rightsquigarrow f}$ score $F(f)$.
- Rules of the form $S_{i-1,i,q\rightsquigarrow r} \rightarrow \pi_i$ score $\delta(q,r,\pi_i)$.

Chapter 2 assumes that weights are added during parsing. This requires that the semiring of the automaton has an extend operator $\otimes \equiv +$. Further, the algorithms perform score maximization, so it is appropriate for the semiring to be $\langle \mathbb{R} \cup -\infty, \max, +, -\infty, 0\rangle$. This is the negation of the tropical semiring.

$$
\begin{array}{rcl}
S & \rightarrow & S_{0,n,i\rightsquigarrow f},\ \forall f \in Q,\ F(q) \neq \bar{0} \\
\\
S_{i,k,q\rightsquigarrow s} & \rightarrow & S_{i,j,q\rightsquigarrow r}\ S_{j,k,r\rightsquigarrow s} \\
S_{i,k,q\rightsquigarrow s} & \rightarrow & S_{j,k,q\rightsquigarrow r}\ S_{i,j,r\rightsquigarrow s},\ \forall (i,j,k) \in \binom{n+1}{3},\ q,r,s \in Q \\
\\
S_{i-1,i,q\rightsquigarrow r} & \rightarrow & \pi_i,\ \forall i \in (0,n),\ \delta(q,r,\pi_i) \neq \bar{0}
\end{array}
$$

Figure 3.6: A grammar for generating permutations as paths through an automaton. S is the start symbol of the grammar. The WFSA is as in Definition 3.4. The grammar as given does not account for arcs labeled with ϵ. Either ϵ-closure in the pre-terminal rules or ϵ-removal for the whole acceptor solves this problem.

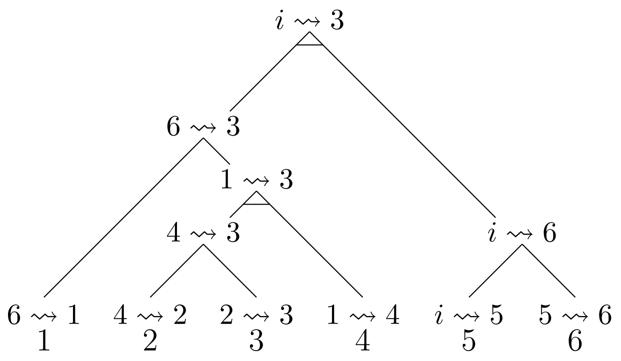

Figure 3.7: An example permutation tree augmented with automaton paths. The automaton is a simple bigram model, so that any arc accepting symbol a leads to a state numbered a. State i is a distinguished initial state with only outgoing arcs. Note that, at reverse nodes, the right child's path ends at the same state where the left child's path begins. The start nonterminal S can rewrite as $i \rightsquigarrow 3$ (technically $S_{0,6,i\rightsquigarrow 3}$) because i is the initial state and 3 is a final state.

The number of rules in this grammar as written is $\Theta(n^3 |Q|^3)$. If A has the structure of a bigram language model, then $|Q| = \Theta(n)$, and there are $\Theta(n^6)$ rules, while if it has the structure of a trigram language model, $|Q| = \Theta(n^2)$ and there are $\Theta(n^9)$ rules.

Huang, Zhang, and Gildea (2005) showed how to speed up Inversion-Transduction Grammar decoding with an integrated language model by adapting the so-called "hook trick" of Eisner and Satta (1999). Eisner and Satta's hook trick for bilexical parsing converts a single maximization with five variables ranging over positions in the sentence—$O(n^5)$—into two maximizations with four variables each—$O(n^4)$. Both tricks are examples of *folding* transformations (Eisner and Blatz, 2007).

Huang et al.'s adapted trick converts ITG decoding with a bigram language model from $O(n^7)$ to $O(n^6)$ and with a trigram model from $O(n^{11})$ to $O(n^9)$. Notice that the grammar of Figure 3.6 already has this faster runtime. Careful inspection reveals that this is because the grammar rules include the source state of the path through A accepting their subpermutations. That source state carries the identity of the single word adjacent to the subpermutation if A is a bigram language model, and the pair of adjacent words if A is a trigram language model.

Several other innovations have improved the efficiency of decoding algorithms integrating language models with syntax-based machine translation systems, including ITG. This chapter touches on some of those in later sections.

Using this grammar, it is possible to perform exact search in the BlockInsert* neighborhood of any starting permutation. However, asymptotic runtimes like $\Theta(n^9)$ are not very attractive, even for relatively small machine translation problems where n averages about 25. Further, this analysis ignores the phrase table, which potentially includes multiple translation candidates for each German word. Moses' standard settings allow up to twenty translations for each source phrase.

Section 3.9.1 provides a more careful asymptotic analysis. Section 3.9.2 proposes an A* procedure to mitigate somewhat the effect of the $A(\pi)$ model. Section 4.11.4, in the next chapter, discusses a specialization of the A model that can be computed efficiently in one neighborhood. Section 3.9.3 abandons the very large-scale neighborhoods and searches using a modification of the Block $\mathcal{LS}_f$ procedure.

These methods are called *exact* because they maximize $A(\pi) + B(\pi) + C(\pi)$ over some neighborhood of the current permutation. They are still subject to search error, though, because the scope of the neighborhood is limited. Section 3.10 proposes additional decoding algorithms with fewer search error guarantees.

3.9.1 Asymptotic Analysis

This section performs a more careful asymptotic analysis of the runtime of parsing with the grammar of Figure 3.6, including the constants on the highest order terms, for one particular A model—a bigram language model. For sentence lengths encountered in practice, coarse asymptotic analysis doesn't tell the whole story. The constants involved are sometimes larger than the average n, for example. On the other hand, the bigram language model is not exactly the model of interest. However, it is easy to analyze more precisely, and the results may have implications for more interesting models as well.

A simple bigram model for a sentence of length n, ignoring the question of translation, has n states, each with n outgoing arcs, one to each state. Only the words in the sentence are possible in the permutation, so other states and arcs in the language model are eliminated.

For a single word span $(i-1, i)$, there are n arcs that could have generated it, each of the form

$$q \xrightarrow{\pi_i/w} q_{\pi_i}.$$

For a two-word span $(i-1, i+1)$, the end state of the first arc must match the start state of the second, so, when combined in-order, the second arc must be

$$q_{\pi_i} \xrightarrow{\pi_{i+1}/w'} q_{\pi_{i+1}}.$$

There are therefore $2n$ path types, starting at any state and ending at either q_{π_i} or $q_{\pi_{i+1}}$, depending on the order.

In general, then, for a span of width k, there are kn path types, since a path can start at any state, but must end with one of the states corresponding to the words in the span.

When combining two spans $(i, i+k)$ and $(i+k, i+k+\ell)$ of widths k and ℓ, there are kn and ℓn paths to consider. For each ending state of the first span, there are ℓ paths in the second span, and for each ending state of the second span, there are k paths in the first span. Therefore, the number of paths the span $(i, i+k+\ell)$ must consider when combining $(i, i+k)$ and $(i+k, i+k+\ell)$ is $(kn)\ell + (\ell n)k = 2k\ell n$.

So, to build $(i, i+m)$, considering all $k+\ell = m$, there are

$$\begin{aligned}
\sum_{k=1}^{m-1} 2k(m-k)n &= 2n\left(\sum_{k=1}^{m-1} km - k^2\right) \\
&= 2n\left(m\,\frac{1}{2}\,\Theta(m^2) - \frac{1}{3}\,\Theta(m^3)\right) \\
&= 2n\left(\frac{1}{6}\,\Theta(m^3)\right) \\
&= \frac{1}{3}\,\Theta(n^4)
\end{aligned}$$

operations per built span.

For Insert$^*_n$, which only combines spans $(i, i+1), (i+1, i+m)$ or $(i, i+m-1), (i+m-1, i+m)$, there are

$$2(1)(m-1)n + 2(m-1)(1)n = 4n(m-1) = 4\ \Theta(n^2)$$

operations per span.

In either case, there are a total of $\Theta(n^2)$ spans to build, but they are not all the same size. A tighter analysis is possible, taking the widths of the spans into account. There are $n-1$ width-2, $n-2$ width-3, ..., 1 width-n spans.

$$\begin{aligned}
&\sum_{m=2}^{n}(n-m+1)\left[\frac{n}{3}\,\Theta(m^3)\right] \\
&\quad= \sum_{m=2}^{n}\frac{n^2}{3}\,\Theta(m^3)-\frac{n}{3}\,\Theta(m^4)+\frac{n}{3}\,\Theta(m^3) \\
&\quad= \frac{n^2+n}{3}\sum_{m=2}^{n}\,\Theta(m^3)-\frac{n}{3}\sum_{m=2}^{n}\Theta(m^4) \\
&\quad= \frac{n^2+n}{3}\,\frac{1}{4}\,\Theta(n^4)-\frac{n}{3}\,\frac{1}{5}\,\Theta(n^5) \\
&\quad= \frac{1}{60}\,\Theta(n^6).
\end{aligned}$$

For Insert$^*_n$,

$$\begin{aligned}
&\sum_{m=2}^{n}(n-m+1)4n\Theta(m) \\
&\quad= 4n^2\sum_{m=2}^{n}\Theta(m)-4n\sum_{m=2}^{n}\Theta(m^2) \\
&\quad= 4n^2\,\frac{1}{2}\,\Theta(n^2)-4n\,\frac{1}{3}\,\Theta(n^3) \\
&\quad= \frac{2}{3}\,\Theta(n^4).
\end{aligned}$$

The Insert$^{\leq 2*}$ neighborhood has the same $4n(m-1)$ operations and also $2(2)(m-2)n+2(m-2)(2)n=8n(m-2)$, for a total of $12\ \Theta(n^2)$ instead of $4\ \Theta(n^2)$. Insert$^{\leq 3*}$ adds $2(2(3)(m-3)n)=12\ \Theta(n^2)$ for $24\ \Theta(n^2)$ total.

Since the language model in practice will be composed with a transducer that maps indices to words, there is no need to consider duplications—every *index* is unique. For a given span $(i,i+k)$, therefore, only $k(n-k)$ paths are compatible with the indices in $(0,i)\cup(i+k,n)$. Could efficiently eliminating those incompatible paths from consideration improve efficiency?

Simply change the n in $2k\ell n$ to $n-m$, where $m=k+\ell$. Therefore, there are

$4(m-1)(n-m)$ operations per span $= 4(n(m-1) - m(m-1))$. So,

$$\begin{aligned}
& 4\sum_{m=2}^{n}(n-m+1)(n-m)(m-1) \\
&= 4\sum_{m-2}^{n}\Theta(m)(n-\Theta(m))^2 \\
&= 4\sum_{m=2}^{n} n^2\Theta(m) - 2n\Theta(m^2) + \Theta(m^3) \\
&= 4n^2\,\frac{1}{2}\,\Theta(n^2) - 8n\,\frac{1}{3}\,\Theta(n^3) + 4\,\frac{1}{4}\,\Theta(n^4) \\
&= \frac{1}{3}\,\Theta(n^4).
\end{aligned}$$

This cuts the expected runtime in half, but does not change the order of magnitude. For BlockInsert$^*_n$:

$$\begin{aligned}
& \sum_{k=1}^{m-1} 2k(m-k)(n-m) \\
&= \frac{1}{3}(n-m)\Theta(m^3)
\end{aligned}$$

per built span.

$$\begin{aligned}
& \frac{1}{3}\sum_{m=2}^{n}\Theta(m^3)(n-\Theta(m))^2 \\
&= \frac{n^2}{3}\,\frac{1}{4}\,\Theta(n^4) - \frac{2n}{3}\,\frac{1}{5}\,\Theta(n^5) + \frac{1}{3}\,\frac{1}{6}\,\Theta(n^6) \\
&= \frac{1}{180}\,\Theta(n^6),
\end{aligned}$$

one-third of before. This constant is small enough to significantly affect the runtime for sentences of length less than forty words. Unfortunately, the factor of n^6 remains problematic—$n^6 > 2^n$ for $n < 30$.

3.9.2 A* Search

The source of the runtime problems for decoding is the size of the state set of the automaton A, $|Q|$. Reducing the number of states leads to faster decoding. However, reducing the number of states also defeats the purpose of higher-order language models.

A compromise is to perform a first decode using an approximation to A—call it A'—with fewer states, and then to use the results of that decode to prune efforts with the full A model. If $A'(\pi) > A(\pi)$, which is easy to arrange, then A' is an *admissible heuristic* for A, and A* search is applicable.

The simplest approximation to A, and the one with the best runtime, is a one-state automaton A'. Ignoring the issue of ϵ transitions, the weight of the transition $0 \xrightarrow{a/w^*} 0$ in A' is the one from the best arc accepting a in A:

$$w^* = \max_{q,r \in Q} \delta(q, r, a).$$

The resulting A* search procedure is an adaptation of the method of Klein and Manning (2003). Recall from Section 2.8.6 that $\beta(N)$ indicates the best derivation starting from non-terminal N. Call this the *inside* score of N. Now, let $\alpha(N)$ be the best derivation starting from the start symbol S and stopping at N. Call this the *outside* score of N.[14] $\alpha(S) = 0$, and $\alpha(N)$ is computed as follows, assuming the grammar is binary:

$$\alpha(N) = \max \left(\begin{array}{l} \max\limits_{N_1 \to N\ N_2} \alpha(N_1) + \gamma(N_1 \to N\ N_2) + \beta(N_2), \\ \max\limits_{N_1 \to N_2\ N} \alpha(N_1) + \gamma(N_1 \to N_2\ N) + \beta(N_2) \end{array} \right). \tag{3.20}$$

Let α' and β' refer to the same quantities computed using the simplified grammar that results from replacing A with A'. Then the A* search procedure is

1. Compute $\beta'(N)$, $\forall N \in G$ bottom up.
2. Compute $\alpha'(N)$, $\forall N \in G$ top down, using $\beta'(N)$.
3. Compute $\beta(N)$ in best-first order, using $\beta(N) + \alpha'(N)$ as the value of N. Stop as soon as $\beta(S)$ pops.

If $A'(\pi) > A(\pi)$ then $\alpha'(N) > \alpha(N)$ is *admissible*, and the result is A* search. An alternative is to use an approximation A' that is not a bound. The resulting best-first procedure is not guaranteed to find the best permutation in the neighborhood. However, this may still be acceptable, especially as part of an iterated local search.

Unfortunately, the one-state bound on A failed to help runtimes significantly in experiments on machine translation decoding. Probably, a better tradeoff between the number of states and the tightness of the approximation is warranted.

Felzenszwalb and McAllester (2007) introduced an architecture into which this formulation of A* permutation parsing fits. Successive approximations of the A model, ultimately arriving at the one-state A' model described above, would be an instance of their HA*LD algorithm.

[14] As Klein and Manning observe, these scores are reminiscent of the Inside-Outside algorithm (Baker, 1979), but they differ—the scores of the same name in the Inside-Outside algorithm *sum* over derivations, while here α and β are *maximum* scores.

One way to reduce the number of states in a finite-state automaton is to merge existing states. For example, Stolcke and Omohundro (1993) used state-merging to generalize strings for unsupervised learning of HMMs. This has the effect of changing the language accepted by the FSA, in general, but that is not a problem in this setting. The choice of which states to merge is not trivial, though. The closely related minimum-cost set cover problem is NP-hard, so many objective functions will result in NP-hard optimization problems as well. Stolcke and Omohundro used greedy search to choose which pair of states to merge next. In the permutation setting, it is not clear how to evaluate the quality of the approximation, since the language the automaton has to score is all possible permutations.

HA*LD is an attractive possibility for potentially rendering decoding under these models much faster, but it remains future work.

3.9.3 Block Insertion

The best performing search method for $B(\pi)$ alone, as described in Chapter 2, was not the VLSN search but the Block $\mathcal{LS}_f$ method of Section 2.5.4. This section resurrects that search procedure and augments it with the $A(\pi)$ scores.

Automaton scores complicate Block $\mathcal{LS}_f$ significantly. This is because moving even a small subsequence of the permutation results, in general, in a completely different path through the automaton.

In order to compute the score, under $A(\pi)$ of the permutation that results from transposing the blocks (i, j) and (j, k), the algorithm must know all the paths compatible with the spans $(0, i)$, (i, j), (j, k), and (k, n). Since i, j, and k range over the entire sequence, this ultimately implies knowing all the paths compatible with all $\binom{n+1}{2}$ subsequences of the permutation.

The A score for the transposition of (i, j) and (j, k) is the best path matching $(0, i)$ (j, k) (i, j) (k, n). Computing this requires matching the end states of the paths matching $(0, i)$ with the start states of the paths matching (j, k), the end states of the paths matching (j, k) with the start states of the paths matching (i, j), and so on. $(0, i)$ might be empty, in which case the path matching (j, k) must also start at the distinguished initial state. Likewise (k, n) might be empty, in which case the path matching (i, j) must end at a final state.

Block $\mathcal{LS}_f$ with A scores is reminiscent of the decoders described in Germann, Jahr, Knight, Marcu, and Yamada (2001); Germann (2003), the latter of which used five update operations:

CHANGE the translation of a source word,

INSERT a zero-fertility target word not aligned to any source word,

ERASE a zero-fertility target word,

JOIN two target words, erasing one and transfering its source language alignments to the other, or

SWAP two non-overlapping target regions.

The transducer (projected to an acceptor) obviates the first four operations for Block $\mathcal{LS}_f$. $A(\pi)$ computes the score of the best possible monotone translation, under the model, of π. The JOIN operation is an artifact of the noisy-channel model, where the target language words "generate" the source language. Germann's search cannot simply delete a target language word, because that target word might be responsible for generating one or more source words. Therefore those source words must be assigned to some other target word in order to perform the deletion.

Germann's SWAP operation is slightly more general than Block $\mathcal{LS}_f$'s transpositions, because SWAP's argument regions need not be adjacent. There are $\binom{n+1}{4}$ possible SWAP operations. Germann ultimately limits the set of swaps for efficiency.

Computing the cost of such a move is barely more complex than a block transposition under the A model. The B cost of a SWAP of (i, j) and (k, ℓ), with (j, k) remaining between them, is $\overleftarrow{\gamma}_{i,j,\ell} + \overleftarrow{\gamma}_{j,k,\ell}$ or, equivalently, $\overleftarrow{\gamma}_{i,k,\ell} + \overleftarrow{\gamma}_{i,j,k}$.[15] However, repeating the argument from Section 2.5.3, where it applied to interchanges versus insertions, SWAP provides no advantage over block transpositions in terms of local maxima. If there exists an improving SWAP, then there necessarily exists an improving block transposition as well: if $\overleftarrow{\gamma}_{i,j,\ell} + \overleftarrow{\gamma}_{j,k,\ell} > 0$ then at least one of the terms is positive.

Decoding with Block $\mathcal{LS}_f$ in practice unfortunately suffers from the same efficiency issues as all of the other procedures of this section. These are exacerbated by the fact that it has to compete with the 5-gram language model that Moses uses. The next section describes possible approximate decoding procedures.

3.10 Approximate Algorithms

This section introduces possible alternatives to the decoding procedures of Section 3.9. Each of the methods described here involves some inherent approximation that makes it qualitatively different from the methods there. This thesis reports no experiments using the algorithms described in this section.

3.10.1 Beam Search

The B and C models are compatible with standard beam-search decoding methods for phrase-based statistical MT systems. In this scheme, the phrase-based model

[15]There is a schedule for computing these that preserves the relative efficiency of Block $\mathcal{LS}_f$. Once a wider span's transposition costs have been computed, it can be exchanged with narrower spans anywhere or with as-wide spans to its left.

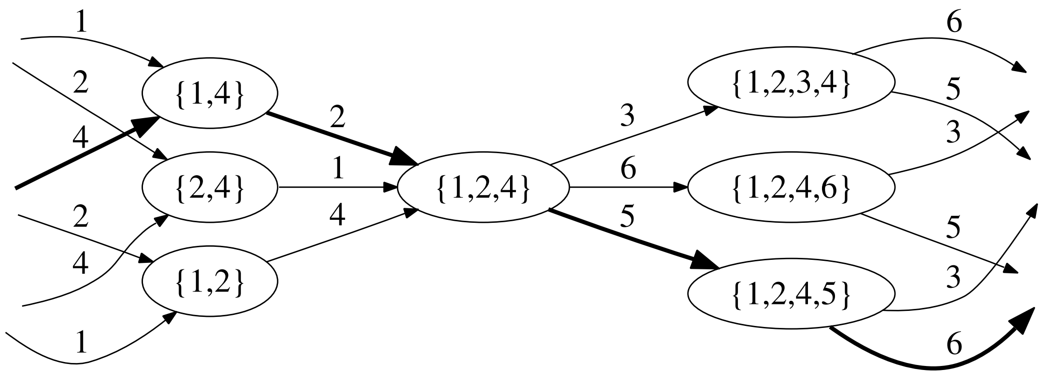

Figure 3.8: A small portion of a permutation automaton for a six-word sentence. Bold arcs indicate part of the path 1 4 2 5 6 3.

replaces the A model of Section 3.6. Beam search decoders, such as Moses (Koehn et al., 2007), are approximate because they prune the state space of their implicit automaton (see Figure 3.8) using inadmissible heuristics.

Continuing in terms of automata, the "states" of the beam search decoder are indexed by the subsets of the source words that paths ending at them cover,[16] and the "arcs" cover some new words by selecting phrases for translation. Figure 3.8 simplifies this by assuming word-based translation.

In any case, the partial score of each path leading up to a given state $q \in Q$ can take into account the B score of both

- the partial permutation, implied by the path, of the words covered by q and
- putting all the words covered by q *before* all the uncovered words.

The contribution of the B model to the "arc" weight in order to achieve this is just the internal B score of the possibly multiple words in the chosen phrase—zero if it is one word—plus the score for putting all the words in the chosen phrase before all the remaining uncovered words.

A reasonable contribution to the inadmissible heuristic for the B model is the B score of the best permutation in the BlockInsert* neighborhood of some default ordering of the uncovered words.

Incorporating the C model in beam search is only slightly more complicated. It is also not subject to the same cyclicity constraint as the dynamic programming search of Section 3.7—it works with an unconstrained C model. Any word on the current "arc" comes after the words already covered, and before the words not yet covered.

[16]There are 2^n such states. These are then implicitly composed with the $\Theta(n^{N-1})$ states of the language model, where N is the maximum n-gram size, assuming a constant number of possible translations for each word.

Additionally, if the "arc" adds multiple words, then phrase-internal C scores may be incurred, as well as scores involving two words on the "arc" and one either already covered, or yet to be.

3.10.2 Forest Rescoring

This section adapts the forest rescoring method of Huang and Chiang (2007), which uses n-gram language models, to the general automaton case.

As previous sections have shown, integrating the finite-state automaton scores into the neighborhood search is the most expensive part. Searching the BlockInsert$^*_n$ neighborhood with linear ordering problem scores is $\Theta(n^3)$, but integrating a bigram or trigram language model, even without the translation model involved, increases the complexity to $\Theta(n^6)$ and $\Theta(n^9)$, respectively.

There is an alternative, however, which is to build a parse forest using only the linear ordering problem scores, in cubic time, and then rescore the permutations using the additional finite-state model. This type of approach was introduced in Huang and Chiang (2007). Instead of allowing the automaton to essentially explode the grammar constant of parsing, this approach would limit computation of scores from the finite-state automaton to K-best lists at each node in the hypergraph forming the parse forest.

The tricky part is that A is a general finite-state automaton. Here, the path assigned to each permutation must traverse from the distinguished initial state to a final state, whereas in the case of an n-gram language model, given sufficient local context, the states can be inferred deterministically. Specifically, each ordering of a subsequence of the input may correspond to multiple possible subpaths in the automaton. The modified algorithm must keep track of all the possible paths, and can only use their weights as estimates of the true cost of the subsequence.

A potentially problematic alternative is to put paths into the K-best list rather than permutations. That is, in the previous formulation, the K-best list for a given span holds just K possible permutations of the items in the span, while each permutation matches possibly many automaton paths. In the new formulation, those many matching paths compete for slots in the K-best list. The potential problem arises if paths capable of connecting fall below those that cannot connect, ultimately leading to no successful paths matching the entire permutation. This is also a potential problem for the earlier K-best lists, because it is possible that the automaton only accepts particular permutations, but this formulation exacerbates that potential.

An obvious solution to this problem is iterative deepening. Start with a small manageable K and repeat the process with increasingly large Ks if smaller Ks fail to produce any paths.

3.11 Summary

The first contribution of this chapter is the representation of IBM Model 4 using a cascade of weighted finite-state transducers. It is not clear that this representation is of any use for translation, but it may well serve as a tutorial example of encoding a complex model with WFSTs. This is certainly a useful practice in general.

The second, and primary, contribution of the chapter is the ABC model of translation. This model cleanly divides translation into two components, namely reordering and monotonic word or phrase translation. At the same time, it pushes both components together to form a single model of source-language ordering. This is also a novel combinatorial optimization problem.

The C scores, a component of the translation model, constitute a third novel contribution. As yet, they have not been applied to any problem, but this chapter clearly lays out algorithms for working with them, including a dynamic program for block transposition scores, assuming a cyclicity constraint. The proposed approaches to decoding with the ABC model provide numerous opportunities for future research.

Preprocessing German for translation to English using search for the Linear Ordering Problem constitutes the dissertation's primary empirical result. This demonstrates both that fully automatic methods based on machine learning and combinatorial optimization can rival hand-written reordering rules, and that the Linear Ordering Problem is a legitimate approach to reordering for translation.

Chapter 4

Learning LOP Matrices for MT

Recall (2.3), which defined each matrix entry for the linear ordering problem as a dot product of a weight vector $\boldsymbol{\theta}$ with a feature function $\boldsymbol{\phi}$ that mapped a word sequence with a pair of distinguished positions to a vector of feature counts:

$$B[i, j] = \boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{w}, i, j).$$

This chapter explores both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. Section 4.4 describes possibile feature functions, including those used in experiments, and the rest of the chapter applies machine learning to the problem of finding a good setting of the weight vector.

This is admittedly controversial. It is natural to ask why linear ordering problems should have anything to do with machine translation whatsoever. The first answer is that there is some correspondence between the LOP and approaches that have proven successful for dependency parsing. This is the topic of Section 4.4.1. The second answer is that the model is in some ways of lesser importance than the features. Sections 4.4.2, 4.4.3, and 4.4.4 will inspect many properties of the sentence, including the words themselves, their parts of speech, and dependency parses, in order to build a sentence-specific reordering model. Hypothetically, that model could be a total order, specifying exactly the right permutation, were the features sufficiently sophisticated.

Before addressing these questions, Section 4.1 reviews the topic of machine learning from a high level, Section 4.2 describes one instance of prior work on learning an ordering function, and Section 4.3 lists resources used in the rest of the chapter.

4.1 Machine Learning

The aim of machine learning, given a model such as the linear model above, is to propose an optimization criterion for the weights $\boldsymbol{\theta}$ that will lead to good performance on unseen data, and to find optimal weights under that criterion.

An important consideration is the tradeoff between observable good performance on the training data and good performance on unseen data. Optimization criteria are subject to potential *overfitting*—learning details of the training data that do not generalize to unseen instances. Introducing some form of parameter regularization into the optimization criterion addresses this tradeoff explicitly.

Sometimes, as in the case of log-linear models with no hidden variables, the optimization criterion is convex, in which case simple optimization methods are guaranteed to find the desired model. For other procedures, e.g., famously, the Expectation Maximization (EM) algorithm, local optima are the norm—as with search using the neighborhoods of Chapter 2—and the quality of the resulting model depends heavily on initialization and other properties of the search.

The parameter optimization problem has much in common with the hard optimization problems of Chapter 2, but it also has one crucial difference. For the linear ordering problem, the optimization criterion is fixed—that's what makes it the linear ordering problem. But for parameter optimization, there are many possible criteria, each of which will lead to different performance on unseen data, even when it is possible to set $\boldsymbol{\theta}$ optimally for the criterion.

The work of this chapter has an additional quirk that distinguishes it from much other work in machine learning. This work attempts to find good weights for the linear ordering problem, which is itself computationally intractable. Most machine learning, on the other hand, works in a setting, for example, where it is possible, given a weight vector $\boldsymbol{\theta}$ and a question, to compute the answer that is optimal under the model. This issue arises throughout this chapter, starting with the simple perceptron learning algorithm in Section 4.7.

This work fits in with a general trend in natural language processing to tackle more sophisticated models in spite of their computational intractability. Methods such as Markov-Chain Monte Carlo (MCMC) and variational approximation have become increasingly popular. This chapter, especially Section 4.8, discusses some instances of prior work of this sort.

4.2 Prior Work

The particular problem of learning a score matrix for the linear ordering problem is unusual. The usual treatment of the LOP takes the matrix as given, derived directly from some quantitative data for the problem at hand.

One precedent for learning precedence relations as in the LOP is Cohen et al. (1999), which combined opinions from multiple ranking systems in the context of information retrieval. Their problem is an instance of the linear ordering problem, though they didn't explicitly acknowledge it as such.

They proposed an algorithm based on "Hedge" (Freund and Schapire, 1997), for learning a linear combination of experts, which in their setting meant search engines or, equivalently, retrieval results (in their experiments they used results from multiple related queries to the *same* search engine). They used a multiplicative update rule:

$$w_i^{t+1} = \frac{w_i^t \cdot \beta^{\mathrm{Loss}(R_i^t, F^t)}}{Z_t} \tag{4.1}$$

that depends on a rate parameter β and the loss of each expert R_i with respect to the true precedence information F^t on the current query t:

$$\mathrm{Loss}(R, F) = 1 - \frac{1}{|F|} \sum_{(u,v)\in F} R(u, v). \tag{4.2}$$

In their setting, the feedback F^t depends on the ordering proposed by the combination of experts, but if F^t is a total order then that dependence is vacuous. The update rule (4.1) for a given expert R_i depends on the other experts only through the normalization factor Z_t, which ensures that the weights of the experts sum to one.[1]

Whereas Cohen et al. had a relatively small set of experts, each of which constructed a relatively dense LOP matrix on its own, the setting of this chapter involves thousands or millions of very sparse features. Cohen et al. proved performance bounds relative to the single best expert, but those are useless in this setting, where each feature expresses an opinion for only a small subset of the matrix entries.

The problem with applying this approach to the machine translation reordering setting, besides the qualitative fact that it doesn't consider feature interactions, is that here "experts" are binary feature functions, which don't inherently express ordering preferences. Factored MIRA (Section 4.9) is a qualitatively similar approach.

A possible, related, approach to learning would be to treat each LOP matrix entry as a separate classification problem and try to give it the correct sign—positive for pairs of words that should stay in order, and negative for pairs that should reverse order. Were the problem separable, the result would be a matrix of the sort from Section 2.11.1 that unambiguously specified a total order, and the LOP would be trivial. Logistic regression, support vector machines, and other binary classifiers would be applicable. Even if the problem is not separable—which it likely isn't given a reasonable feature set—the LOP search may be able to find the correct ordering even in the presence of a few classification errors among the $\binom{n}{2}$ scores.

The problem with this approach is similar to the problem of converting the output of the model learned via support vector machines into a probability. If the classifier makes the correct prediction for all $\binom{n}{2}$ pairs of words, then the matrix is a total order and the resulting linear ordering problem is trivial. But if the classifier makes mistakes, as is inevitable, there is no mechanism in such a learning procedure for encouraging it to make the kind of mistakes from which the LOP search can recover.

The learning algorithms this chapter proposes and applies all take the linear ordering problem into account during parameter estimation, trying to find weights that

[1]Note that their update rule does not allow the weight of an expert to be negative. This ignores the possiblity that the expert might be negatively correlated with the feedback.

lead to good orderings during search. In theory, this allows the learner to deliberately ignore some matrix entries that would be hard to get right consistently, as long as other entries involving the same words can succeed at putting them in the right order.

4.3 Resources

Most of the tools that this chapter uses consist only of published algorithms, rather than software. However, data preparation and feature extraction made use of the following:

- The TIGER Treebank, a corpus of German news stories annotated with parts of speech and syntactic parses. Appendix B details this corpus, including its tag set and dependency labels derived from its grammatical functions.
- TreeTagger (Schmid, 1994, 1999) is a probabilistic part-of-speech tagger based on decision trees. It was used to tag the entire German Europarl data.
- MSTParser (McDonald, Crammer, and Pereira, 2005a; McDonald, Pereira, Ribarov, and Hajic, 2005c) is a dependency parser based on minimum-spanning tree algorithms. Unlike many traditional parsers, it can produce non-projective trees, which are appropriate for German.

Many of the tools from Chapter 3 are in use here as well: the Europarl corpus, the oracle English orderings described in Section 3.8.1, and the BLEU translation evaluation measure.

4.4 Features

In many ways, features are the most essential aspect of machine learning. To a large extent, any reasonable learning algorithm can achieve good performance given an adequate representation of the data. On the other hand, no learning algorithm, no matter how sophisticated, can be expected to overcome an inadequate representation.

Feature functions are a primary means of injecting human linguistic knowledge into machine learning for natural language applications. Although it is possible in principle to automatically discover features, cf. Della Pietra, Della Pietra, and Lafferty (1997); McCallum (2003), this remains an elusive goal for most applications. Far more common is to propose a large number of possible features, and use feature selection, cf. Guyon and Elisseeff (2003), if need be, to reduce them to a manageable set.

This section proposes a large number of potential features, and uses a very simple selection heuristic, namely to eliminate any features that occur less than some fixed number of times in the training data. This threshold is sometimes set to 5, and other times more conservatively to 100. This serves two largely orthogonal purposes:

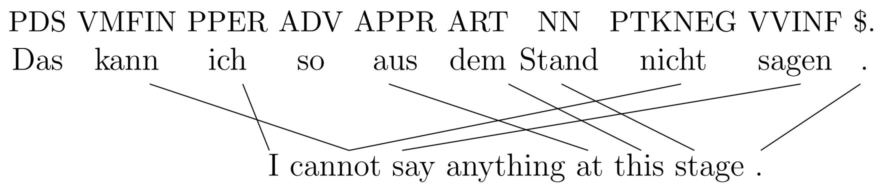
PDS VMFIN PPER ADV APPR ART NN PTKNEG VVINF $.
Das kann ich so aus dem Stand nicht sagen .
I cannot say anything at this stage .

Figure 4.1: An example German-English sentence pair, with automatically generated part of speech tags and a human-generated alignment. The matrix verb *kann* occurs in second position, but the negative particle *nicht* and the infinitive *sagen* are at the end, requiring long-distance reordering for translation to English *cannot say.* A good reordering would be *Das so ich kann nicht sagen aus dem Stand .* The placement of *Das* and *so* is of little importance since they don't appear in this translation.

1. it makes the feature set smaller, reducing the threat of overflowing the memory of the computer, and
2. it reduces the susceptibility of the model to overfitting—features that occur only a few times are likely to have chance correlations that would not generalize.

The feature functions used here consider both lexical and syntactic properties:

- Many of the characteristics of German from Section 3.4 are described in terms of simple parts of speech. It is reasonable, therefore, to use features that look only at those tags. These features are far more frequent than those involving particular word tokens.
- Much reordering may be related to the syntactic structure of the sentence. There are features derived from dependency parses that try to account for this possibility.
- Finally, it may be impossible to detect idiomatic constructions and some other phenomena without looking directly at the words themselves. Such features are included as well.

4.4.1 An Analogy to Dependency Parsing

At first glance, the linear ordering problem may seem a poor fit for a natural language application, where the use of n-gram language models and other computationally convenient local models assigns paramount importance to adjacency. If the LOP is appropriate at all for this task, its propriety can probably be understood through an analogy with the natural language task of dependency parsing.

In unlabeled non-projective dependency parsing, the task consists of assigning a parent to each word in a sentence. One word, and usually only one, is allowed to choose a distinguished root node as its parent, not corresponding to any word in

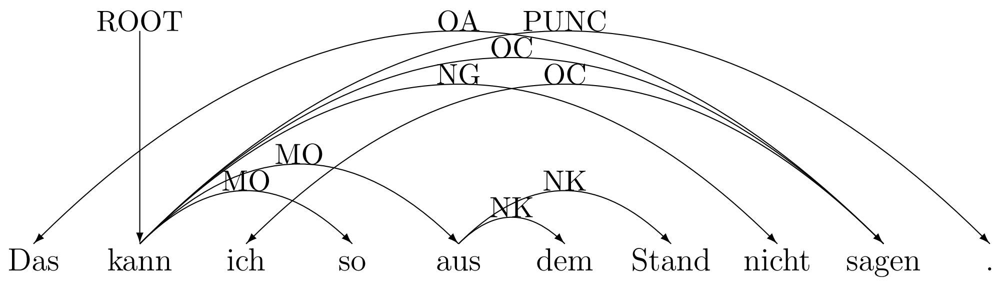

Figure 4.2: An automatic labeled dependency parse of the example sentence from Figure 4.1. The verb *kann* is the root, and its direct children are the modifiers *so* and *aus dem Stand*, the negative particle *nicht*, the verb *sagen*, and the period. The subject *ich* is mislabeled a clausal object.

the sentence. In the example from Figure 4.1, the verb *kann* would choose the root node, and the subject *ich* would choose *kann*, as its parent. Figure 4.2 shows the dependency parse that MSTParser produced for this sentence.

High-performing systems for this task have been built using features that look only at individual edges in the dependency tree (McDonald et al., 2005c). That is, the features that decide the parent of a given word never consider what words have been chosen as the parents or children of other words in the sentence. Instead, the features consider only properties of the sentence that can be derived independent of the dependency tree.

In this setting, every word i has a most preferred parent $\hat{p}_i$:

$$\hat{p}_i \overset{\text{def}}{=} \arg\min_{j \neq i} C_{j \to i}, \tag{4.3}$$

where $C_{j\to i}$ is the total cost of all features pertaining to the link from j to i. (Directed edges go from parents to children.) The complexity in this task comes from the constraint that the set of chosen edges must form a tree.

Similarly, the LOP cost matrix is an assignment of precedence costs to all pairs of words in the sentence independent of those words' interactions with other words. For every pair of words (a, b), the matrix assigns a cost to $a \prec b$ and a separate cost to $b \prec a$, neither of which depend on where other words, such as c, are placed relative to a and b. Every pair of words thus has a preferred order (unless the costs of $a \prec b$ and $b \prec a$ are equal).

The difference between the two tasks is that dependency parsing ultimately only incurs n costs. Notably, if the parser chooses $k \to j \to i$, it doesn't matter, in dependency parsing, if $k \to i$ costs more than $i \to k$. The greater complexity of the LOP comes from trying to maximize the sum of all $\binom{n}{2}$ preferences. The lack of an optimal substructure property means dynamic programming cannot solve the problem exactly.

4.4.2 Parts of speech

Each word in the German sentence is automatically labeled with a part of speech. German part of speech tags are from the set used in the TIGER corpus (Brants, Dipper, Hansen, Lezius, and Smith, 2002)—Appendix B details this corpus, and Table B.1 describes the tag set. Properties for the features of the pair (i, j) consider not only the parts of speech of words i and j, but also those of surrounding words:

p_{i-1} returns the part of speech of the word to the left of word i, or a special start of sentence tag if $i = 1$.

p_i returns the part of speech of word i.

p_{i+1} returns the part of speech of the word to the right of word i, which might be identically p_{j-1} or p_j.

p_b returns the part of speech of each word at a position b, $i < b < j$. This template therefore doesn't fire at all if $j = i + 1$, and fires multiple times if $j > i + 1$.

p_{j-1} returns the part of speech of the word to the left of word j, which might be identically p_i or p_{i+1}.

p_j returns the part of speech of word j.

p_{j+1} returns the part of speech of the word to the right of word j, or a special end of sentence tag if $j = n$.

Feature templates comprised of conjunctions of one or more of these properties are shown in Table 4.1. These are the same templates used for dependency parsing by McDonald, Crammer, and Pereira (2005b).

4.4.3 Words

The words themselves act as properties as well. These are far sparser than the parts of speech because the parts of speech are drawn from a relatively small fixed set, while the set of words is practically unbounded, and in the hundreds of thousands for the corpora used in this dissertation.

A compromise between the two, used by McDonald et al. (2005b), is the five-character prefix of the word. (For words five characters or fewer, the prefix is the entire word.) Combined with the words, there are the following properties:

w_i returns word i.

w_j returns word j.

$\mathbf{prefix}_i$ returns the five-character prefix of word i.

p_{i-1}	w_i	p_i	p_{i+1}	p_b	p_{j-1}	w_j	p_j	p_{j+1}
	√	√				√	√	
	√	√				√		
	√					√	√	
	√	√					√	
		√				√	√	
	√					√		
		√					√	
	√	√						
						√	√	
	√							
						√		
		√						
							√	
		√		√			√	
		√	√		√		√	
		√	√				√	
		√	√				√	√
		√					√	√
√		√					√	√
√		√					√	
√		√			√		√	
		√			√		√	

Table 4.1: LOP feature templates for $B[i, j]$ (w_i is the ith word, p_i its part of speech, and b matches any index such that $i < b < j$). Each of the above is also conjoined with the distance between the words, $j - i$, to form an additional feature template. Distances are binned into 1, 2, 3, 4, 5, > 5, and > 10.

prefix$_j$ returns the five-character prefix of word j.

Table 4.1 shows combined feature templates, using properties returning both words and parts of speech.

4.4.4 Dependencies

Each German sentence has also been labeled with a dependency parse, following McDonald, Lerman, and Pereira (2006), using non-projective minimum spanning tree algorithms.

The properties of the previous two sections—parts of speech and words—apply to single positions in the sentence to be reordered. Properties derived from dependency parses, on the other hand, apply to pairs of positions. A dependency parse consists

left parent	**right parent**	**left sibling**	**right sibling**
$\surd$			
	$\surd$		
		$\surd$	
			$\surd$
		$\surd$	$\surd$

Table 4.2: Dependency feature templates. Each templates is also conjoined with the distance between the words, as with the templates in Table 4.1. Note that only the sibling properties can occur in combination. Any of these templates *could* additionally be combined with part-of-speech and word features.

of a set of labeled directed edges between pairs of words.

It is therefore quite natural to have properties that return the label of a dependency, if one exists, between the pair of words implied by a LOP matrix entry. Two properties of this type naturally exist, corresponding to the two possible directions of the dependency. The same labels can be returned by different properties if the two words in question are siblings. Table 4.2 shows feature templates using the following properties:

left parent returns the dependency label when word j is word i's parent.

right parent returns the dependency label when word i is word j's parent.

left sibling returns the dependency label of $p \to i$ when i and j have the same parent p.

right sibling returns the dependency label of $p \to j$ when i and j have the same parent p.

4.5 Baseline

A simple baseline reordering model, and the one used for initialization of the learning procedures the rest of this chapter describes, is a naïve Bayes model. This model pretends that the features are all independent of one another and sets the weight of each one using a smoothed maximum likelihood criterion.

Let $\Phi_m = \{(\boldsymbol{w}, i, j) \mid \phi_m(\boldsymbol{w}, i, j) = 1\}$ be the set of word pairs in the training data for which feature m fires. Let Φ_m^+ be the subset of Φ_m for which the words stay in order, and Φ_m^- the subset for which the words reverse order. Then this model sets

$$\theta_m = \log\left(\left|\Phi_m^+\right| + \frac{1}{2}\right) - \log\left(\left|\Phi_m^-\right| + \frac{1}{2}\right). \tag{4.4}$$

Measure	**Correlation**
Oracle 1 BLEU	0.58*
Symmetric difference distance	0.42
Oracle 2 BLEU	0.38
Adjacencies	0.24

Table 4.3: Correlations of convergence criteria with English BLEU. BLEU score is measured using the preprocessing approach described in Section 3.8. Correlation is measured using Kendall's τ rank correlation of twelve different reorderings. Because of the small sample size, only the first is significantly non-zero at 95%.

This requires a bit of explanation. Because $B(\pi)$ is additive, it is appropriate to use log probabilities. If $\phi_m(\boldsymbol{w}, i, j) = 1$ then the contribution of feature m to $B[i, j]$ should be

$$\log \Pr(i \prec j \mid \phi_m = 1) = \log \frac{|\Phi_m^+|}{|\Phi_m|},$$

and its contribution to $B[j, i]$ should be

$$\log \Pr(j \prec i \mid \phi_m = 1) = \log \frac{|\Phi_m^-|}{|\Phi_m|}.$$

Because one of $B[i, j]$ and $B[j, i]$ must accrue to every permutation, both the maximum and the probability under a log-linear model (Section 4.11) are unchanged by setting $B'[j, i] = 0$ for $i < j$ and $B'[i, j] = B[i, j] - B[j, i]$. That is what (4.4) does. The denominators cancel, and the $\frac{1}{2}$ terms come from adding a pseudocount to both subsets in order to mitigate overfitting, particularly when one of the subsets is empty.

4.6 Convergence Criteria

A key component of the learning methods described in the following sections is the stopping criterion. The procedure that most of the learning methods use is to measure performance on a held-out set after each iteration through the training data. When performance on the held-out set starts to degrade, the procedure stops and reverts to the highest-scoring model. Experiments running additional iterations of learning with several models showed no further improvement, so stopping after the first degradation seems reliable.

Table 4.3 shows four different possible performance measures, and their correlations with the resulting BLEU score on the German-English translation task, as described in Section 3.8:

Oracle BLEU measures the BLEU of the candidate German ordering against a reference German ordering. Because the two orderings contain the same words, the unigram precision of this score is always 100%, and only higher-order precisions vary.

Symmetric difference distance counts the number of pairs of words in the candidate ordering that are in order according to the reference ordering. The score is between 0 and $\binom{n}{2}$ for a sentence of length n. Section 2.11.2 discussed this score in detail.

Adjacencies counts the number of pairs of words adjacent in the reference ordering that are also adjacent in the candidate ordering. This score is between 0 and $n-1$ for a sentence of length n, and is identical to the bigram precision count of BLEU.

4.7 Perceptrons

The perceptron algorithm (Rosenblatt, 1958) is a simple supervised procedure for learning model weights. It considers labeled training examples one at a time, and updates the current parameters to increase the weight of the correct labeling and decrease the weight of the labeling that was optimal according to the current model. The perceptron, and a variation that the next section introduces, proves to be the most reliable learning method for the dissertation's task, as Section 4.12 describes. The performance of the two variations is quite similar.

Let x represent a given input, y^* the corresponding correct label, $\boldsymbol{\phi}(\cdot,\cdot)$ a function from inputs and labels to features, and $\boldsymbol{\theta}$ the corresponding feature weights. Let

$$\hat{y} \overset{\text{def}}{=} \arg\max_{y} \boldsymbol{\theta}^{(t)} \cdot \boldsymbol{\phi}(x, y) \tag{4.5}$$

be the best label according to the model at time t. Then the update is as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha \left[\boldsymbol{\phi}(x, y^*) - \boldsymbol{\phi}(x, \hat{y})\right], \tag{4.6}$$

where α is a hyper-parameter called the learning rate. These updates proceed to form a sequence of models, from the initial parameters $\boldsymbol{\theta}^{(0)}$ to some final parameters $\boldsymbol{\theta}^{(T)}$, where T is determined by some convergence criterion, as in Section 4.6.

The perceptron has no explicit objective function. However, its update is a step along the gradient of the following objective:

$$\max_{\boldsymbol{\theta}} \left[\boldsymbol{\theta} \cdot \boldsymbol{\phi}(x, y^*) - \boldsymbol{\theta} \cdot \boldsymbol{\phi}(x, \hat{y})\right], \tag{4.7}$$

which attempts to maximize the difference between the score of the true labeling and the score of the labeling the model prefers. This maximum occurs at zero, when the model prefers the true label. If the data are separable and the algorithm achieves a setting of $\boldsymbol{\theta}$ such that the classifier no longer makes any mistakes, then no further learning will occur.

It is also possible to characterize the loss function that the perceptron uses. It is a $\{0, 1\}$ loss function—the loss of the true labeling is zero, and the loss of any other labeling is one. The perceptron thus does not distinguish between good and bad wrong answers. This coarse granularity is especially inappropriate for structured labeling problems, where y^* is complex, and partial credit may be meaningful.

The perceptron algorithm as given in (4.6) is not exactly applicable to the setting of this chapter. Here, a "labeling" is a permutation. Efficiently computing $\hat{y}$, the best permutation according to the model, is intractable. However, replacing the model's true optimum, $\hat{y}$, with the local optimum achieved during search allows the perceptron updates to proceed in spite of possible search error. In a sense, this replaces the exact linear ordering problem model with a black box—a heuristic procedure whose inner workings are inscrutable to the learning procedure. One particular attraction of the perceptron algorithm is that it can work with such black boxes unaltered.

The voted perceptron and the averaged perceptron (Freund and Schapire, 1998) are two alternatives to the standard perceptron as described above, both intended to prevent overfitting. The updates remain the same in both cases, but the final models differ. In the case of the voted perceptron, the final model is a vote among all instances of the model $\{\boldsymbol{\theta}^{(0)}, \boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(T)}\}$. This alternative is not obviously applicable to the LOP setting, because possible labels are the set of permutations of the input. However, the averaged perceptron, which uses a single model,

$$\bar{\boldsymbol{\theta}} = \frac{\sum_{i=0}^{T} \boldsymbol{\theta}^{(i)}}{T+1}, \tag{4.8}$$

does apply.

Averaging the model after the fact bears a close resemblance to a method commonly used for stochastic gradient descent (Bottou, 2004) (see Section 4.11)—that of setting the learning rate α proportional to $\frac{1}{t_0+t}$, where t counts the training examples. In a sense, the SGD approach computes a running average, while the averaged perceptron approach waits until the end.

Freund and Schapire (1998) prove a generalization bound for the voted perceptron. Collins (2002) discusses justification for the averaged perceptron. Averaged perceptron can be interpreted as a cheaper alternative to the voted perceptron.

4.8 Search-based methods

Learning as Search Optimization (LaSO) (Daumé III and Marcu, 2005) and Searn (Daumé III, Langford, and Marcu, 2007) are two parameter optimization methods that apply search in the context of computationally hard problems. This approach has much appeal, but the methods don't transfer directly to the VLSN search for the linear ordering problem. LaSO and Searn both use *constructive* search, rather than *local* search—see Section 2.5 for a discussion of the difference.

Nevertheless, the basic idea of taking the search into account in the learning algorithm does transfer. This section describes a "search perceptron", which may make multiple updates to the parameter vector for a given sentence at several stages in the search.

Definition 4.1 *A* **search trajectory** *is the sequence of permutations that a greedy local search procedure visits, starting with the initial permutation $\pi^{(0)}$ and ending with the local maximum $\pi^{(T)}$, for some T.*

There are two possible trajectories that search can follow:

1. the model trajectory,
$$\pi^{(t+1)} = \max_{\pi \in \mathcal{N}(\pi^{(t)})} B(\pi), \text{ or} \tag{4.9}$$

2. the loss trajectory,
$$\pi^{(t+1)} = \max_{\pi \in \mathcal{N}(\pi^{(t)})} B^{\tau}_{\pi^*}(\pi). \tag{4.10}$$

In either case, the search perceptron update is the same:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \left[\boldsymbol{\phi}\left(\boldsymbol{w}, \arg \max_{\pi \in \mathcal{N}(\pi^{(t)})} B^{\tau}_{\pi^*}(\pi)\right) - \boldsymbol{\phi}\left(\boldsymbol{w}, \arg \max_{\pi \in \mathcal{N}(\pi^{(t)})} B(\pi)\right)\right], \tag{4.11}$$

where $\boldsymbol{\theta}$ are the current parameters and $\boldsymbol{\theta}'$ the new ones. The index is suppressed because it depends on the number of updates that occur for each example, and is therefore unnecessarily complicated to write down.

The idea behind this procedure is to find parameters that encourage the search to go in the right direction at each step. Using the model trajectory lets the model decide how to proceed, but tells it at each step where it really ought to go, while using the loss trajectory pulls the search straight to the target permutation.

The search perceptron bears a resemblance to the *local updating* strategy introduced by Liang, Bouchard-Côté, Klein, and Taskar (2006a) and further successfully applied by Watanabe, Suzuki, Tsukada, and Isozaki (2007) and Arun and Koehn (2007). This update rule finds the best candidate in a K-best list according to some variant of the BLEU score.[2]

[2]Recall that sentence-level BLEU is problematic because of potential zeroes—see Section 4.11.4.

4.9 MIRA

This section describes large-margin alternatives to the perceptron update rule. These methods have not undergone empirical validation.

The margin-infused relaxed algorithm (Crammer and Singer, 2003) uses the same basic procedure as the perceptron but replaces the simple perceptron update rule (4.6) with an explicit large-margin update. As a result of the similarity to perceptron, voted and averaged variants are possible here as well. The new update rule is a quadratic program, which seeks:

- the minimum change in the norm of the parameters
- subject to the constraint that the correct label is preferred to other labels by some margin.

An important difference between MIRA and the perceptron is that MIRA has no learning rate hyper-parameter α. The loss function, which will determine the margin for each incorrect label, is a hyperparameter of sorts, but there is usually an obvious one available to use, and its scale is not important.

In practice, the aforementioned constraint can take on many different forms. The closest variant to the perceptron would use[3]

$$\boldsymbol{\theta}^{(t+1)} \cdot [\boldsymbol{\phi}(x, y^*) - \boldsymbol{\phi}(x, \hat{y})] \geq 1 \tag{4.12}$$

to constrain the standard MIRA objective function,

$$\min \frac{1}{2} \left|\left|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right|\right|^2 . \tag{4.13}$$

Notice that (4.12) uses the *current* model $\boldsymbol{\theta}^{(t)}$ to compute the model's preferred label $\hat{y}$ as in (4.5), but constrains the dot product of the *new* model $\boldsymbol{\theta}^{(t+1)}$.

As long as there is at least one difference between $\boldsymbol{\phi}(x, y^*)$ and $\boldsymbol{\phi}(x, \hat{y})$ when $\hat{y} \neq y^*$, the single constraint (4.12) is achievable. This *separability* is a consequence of the on-line algorithm: MIRA only looks at one sentence at a time. However, the following sections introduce more complex sets of contraints that may not always be separable. In that case, it is necessary to introduce *slack variables* into the optimization criterion (4.13) and the constraints. This is a standard procedure, familiar in the literature on support vector machines. The constraints are given here as though the slack variables are unnecessary, however.

[3]Note that no update is necessary if $\hat{y} = y^*$.

4.9.1 K-best MIRA

Rather than just the one best permutation, it is possible to compute the K best permutations, according to the current model, in some neighborhood. An especially efficient method is to adapt the lazy algorithm of Huang and Chiang (2005) to the normal-form grammars of Section 2.9. It is essential to use normal form in order to avoid multiple spurious derivations of the same permutation in the K-best list.

Let $\{\hat{y}_1, \ldots, \hat{y}_{K_t}\}$ be the K-best list. $K_t \leq K$ is the size of the K-best list for sentence t, which is limited by the size of the neighborhood and may be less than K especially if K is very large or n is small for sentence t. Let $L(y^*, y)$ measure the loss of y with respect to y^*. Then an alternative to the constraint of (4.12) is the set of constraints

$$\boldsymbol{\theta}^{(t+1)} \cdot [\boldsymbol{\phi}(x, y^*) - \boldsymbol{\phi}(x, \hat{y}_i)] \geq L(y^*, \hat{y}_i),\ \forall i : 1 \leq i \leq K_t. \tag{4.14}$$

This is called K-best MIRA. Any loss function, such as the convergence criteria of Section 4.6, can be applied, because the K-best list is an enumeration of permutations.

Section 4.10 discusses a different situation, where the requirements on the loss function are necessarily more strict. Section 4.11.3 addresses the related question of minimizing the expected loss, with similar requirements.

4.9.2 Factored MIRA

McDonald et al. (2005c) used a variant of the training procedure described above called "Factored MIRA". Instead of constraining the weights of features of an entire parse, they factored the features according to the edges used to construct the parse. The contraints then required, for each word, that the difference between the weight of the true parent and the weight of each other possible parent be larger than a margin of one.

A similar approach is possible for the linear ordering problem. The constraints can simply require that

$$B[\ell, r] - B[r, \ell] \geq 1,\ \forall \ell \prec r \in \pi^*. \tag{4.15}$$

Were it possible to learn such weights, the resulting linear ordering problems would be total orders (Section 2.11.1) and search would be trivial. The drawbacks of this approach are discussed earlier in this chapter, in Section 4.2.

4.10 Max-Margin Parsing

An even more sophisticated application of MIRA would use the marginals from Taskar, Klein, Collins, Koller, and Manning (2004) to collapse an exponential number of constraints to a polynomial number and ensure that the correct permutation was better than any other permutation in its neighborhood by an amount proportional to some loss function. A simple measure of loss can be derived from a LOP matrix with entries in $\{0,1\}$ that assigns a loss of 0 to the correct ordering and $\binom{n}{2}$ to its reverse—the same matrix from Section 2.11.1. This loss function counts the number of pairs out of order with respect to the correct permutation. This section explores this possibility in detail.

First, recall (2.3), and let

$$\boldsymbol{\phi}\left(\boldsymbol{w},\pi\right) \stackrel{\text{def}}{=} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \boldsymbol{\phi}\left(\boldsymbol{w},\pi_i,\pi_j\right), \tag{4.16}$$

such that $B(\pi) \equiv \boldsymbol{\theta}\cdot\boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)$.

Adapting the method of Taskar et al. (2004) to the setting of MIRA for the linear ordering problem leads to the following update:

$$\begin{aligned} \min \frac{1}{2}\left|\left|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right|\right|^2 \text{ subject to} \\ \boldsymbol{\theta}^{(t+1)}\cdot\left[\boldsymbol{\phi}\left(\boldsymbol{w},\pi^*\right) - \boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)\right] \geq L(\pi^*,\pi),\ \forall\pi\in\mathcal{N}, \end{aligned} \tag{4.17}$$

given the current pair $(\boldsymbol{w},\pi^*)$ of an input sentence and its target permutation, and some neighborhood $\mathcal{N}$. For now, the permutation from which that neighborhood derives will remain unspecified.

Introduce a Lagrange multiplier λ_π for each $\pi\in\mathcal{N}$. For the VLSNs, there are an exponential number of these. The following section will take care of this problem. Rewrite (4.17) as a Lagrangian,

$$\min \frac{1}{2}\left|\left|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right|\right|^2 - \sum_{\pi\in\mathcal{N}} \lambda_\pi \left[\boldsymbol{\theta}^{(t+1)}\cdot\left[\boldsymbol{\phi}\left(\boldsymbol{w},\pi^*\right) - \boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)\right] - L(\pi^*,\pi)\right], \tag{4.18}$$

subject to $\lambda_\pi \geq 0,\ \forall\pi\in\mathcal{N}$, and solve

$$0 = \nabla_{\boldsymbol{\theta}^{(t+1)}} = \left[\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right] - \sum_{\pi\in\mathcal{N}} \lambda_\pi\left[\boldsymbol{\phi}\left(\boldsymbol{w},\pi^*\right) - \boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)\right]. \tag{4.19}$$

This leads to the update rule

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \sum_{\pi\in\mathcal{N}} \lambda_\pi\left[\boldsymbol{\phi}\left(\boldsymbol{w},\pi^*\right) - \boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)\right], \tag{4.20}$$

which is reminiscent of the perceptron update, except now each $\pi\in\mathcal{N}$ may contribute, instead of just $\hat{\pi}$.

The substitution of (4.20) into the Lagrangian (4.18) results in the dual problem,

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \sum_{\pi\in\mathcal{N}} \lambda_\pi \left[L(\pi^*,\pi) - \boldsymbol{\theta}^{(t)}\cdot\left[\boldsymbol{\phi}\left(\boldsymbol{w},\pi^*\right) - \boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)\right]\right] \\ -\frac{1}{2}\left|\left|\sum_{\pi\in\mathcal{N}} \lambda_\pi\left[\boldsymbol{\phi}\left(\boldsymbol{w},\pi^*\right) - \boldsymbol{\phi}\left(\boldsymbol{w},\pi\right)\right]\right|\right|^2. \end{aligned} \tag{4.21}$$

The first term forces λ_π to zero for any π that already satisifes the margin constraint, and otherwise encourages λ_π to be large. The second term, on the other hand, encourages all λ_π to be small. Intuitively, this captures the tradeoff between getting the answers right and keeping the update small.

4.10.1 Factored Dual

The insight of Taskar et al. (2004) was that the exponential number of variables λ_π could be reduced to a polynomial number if the loss function was well-behaved. The symmetric difference distance $\delta(\pi^*, \pi)$, defined in (2.58), Section 2.11.2, is well behaved, because it decomposes in the same way that any B matrix does. Let $\vec{\delta}_{i,j,k}$ and $\overleftarrow{\delta}_{i,j,k}$ play the role of $\vec{\gamma}_{i,j,k}$ and $\overleftarrow{\gamma}_{i,j,k}$ from (2.24) and (2.25) for the loss function $\delta(\pi^*, \pi)$, and let

$$\vec{\boldsymbol{\phi}}_{i,j,k} \overset{\text{def}}{=} \sum_{\ell \in (i,j)} \sum_{r \in (j,k)} \boldsymbol{\phi}\left(\boldsymbol{w}, \ell, r\right), \tag{4.22}$$

$$\overleftarrow{\boldsymbol{\phi}}_{i,j,k} \overset{\text{def}}{=} \sum_{\ell \in (i,j)} \sum_{r \in (j,k)} \boldsymbol{\phi}\left(\boldsymbol{w}, r, \ell\right), \tag{4.23}$$

such that $\vec{\gamma}_{i,j,k} = \boldsymbol{\theta}^{(t)} \cdot \vec{\boldsymbol{\phi}}_{i,j,k}$ and $\overleftarrow{\gamma}_{i,j,k} = \boldsymbol{\theta}^{(t)} \cdot \overleftarrow{\boldsymbol{\phi}}_{i,j,k}$. Now, let

$$\begin{aligned} \vec{\mu}_{i,j,k} &= \sum_{\pi \in \mathcal{N}} \lambda_\pi I((i,k) \to (i,j)\ (j,k) \in \pi), \\ \overleftarrow{\mu}_{i,j,k} &= \sum_{\pi \in \mathcal{N}} \lambda_\pi I((i,k) \to (j,k)\ (i,j) \in \pi), \end{aligned} \tag{4.24}$$

where $I(\cdot)$ is an indicator function. Note that normal form will be necessary to enforce a one-to-one mapping between permutations and derivation trees. Finally, let S be the set of all four-tuples $(i, j, k, \to / \leftarrow)$, for convenience, such that each of δ, $\boldsymbol{\phi}$, and μ can be indexed by elements $s \in S$. Then the factored dual is

$$\max_{\boldsymbol{\mu}} \sum_{s \in S} \mu_s \left[\delta_s - \boldsymbol{\theta}^{(t)} \cdot \left[\frac{\boldsymbol{\phi}\left(\boldsymbol{w}, \pi^*\right)}{n-1} - \boldsymbol{\phi}_s \right] \right] - \frac{1}{2} \left\| \sum_{s \in S} \mu_s \left[\frac{\boldsymbol{\phi}\left(\boldsymbol{w}, \pi^*\right)}{n-1} - \boldsymbol{\phi}_s \right] \right\|^2, \tag{4.25}$$

subject to the following marginal constraint for each span (i, k):

$$\sum_{j=i+1}^{k-1} \left[\vec{\mu}_{i,j,k} + \overleftarrow{\mu}_{i,j,k} \right] = \sum_{o=0}^{i-1} \left[\vec{\mu}_{o,i,k} + \overleftarrow{\mu}_{o,i,k} \right] + \sum_{o=k+1}^{n} \left[\vec{\mu}_{i,k,o} + \overleftarrow{\mu}_{i,k,o} \right], \tag{4.26}$$

which ensures that $\boldsymbol{\mu}$ corresponds to some $\boldsymbol{\lambda}$. This is a quadratic program in $2\binom{n+1}{3}$ variables. The factor of $n-1$ on the features of the target permutation comes from the fact that

$$\sum_{s \in S} \mu_s = (n-1) \sum_{\pi \in \mathcal{N}} \lambda_\pi,$$

since there are $n-1$ nodes in every permutation tree.

4.11 Likelihood

It is possible to interpret LOP matrix scores as probabilities by introducing the following transformation:

$$\Pr(\pi; B) = \frac{\exp\left(B(\pi)\right)}{\displaystyle\sum_{\pi' \in \Pi_n} \exp(B(\pi'))} \tag{4.27}$$

The exponential function ensures that the numerator is positive, and the denominator, $Z(B)$, provides normalization onto the interval $[0, 1]$. The probability could also include a scaling factor on each $B(\pi)$ to sharpen or flatten the distribution. This section ignores that factor in derivations for simplicity.

Define the training data $\mathcal{D} = \{(\pi^{(1)}, B^{(1)}), (\pi^{(2)}, B^{(2)}), \ldots, (\pi^{(D)}, B^{(D)})\}$ as ordered pairs consisting of a target permutation and a LOP matrix derived from features of a sentence according to the linear model (2.3). Then the log likelihood of the data is

$$L = \sum_{(\pi^*, B) \in \mathcal{D}} \log \Pr(\pi^*; B), \tag{4.28}$$

and the gradient of the log likelihood with respect to the matrix entry $B^{(t)}[i, j]$ is

$$\nabla_{t,i,j} L = \frac{\partial L}{\partial B^{(t)}[i, j]} = \sum_{(\pi^*, B) \in \mathcal{D}} \frac{\partial}{\partial B^{(t)}[i, j]} \log \Pr(\pi^*; B) = \frac{\partial}{\partial B^{(t)}[i, j]} \log \Pr(\pi^{(t)}; B^{(t)}). \tag{4.29}$$

Because $B^{(t)}[i, j]$ is computed according to a linear model, it is trivial to convert this gradient into a gradient of each entry in the weight vector $\boldsymbol{\theta}$.

$$\begin{aligned}
\frac{\partial \log \Pr(\pi^*; B)}{\partial B[i, j]} &= \frac{\partial}{\partial B[i, j]} \left(B(\pi^*) - \log \sum_{\pi \in \Pi_n} \exp(B(\pi)) \right) \\
&= I(i \prec j \in \pi^*) - \frac{\displaystyle\sum_{\pi \in \Pi_n} \exp(B(\pi)) I(i \prec j \in \pi)}{\displaystyle\sum_{\pi \in \Pi_n} \exp(B(\pi))} \\
&= I(i \prec j \in \pi^*) - \mathrm{E}\left[I(i \prec j \in \pi); B \right],
\end{aligned} \tag{4.30}$$

where I is an indicator function that returns 1 if its argument is true, and 0 otherwise.

Unfortunately, it is intractable to compute these expectations over all $n!$ permutations $\pi \in \Pi_n$, so using the likelihood of the true permuation π^* directly as an optimization criterion is out of the question. One potential alternative is to use sampling to compute approximate expectations. Section 4.11.1 explores this possibility, which is closest to the exact likelihood objective. Section 4.11.2 uses a different approximation, replacing the sum over all $n!$ permutations with just those in a neighborhood of π^*. Finally, Section 4.11.3 proposes an alternative neighborhood-based optimization criterion that can be tailored to specific loss functions.

4.11.1 Sampling

This section presents another possible approach to learning that has not yet seen empirical validation. This particular method, while attractive because of its arbitrarily close approximation of the true objective function, is likely to require far more computation time than the other methods this chapter proposes.

The Metropolis-Hastings algorithm (Hastings, 1970) is a Markov Chain Monte Carlo sampling method that generalizes the simpler Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller, 1953). This section proposes application of the Metropolis-Hastings algorithm to the neighborhoods computed by the normal-form grammars of Section 2.9. As with K-best MIRA in Section 4.9.1, normal form is crucial. Here it is necessary to compute the distribution over permutations rather than the possibly multiple derivations of those permutations under the grammar.

The Metropolis-Hastings algorithm is instantiated with a proposal distribution $Q(\pi \mid \pi^{(t)}; B)$, where $\pi^{(t)}$ is the current permutation at the tth step. Given a neighborhood $\mathcal{N}$ and a normal-form grammar for parsing it, the available proposal distribution is

$$Q(\pi \mid \pi^{(t)}; B) = \Pr\left(\pi \mid \mathcal{N}(\pi^{(t)}); B\right) = \frac{\exp(B(\pi))}{\displaystyle\sum_{\pi' \in \mathcal{N}(\pi^{(t)})} \exp(B(\pi'))}. \tag{4.31}$$

This differs from the full distribution (4.27) in that the sum in the denominator is limited to the permutations in the neighborhood.

The algorithm computes a sample π' from this proposal distribution and accepts it with probability

$$\min\left(1, \frac{\Pr(\pi'; B) Q(\pi^{(t)} \mid \pi'; B)}{\Pr(\pi^{(t)}; B) Q(\pi' \mid \pi^{(t)}; B)}\right). \tag{4.32}$$

If accepted, π' becomes $\pi^{(t+1)}$, otherwise $\pi^{(t+1)}$ is another sample of $\pi^{(t)}$. The probabilities $\Pr(\pi'; B)$ and $\Pr(\pi^{(t)}; B)$ both have the same normalization factor, $Z(B)$, which cancels, so they can be replaced by the numerator of (4.27), $\exp(B(\pi'))$ and $\exp(B(\pi^{(t)}))$.

The acceptance probability requires computation of two neighborhood conditional probabilities. However, the parse forests can be reused from step to step. If π' is accepted, then its parse forest—needed to compute the denominator of $Q(\pi^{(t)} \mid \pi'; B)$ from (4.31) in the current step—can be reused to compute a new sample at the next step. If π' is rejected, though, its parse forest is wasted in this respect.

Each sample contributes a count to either $i \prec j$ or $j \prec i$. In the limit, these are guaranteed to converge to the true expectations under the full distribution. In practice, deciding when the approximation is good enough is a field of study unto itself, beyond the scope of this dissertation.

Popping back up one level to the parameter estimation problem, the procedure is as follows:

1. Run Metropolis-Hastings on each instance $(\pi^*, B) \in \mathcal{D}$, computing expectations of $B[i, j]$ for each pair of words (i, j). Subtract the approximate expectation from the first term of (4.30) to compute each gradient.

2. Propagate the gradients from each $B[i, j]$ to the entries in $\boldsymbol{\theta}$.

This procedure goes into an outer loop of parameter optimization, which generally requires many evaluations of the gradients from many settings of the parameter vector in order to arrive at the optimum.

An alternative is to replace the outer loop with an on-line update, the leading candidate being stochastic gradient descent (SGD), which updates the parameters by some learning rate times their gradient on the current problem instance—see Section 4.7.

4.11.2 Contrastive Likelihood

Rather than maximize the likelihood of the true permutations, it may suffice to maximize their likelihood given their own neighborhoods—$\Pr(\pi^* \mid \mathcal{N}(\pi^*); B)$ as defined in (4.31). This is a sort of *contrastive* likelihood, inspired in part by the unsupervised learning of Smith and Eisner (2005).

This can be considered an approximation to the "true" objective function, or it can be thought of as an alternate objective. In the latter case, it is interesting to observe the following property of the very large-scale neighborhoods.

Theorem 4.1 *Exactly half the permutations in BlockInsert$_n^*(\pi)$ have $\ell \prec r$ and half $r \prec \ell$, for any $1 \leq \ell < r \leq n$.*

Even though the neighborhood of π^* is a relatively small subset (though itself exponentially large) of the entire set of permutations, each matrix entry is equally well represented. This, along with computational tractability, is a strong point in favor of contrastive likelihood.

In contrast, the arrangement of other items may conspire to drastically change the expectation of some orderings as compared to the full distribution. This is a constant danger with this objective. Ultimately, it is an empirical question whether it is appropriate, and performance probably depends on the problem.

Proof of Theorem 4.1: Let T be a normal-form permutation tree in $G_B^{\text{NF}}(\pi)$ from Section 2.9.2 for which $\ell \prec r$. This proof introduces a transformation $\mathcal{R}_{\ell,r}$ with these three essential properties:

- $\mathcal{R}_{\ell,r}(T)$ is in $G_B^{\text{NF}}(\pi)$,
- the yield of $\mathcal{R}_{\ell,r}(T)$ reverses $\ell \prec r$ to $r \prec \ell$, and
- $\mathcal{R}_{\ell,r}(\mathcal{R}_{\ell,r}(T)) = T$.

The first and second properties prove that there exists a permutation in the neighborhood with $r \prec \ell$ for every one that has $\ell \prec r$. The third property proves that correspondence to be one-to-one, which suffices to prove the theorem. The transformation $\mathcal{R}_{\ell,r}$ has the following steps:

1. Let N be the smallest node in T that governs both ℓ and r, i.e. N determines the relative order of ℓ and r. N has some span (i, k).
2. Let $\mathcal{B}_k$, for "branch", be the set of nodes, including N, whose spans end at k.
3. Reverse the orientation of each node in $\mathcal{B}_k$—in-order nodes become reverse nodes, and reverse nodes become in-order.

It suffices to show that this transformation has the three properties indicated above.

The second property is clear—reversing N changes the order of ℓ and r. The third property is also straightforward. The transformation changes orientations of nodes only, it doesn't change the structure of the tree. The same node N is still the smallest to govern both ℓ and r, and the set $\mathcal{B}_k$ also hasn't changed. Therefore the outer transformation of $\mathcal{R}_{\ell,r}(\mathcal{R}_{\ell,r}(T))$ reverses the same set of nodes as the inner transformation, and results in T again.

In order to prove that $\mathcal{R}_{\ell,r}$ satisfies the first necessary property, consult G_B^{NF}. Left children in the grammar are always unconstrained. Let M be the topmost node in $\mathcal{B}_k$. M must either be a left child or span $(0, n)$, because were it a right child, its parent would be in $\mathcal{B}_k$ as well. Therefore, reversing the orientation of M does not affect normality of the (possibly empty) tree outside it.

Changing M requires changing its right child in order to preserve normality. Because its right child's span ends at k, that right child is also in $\mathcal{B}_k$ and does also change. The left child of M is irrelevant, because it is unconstrained by normal form.

Now that the tree outside M's right child is normal, apply the argument of the previous paragraph recursively to M's right child. The base case of the recursion is the leaf π_k, which can have its label—either $\vec{S}_{k-1,k}$ or $\overleftarrow{S}_{k-1,k}$—changed arbitrarily without affecting its normality. The conclusion is that $\mathcal{R}_{\ell,r}(T) \in G_B^{\text{NF}}$. $\square$

Figure 4.3 shows an example transformation $\mathcal{R}_{3,6}$ applied to an example permuation tree.

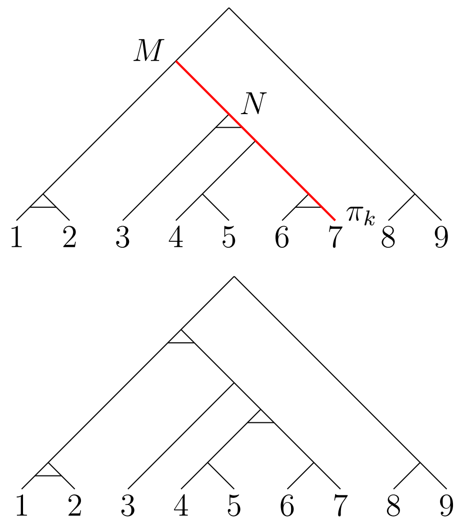

Figure 4.3: An example permutation tree transformation reversing the order of two items. The yield of the first tree is 2 1 4 5 7 6 3 8 9, which puts $6 \prec 3$. To generate the corresponding tree with $3 \prec 6$, find the smallest node governing both 3 and 6—the node labeled N in the tree—and reverse the orientation of all nodes along the right "branch"—the thick red line from M to π_k—that contains that node. The result is the second tree, which is also normal, and whose yield is 3 6 7 4 5 2 1 8 9. Performing the same transformation on the second tree produces the original tree.

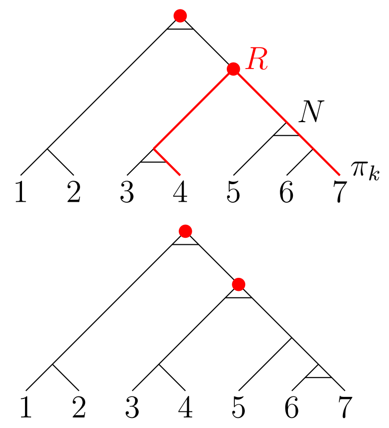

Figure 4.4: An example permutation tree transformation for $G_{B\leq 2}^{\mathrm{NF}}$. The yield of the first tree is 4 3 6 7 5 1 2, which puts $7 \prec 5$. To generate the corresponding tree with $5 \prec 7$, find N governing 5 and 7 and reverse the orientation along the right "branch", stopping at the red node R. Also, reverse the orientation of R's *left* child and the right "branch" down from there. The result is the second tree, which is also normal, and whose yield is 5 7 6 3 4 1 2. Again, performing the same transformation on the second tree produces the original tree.

Corollary 4.2 *Exactly half the permutations in $L(G_{B\leq w}^{\mathrm{NF}}(\pi))$ have $\ell \prec r$ and half $r \prec \ell$, for any $1 \leq \ell < r \leq n$, and for any $w \geq 1$.*

Proof: This grammar has red nodes in addition to those of G_B^{NF}. Modify the tree transformation $\mathcal{R}_{\ell,r}$ to ignore nodes in $\mathcal{B}_k$ above the first red node. Because red nodes can have either orientation, there is no effect on the normalcy of the tree above it. However, red nodes require their left children, which are always narrow and therefore never also red, to have the opposite type. Node swapping must therefore propagate to the left child as well, unless it is a leaf. In any case, the structure of the tree again doesn't change, and applying the same procedure again reverses the swaps, so the three properties from the proof of Theorem 4.1 are retained. □

Figure 4.4 shows an example of this modified tranformation, $\mathcal{R}_{5,7}$ applied to another example permutation tree.

Section 4.12.1 applies stochastic gradient descent to maximization of this objective function.

4.11.3 Expected Loss Minimization

A more significant departure from the intractable likelihood maximization objective is minimization of the expectation of some loss function, where the expectation is taken only over the conditional distribution of permutations under the model given some neighborhood. Let $L(\pi^*, \pi)$ measure the loss of π with respect to π^*, as in Section 4.9.1.[4] Then the objective is

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{(\pi^*, B) \in \mathcal{D}} \mathrm{E}_{\pi \in \mathcal{N}(\pi^*)} \left[L(\pi^*, \pi); B \right], \tag{4.33}$$

where

$$\mathrm{E}_{\pi \in \mathcal{N}(\pi^*)} \left[L(\pi^*, \pi); B \right] = \sum_{\pi \in \mathcal{N}(\pi^*)} L(\pi^*, \pi) \Pr\left(\pi \mid \mathcal{N}(\pi^*); B\right). \tag{4.34}$$

This objective is reminiscent of minimum Bayes risk (MBR) decoding (Goel and Byrne, 2000; Kumar and Byrne, 2004). Whereas MBR chooses a single minimum expected loss output given the model parameters, this objective attempts to choose model parameters that minimize expected loss over all possible outputs.

In order to compute the expected loss of the exponentially many permutations in the very large-scale neighborhoods, it is useful to introduce the expectation semiring of Eisner (2002).

Definition 4.2 *The* **expectation semiring** *is* $\langle \mathbb{R} \times \mathbb{R}, \oplus, \otimes, (0,0), (1,0) \rangle$.[5] *The weight of an event* $\mathcal{E}$ *is the ordered pair* $(\Pr(\mathcal{E}), \Pr(\mathcal{E})\,\mathrm{E}[f \mid \mathcal{E}])$ *representing probability and expectation of some function* f*, respectively. The operators on these ordered pair weights are defined as follows:*

$$\begin{aligned} (p_1, v_1) \oplus (p_2, v_2) &\overset{\text{def}}{=} (p_1 + p_2, v_1 + v_2), \text{ and} \\ (p_1, v_1) \otimes (p_2, v_2) &\overset{\text{def}}{=} (p_1 p_2, p_1 v_2 + p_2 v_1). \end{aligned}$$

If w is an element of this semiring, then let $p(w)$ return the first element of the ordered pair, and let $v(w)$ return the second.

In the setting of this section, the events are instances of grammar rules, which may or may not occur in any given derivation from the grammar. The weights compute the probabilities of those grammar rules occurring, and the expectation of the loss function.

Because computation of the expected loss uses dynamic programming with a normal-form neighborhood grammar, the loss function must decompose nicely. It is most convenient if the loss can itself be expressed as a linear ordering problem, as is the case with the symmetric difference distance δ of Section 2.11.2. This is certainly not true of BLEU score. However, the adjacencies measure of Section 4.6 can work with a slight modification of the grammar G_B^{NF}.

[4] Apologies for the double use of L.

[5] In practice, it may be necessary to replace $\mathbb{R} \times \mathbb{R}$ with ordered pairs from the log semiring of Section 3.5 to avoid underflow. In that case, the expectation semiring operators multiply using $+$ and add using $\log+$.

The gradient of the expected loss (4.34) with respect to a single matrix entry $B[\ell, r]$ is

$$\frac{\partial \mathrm{E}_{\pi \in \mathcal{N}(\pi^*)}\left[L(\pi^*, \pi); B\right]}{\partial B[\ell, r]} = \sum_{\pi \in \mathcal{N}(\pi^*)} L(\pi^*, \pi) \frac{\partial}{\partial B[\ell, r]} \frac{\exp(B(\pi))}{\sum\limits_{\pi' \in \mathcal{N}(\pi^*)} \exp(B(\pi'))}$$

Abstracting this for the moment, let $Z = \sum_f e^f$. Then,

$$\begin{aligned} \frac{d}{dx}\frac{e^f}{Z} &= \frac{f' e^f}{Z} - \frac{e^f \sum_g g' e^g}{Z^2} \\ &= \frac{e^f}{Z}\left(f' - \sum_g g' \frac{e^g}{Z}\right), \end{aligned}$$

leads to the following expression for the partial derivative of interest:

$$\sum_{\pi \in \mathcal{N}(\pi^*)} L(\pi^*, \pi) \Pr\left(\pi \mid \mathcal{N}(\pi^*); B\right) \big(I(\ell \prec r \in \pi) - \mathrm{E}_{\pi' \in \mathcal{N}(\pi^*)}\left[I(\ell \prec r \in \pi'); B\right]\big).$$

The expectation term inside parentheses is constant with respect to the variable π of the outermost sum, so this refactors to

$$\mathrm{E}_{\pi \in \mathcal{N}(\pi^*)}\left[L(\pi^*, \pi) I(\ell \prec r \in \pi)\right] - \mathrm{E}_{\pi \in \mathcal{N}(\pi^*)}\left[L(\pi^*, \pi)\right] \mathrm{E}_{\pi \in \mathcal{N}(\pi^*)}\left[I(\ell \prec r \in \pi)\right],$$

the difference between the expected loss when $\ell \prec r$ and the total expected loss times the probability that $\ell \prec r$.

Computation of these gradients runs the Inside-Outside algorithm (Baker, 1979)—see Section 3.9.2—using the expectation semiring to compute both the total (unnormalized) probability inside and outside each constituent, and the corresponding expected losses.

In every permutation tree, there is a single node that decides whether $\ell \prec r$. There are many such nodes in the parse forest, but only one per tree. Consider each such node N, and let it combine the spans (i, j) and (j, k). N either puts $\ell \prec r$ and updates $B[\ell, r]$ or puts $r \prec \ell$ and updates $B[r, \ell]$. Either way, the amount of the update is the same. First, let

$$w = \alpha(N) \otimes \gamma(N \to L\ R) \otimes \beta(L) \otimes \beta(R),$$

be the sum, in the semiring, of the weights of all trees in the forest that use the particular grammar rule $N \to L\ R$. Then the update is

$$\frac{v(w)}{p(\beta(S))} - \frac{v(\beta(S))}{p(\beta(S))} \frac{p(w)}{p(\beta(S))},$$

n	1000 Random	100,000 Best	1000 Best
5	0.991	0.985	0.985
10	0.990	0.929	0.857
15	0.992	0.800	0.787
20	0.990	0.763	0.796
25	0.988	0.774	0.828
30	0.993	0.806	0.799
35	0.995	0.790	0.817
40	0.994	0.805	0.752

Table 4.4: Kendall's τ rank correlation of adjacencies with BLEU score. Random permutations include all of Π_n, but 1000 and 100,000 Best include only those in $LG_B^{\mathrm{NF}}(\pi^*)$. Best is according to adjacency, rather than BLEU. The $n = 5$ case includes fewer permutations because $5! = 120$ and the neighborhood contains only 90 of those. See the text for discussion of BLEU score for single permutations.

where S is the start symbol of the grammar. The first term is the portion of the expected loss contributed by node N, and the second term is the product of the total expected loss ($\beta(S)$ is the semiring sum of all trees in the forest) and the probability of $N \to L\ R$.

This correctly computes the gradients even though many Δs sum to the gradient of each matrix entry. The left-hand term accumulates the total expectation as the sum of node expectations, and the right-hand term accumulates the total probability times a constant expectation as the sum of node probabilities times that constant.

4.11.4 Preserved Adjacencies

Table 4.4 shows the correlation between BLEU score[6] and the number of preserved adjacent pairs introduced in Section 4.6. The correlation is very high, though it is lower among the best permutations than among random ones. Due to this high correlation, it is not unreasonable to use the adjacency count as a substitute for or approximation to BLEU score during optimization of minimum expected loss.

Computing the number of preserved adjacent pairs has two special requirements.

1. Parsing must use the neighborhood of the true permutation π^* against which the loss function measures other permutations. Parsing cannot handle the neighborhood of any other permutation efficiently.

[6]BLEU score measured on single permutations is problematic, because especially higher-order n-grams often have zero precisions. Therefore these scores are measured as though the single permutation had been added to a corpus of 100 other permutations with baseline 1–4-gram precisions of 1.00, 0.57, 0.38, and 0.28. The sign test described in Section A.2 uses a similar procedure.

$$
\begin{array}{rcl}
S_{i-1,i} & \rightarrow & \pi_i,\ \forall i \in \binom{n}{1} \\
\\
S_{i,k} & \rightarrow & \overrightarrow{S}_{i,k} \ \text{ or } \ \overleftarrow{S}_{i,k} \\
\overrightarrow{S}_{i,k} & \rightarrow & \overrightarrow{S}^{+R}_{i,k} \ \text{ or } \ \overrightarrow{S}^{-R}_{i,k} \\
S^{-R}_{i,k} & \rightarrow & \overrightarrow{S}^{-R}_{i,k} \ \text{ or } \ \overleftarrow{S}_{i,k},\ \forall (i,k) \in \binom{n+1}{2}, k-i \geq 2 \\
\\
\triangleright\ \overrightarrow{S}^{+R}_{i,k} & \rightarrow & \overrightarrow{S}^{+R}_{i,k-1} \ \ S_{k-1,k} \\
\overrightarrow{S}^{+R}_{i,k} & \rightarrow & S^{-R}_{i,k-1}\ S_{k-1,k} \\
\overleftarrow{S}_{i,k} & \rightarrow & S_{k-1,k}\ S_{i,k-1},\ \forall (i,k) \in \binom{n+1}{2}, k-i \geq 2 \\
\\
\overrightarrow{S}^{-R}_{i,k} & \rightarrow & S_{i,j} \ \ \overleftarrow{S}_{j,k} \\
\overleftarrow{S}_{i,k} & \rightarrow & \overrightarrow{S}_{j,k} \ \ S_{i,j},\ \forall (i,j,k) \in \binom{n+1}{3}, j < k-1
\end{array}
$$

Figure 4.5: A normal-form grammar for block insertions accounting for adjacent pairs. $\overrightarrow{S}^{+R}_{i,k}$ generates subpermutations that end with π_k, while $\overrightarrow{S}^{-R}_{i,k}$ generates subpermutations that do not end with π_k. $\overleftarrow{S}_{i,k}$ likewise cannot end with π_k. Only the rule marked $\triangleright$ results in a preserved adjacent pair—its left child ends with π_{k-1} and its right child is π_k.

n	1000 Random	100,000 Best	1000 Best
5	0.154	0.162	0.162
10	0.037	0.089	0.269
15	0.053	0.195	0.317
20	0.077	0.294	0.433
25	0.032	0.293	0.440
30	0.021	0.335	0.390
35	0.020	0.223	0.424
40	0.043	0.344	0.161

Table 4.5: Kendall's τ rank correlation of $B^{\tau}_{\pi^*}(\pi)$ with BLEU score. See Table 4.4 for a description of columns. Best is according to $B^{\tau}_{\pi^*}(\pi)$, rather than BLEU. The correlations less than 0.05 in the Random column are *not* significant at 95%.

2. The grammar $G^{\mathrm{NF}}_{B}(\pi^*)$ from Figure 2.35 requires modifications, shown in Figure 4.5. Only one rule in the new grammar results in a preserved adjacent pair.

Section 4.12.1 experiments with learning parameters to optimize this criterion using stochastic gradient descent, as well.

4.11.5 Rank Correlation

Table 4.5 shows the correlation between BLEU score and the count $B^{\tau}_{\pi^*}(\pi)$, introduced in Section 2.11.2, of correctly ordered pairs in π. The correlation is very small, though positive, over random permutations, but improves somewhat over the best permutations according to $B^{\tau}_{\pi^*}$. This is the opposite pattern from the adjacency measure of Section 4.11.4.

This measure, related to rank correlation, is clearly not a good approximation to BLEU score for minimization of expected loss. However, it is an interesting loss function in its own right, particularly because of its similarity to the linear ordering problem. It is the correct loss function if the relative order of every pair of words is equally important.

4.12 Results

4.12.1 Learnability

This section reports the performance of several algorithms using the development set for *both* training and testing. This serves two purposes:

1. it validates learning algorithms, avoiding the expense of running on the much larger training set for algorithms that show poor potential, and

2. it validates feature sets, ensuring that it is possible to learn good orderings at all.

Both of these purposes are somewhat confounded by the problem of overfitting. In the first case, an algorithm that performs worse on its own training data may still perform better on unseen data, because of the issue of generalization. These experiments therefore do not attempt to decide the best learning algorithm. Rather, they attempt to distinguish between algorithms that are viable and those that are moribund.

In order to mitigate the effect of overfitting on the second purpose, the features used are still *selected* on the training set, so that training and testing on the development set use reasonable feature sets. Still, with a relatively small development set, many features may occur only a single time. Those will prove quite useful to the learning algorithms, though they will obviously overfit.

Figure 4.6 shows performance of two models each trained with averaged perceptron and averaged search perceptron. These results are a strong validation of both the feature sets and the learning algorithms. The models with POS-only features reached a monolingual BLEU score of 73.16 at iteration 80, while the POS+Word features achieved BLEU 82.53 with the search perceptron after 60 iterations.

Figure 4.7 shows a plot of log likelihood and BLEU score of models trained using stochastic gradient descent to maximize contrastive likelihood, with two different learning rates. Increasing the log likelihood of the target permutation given its neighborhood does *not* lead to improved decoding performance as measured by BLEU score. In fact, the curves show approximately opposite trends. This does not appear to be a viable learning criterion for this task.

Figure 4.8 shows a plot of expected adjacencies and BLEU score of models trained using SGD to maximize expected adjacencies, again with two different learning rates. Increasing the number of expected adjacencies *does* also increase decoding performance as measured by BLEU. However, it doesn't increase BLEU by very much, as compared to the perceptron methods. Strangely, the perceptron-trained models also have more expected adjacencies. The experiments in the following section use perceptrons exclusively.

4.12.2 Learning and Generalization

This section experiments with the entire Europarl training set of 747,088 sentence pairs, measuring performance on the held-out development set of 2,000 sentences. There are three variables:

1. the feature set, including the selection threshold,
2. the learning procedure, which has parameters of its own, and
3. the oracle ordering, the possibilities each described in Section 3.8.1.

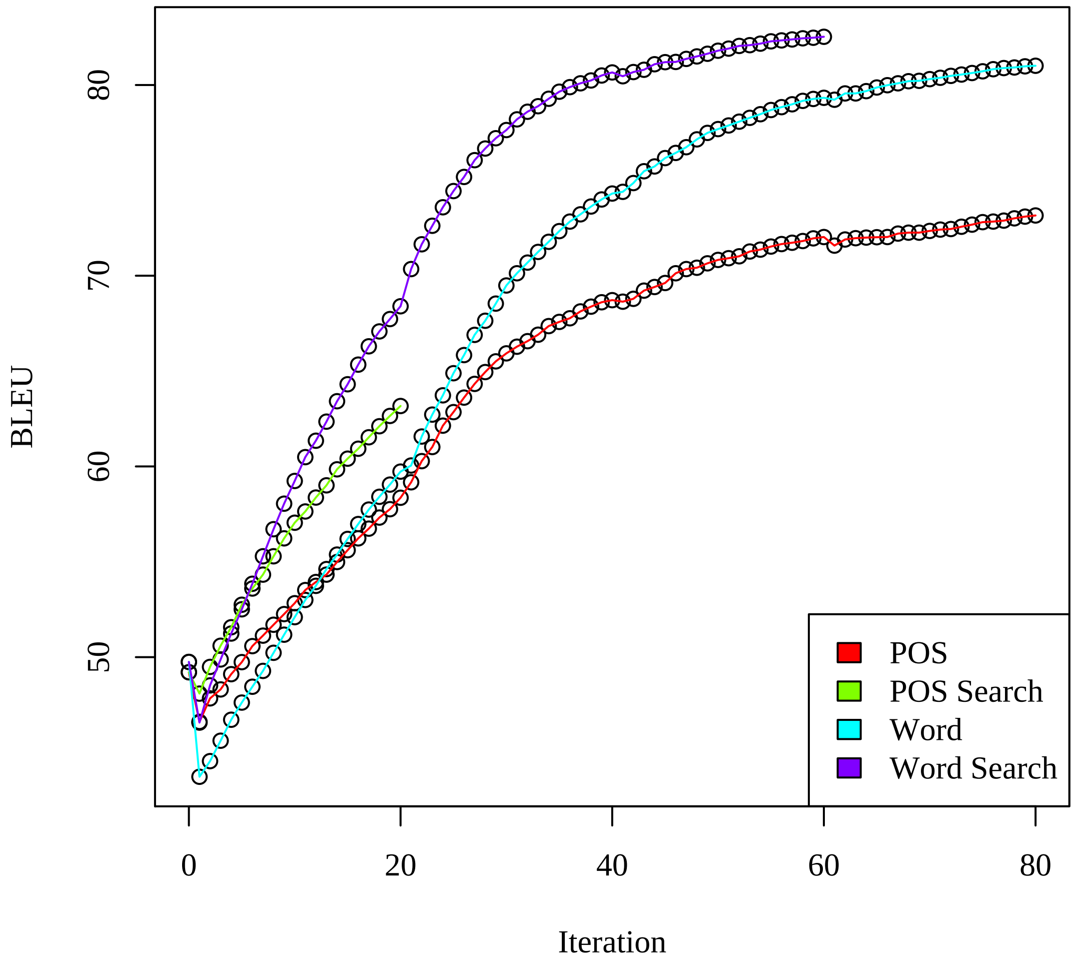

Figure 4.6: BLEU score for models trained with perceptrons. The models labeled "POS" use only part-of-speech features, while the models labeled "Word" also include word features. The models labeled "Search" use the guided search procedure of Section 4.8 along the *model* trajectory, while the others are the single-update perceptrons of Section 4.7. Search perceptrons learn faster as counted by iterations, but undergo more model updates per iteration.

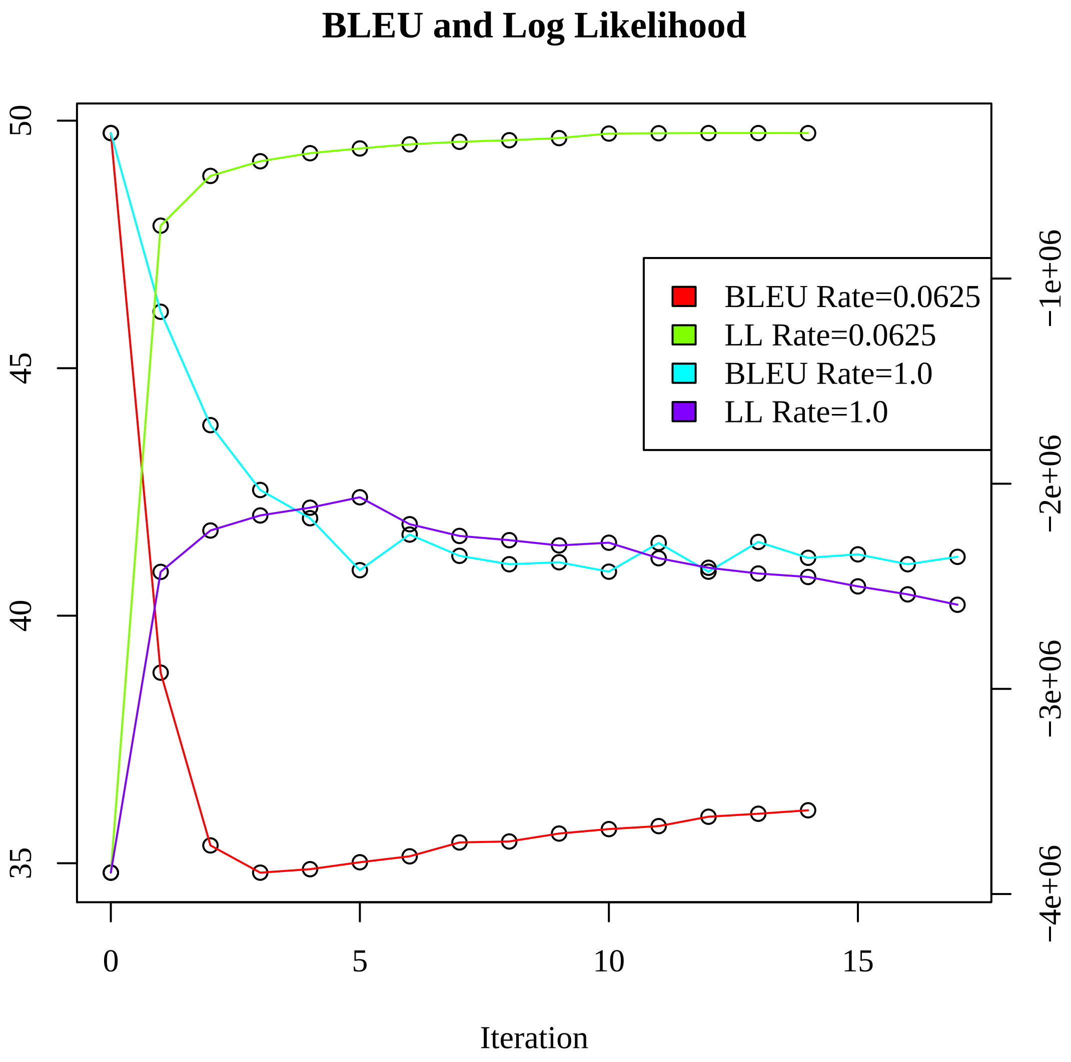

Figure 4.7: Likelihood vs. BLEU score for two different learning rates.

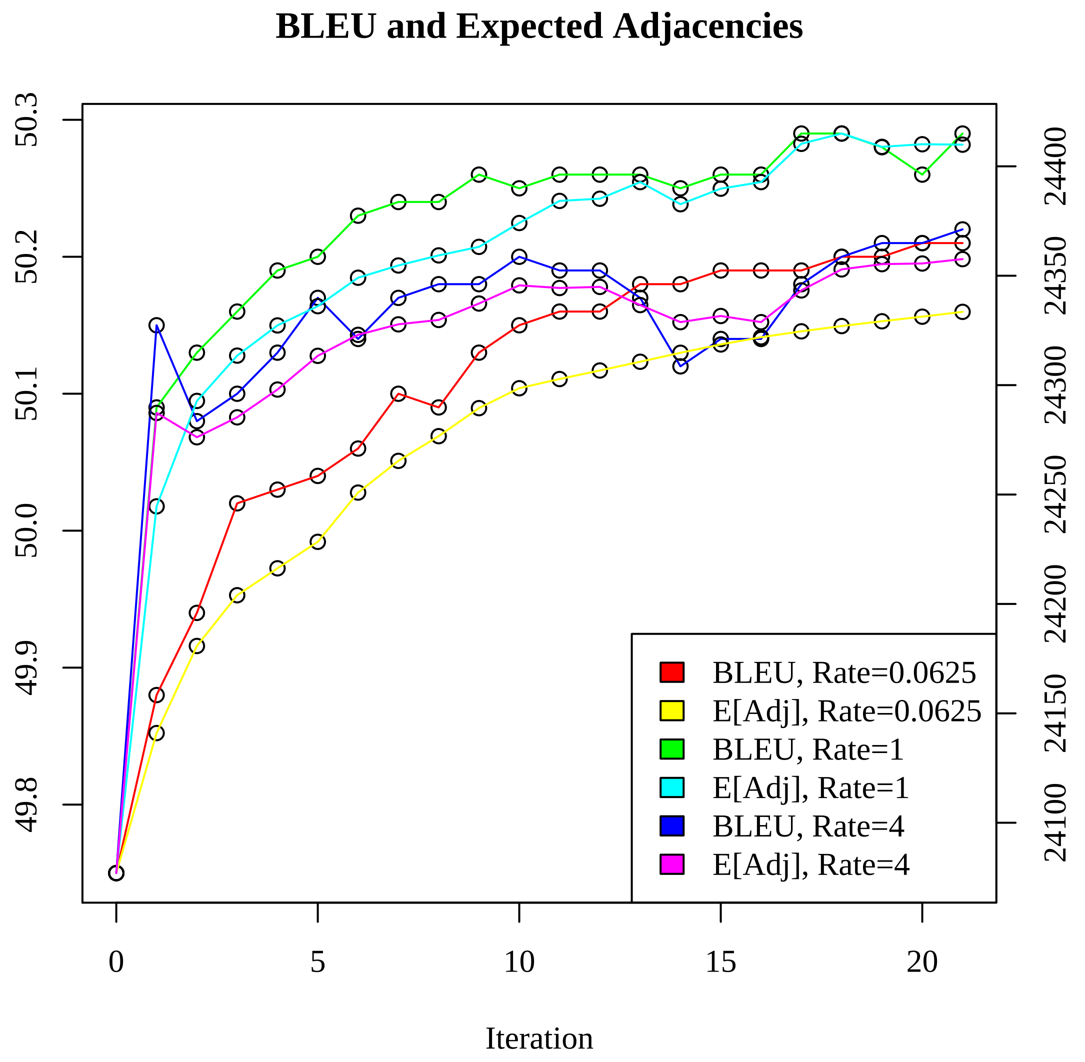

Figure 4.8: Expected adjacencies vs. BLEU score for three different learning rates. In each case, expected adjacencies in the target neighborhood is a good predictor of BLEU score. However, eventually, improving expected adjacencies fails to also improve BLEU. BLEU also improves more slowly with this learning method than with perceptrons.

Features	**Learning**	**Oracle**	**CPUs**	**BLEU 1**	**BLEU 2**	**BLEU 3**
German unreordered				44.53	49.65	41.38
POS:5	Baseline	1		44.15	48.62	40.96
POS:100	Baseline	1		44.36	49.21	41.14
POS:5	Search:0.0625	1	1	47.10	49.77	42.75
POS:5	Perceptron:0.0625	2	1	46.43	50.22	42.45
POS:5	Search:0.0625	2	4	46.22	50.21	42.42
POS:5	Search:0.0625	2	10	45.93	49.89	42.06
POS:5	Search:0.125	2	10	46.41	50.40	42.42
POS:5	Search:0.25	2	10	46.51	50.70	42.36
POS:100	Search:0.125	2	10	46.37	50.55	42.46
POS:5	Baseline	2		44.33	49.21	41.09
POS:5	Search:0.0625	2	10	46.01	50.05	42.12
POS+Word:100	Baseline	2		44.60	49.75	41.44
POS+Word:100	Perceptron:0.0625	2	1	46.92	51.51	43.31
POS+Word:100	Search:0.0625	2	10	46.76	51.30	43.13
+Dep:5	Search:0.0625	2	10	46.87	51.45	43.22
POS:5	Baseline	3		42.43	46.43	40.03
POS+Word:100	Baseline	3		43.32	47.68	40.56
POS+Word:100	Search:0.0625	3	10	45.44	48.08	43.39

Table 4.6: Performance of various models on held-out development data. All search perceptrons used the *model* trajectory. BLEU 1, 2, and 3 are measured against Oracles 1, 2, and 3, respectively. The two models reported with identical parameters have different *initialization*. The first started from the baseline model trained on Oracle 1, while the second started from the baseline model trained on Oracle 2—see Section 4.5 on page 137.

In addition, because iterative training over three quarters of a million sentences can require a great deal of time, many of the experiments split the training data into several smaller parts and trained multiple models simultaneously, combining them at the end by averaging. This is yet another variable with implications for performance.

The performance of several models at reordering the held-out development data is indicated in Table 4.6. There are a number of observations to make about this data:

- As expected, the two models trained on Oracles 1 and 3 have the best performance measured against their own oracles, but the worst performance on Oracle 2. Models trained on Oracle 2 have the best performance on the translation task of Section 3.8.

- There does not appear to be much difference between the quality of the models trained with search perceptrons and the one trained with standard perceptron.

This may be in part because of the restrictions made during decoding, which the next section discusses.

- Parallelization appears to be harmful. The benefit of parallelization is much faster training, but the cost is a small reduction in generalization performance.

- More features help. The model that includes word features scores 51.30 on BLEU 2, significantly higher than the best model with POS features—50.70—according to the paired permutation test. (This difference is not significant according to the sign test.) The model that includes dependency features further improves this to 51.45, though this is too small to be significant—it is only at the 79th percentile on the paired permutation test. Unfortunately, neither of these models result in better performance on the full translation task—perceptron models have the highest performance there. See Table 3.3 on page 109 for details.

4.12.3 Search Error

Traditional reordering models (cf. Och and Ney (2004)) *rely* on search error (in the form of a limited window) for performance (Al-Onaizan and Papineni, 2006). That is, the translation candidate that is best according to the evaluation metric (namely, BLEU score) is often different from the one that has the lowest cost according to the model. The translation systems restrict the distance that words can move away from their position in the source sentence, introducing “errors” according to the model, but producing better translations according to BLEU.

The models learned in this chapter *also* suffer from a reliance on search error—not in the same way as traditional models, however. Limiting search to the neighborhood of the identity permutation (exactly the set of rearrangements allowed by ITG) improves performance relative to iterated local search—which potentially considers any of the $n!$ permutations—for window widths larger than eight. This constitutes systematic introduction of search error. However, limiting LOP reordering models to rearranging words within a maximum window size, given the neighborhood constraint, always *degrades* performance relative to the unlimited system (see Figure 4.9).[7] The model error is not such that long-distance movement needs to be prohibited completely.

This reliance on the ITG neighborhood for performance is mysterious, particularly considering the result, illustrated in Figure 2.50 on page 80, that fewer than half of the “gold standard” reorderings implied by the alignments are in the neighborhood of the initial German sentences. Understanding this limitation, and perhaps correcting it, is an elusive goal.

[7]For this reason, all of the results reported in this dissertation decode under the LOP models without leaving the neighborhood of the identity permutation. Given the neighborhood constraint, the reordering is exact.

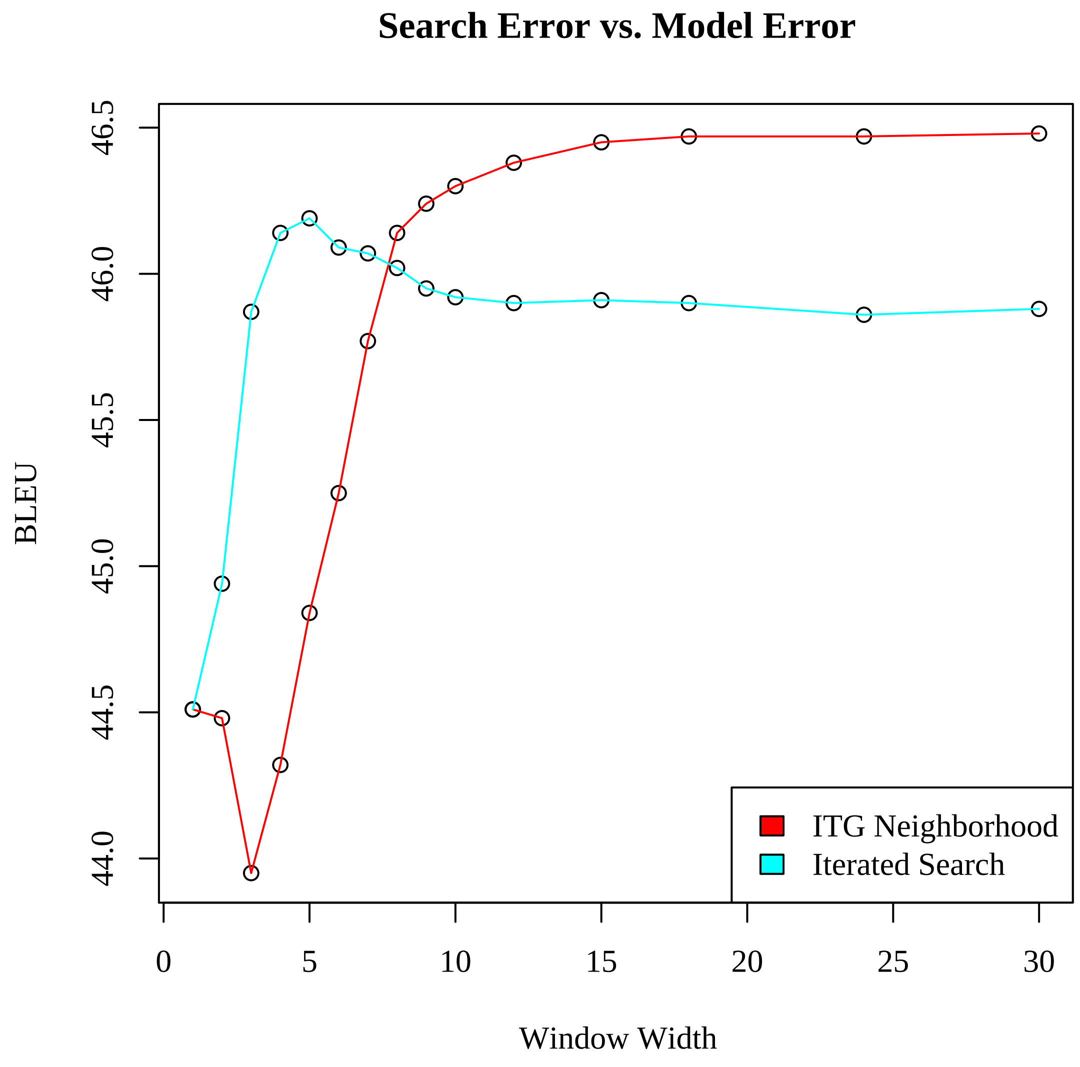

Figure 4.9: Search error vs. model error for one reordering model. Monolingual BLEU (see Section 4.6) measured on the held-out Dev set with various restrictions on the maximum width of a swap node in the "parse tree" during reordering. Width 1 corresponds to the baseline system—no reordering—, while width 30 is indistinguishable from the unrestricted system to four decimal places.

While the constraint is helpful during decoding, early experiments limiting search to the neighborhood of the identity permutation during *training* did not lead to improvements over unconstrained hillclimbining.

4.13 Other Work

Boyan and Moore (2000) describe an algorithm they call Stage. They let V^π be a function that gives the value of the objective function achievable if local search begins at a given configuration. They use machine learning to estimate $\tilde{V}^\pi$, a linear or quadratic function of features of configurations.

It might be possible to learn C costs or adjacency costs (which would turn LOP into TSP) using Stage. Then search using the augmented evaluation function would lead to a good restart for LOP-only search.

4.14 Summary

This chapter makes the first application of full-scale machine learning for constructing Linear Ordering Problems from data. The prior work of Cohen et al. (1999) that Section 4.2 describes is more akin to classifier combination. It requires that several candidate orderings of the items already exist.

The possibility of learning the ordering function, which this chapter introduces, may prove more generally useful than its application here to machine translation. Ranking and ordering problems are quite common, and they don't always come accompanied by an obvious scoring function. The information retrieval application is one example, but there are many others. Ranking competitors based on their head-to-head performance, such as in college football, might be another.

In addition to this major contribution, this chapter adapts a number of previous machine learning methods to the problem at hand. It also introduces new variations on these algorithms, including the search-based perceptron, contrastive likelihood, and maximization of preserved adjacencies. This last requires its own new variation on the grammars of Chapter 2. The adaptation of loss minimization to the very large-scale neighborhood requires adaptation of existing algorithms for computing the expected loss, and makes use of the expectation semiring.

Finally, the chapter uses training over the development set to demonstrate that it is possible to learn Linear Ordering Problems from data with perceptrons, and uses a standard training/development arrangement to further demonstrate that this learning generalizes to unseen data, producing significant improvements in German sentence reordering, as measured by monolingual BLEU.

Chapter 5

Conclusion

This dissertation has explored three hard problems in three chapters—the linear ordering problem, machine translation, and parameter estimation. Each chapter contributes some novel ideas or techniques.

One primary goal of the dissertation as a whole is to explore the ways in which these problems are related, and to bring them closer together, in some sense. Chapter 2 applies ideas from natural language processing, particularly grammars and parsing, to the linear ordering problem. Chapter 3 introduces the linear ordering problem as a new model for rearranging words. Chapter 4 introduces parameter estimation to the general linear ordering problem. Each chapter applies search in one form or another to a computationally intractable problem.

The novel contributions of the dissertation are numerous. Chapter 2 includes:

- a dynamic program for computing the costs of block insertions in constant time per neighbor,
- a short-cut local search procedure using block insertions, with state-of-the art performance on the XLOLIB benchmarks,
- the first application of very-large scale neighborhoods to search for the linear ordering problem,
- normal forms and neighborhood sizes for $G_{B\leq w}$ and special cases, and
- neighborhood graph diameter using analogies to sorting.

Chapter 3 introduces:

- formulation of IBM Model 4 as a cascade of finite-state transducers,
- the ABC model of reordering for translation, itself a novel combinatorial optimization problem,

- a generalization of the linear ordering problem to triples, rather than pairs,
- preprocessing for translation using the linear ordering problem, and
- several prospective decoding methods for use with the ABC model.

Finally, Chapter 4's contributions include:

- the first full-scale application of machine learning for constructing linear ordering problems,
- an adaptation of max-margin parsing to the linear ordering problem,
- the search perceptron,
- contrastive likelihood using a very large-scale neighborhood as the contrast set,
- an algorithm for computing expected loss over all permutations in the very large-scale neighborhoods, and
- demonstration that linear ordering problems can be learned for sentence ordering.

Although the experiments of this dissertation apply the search and learning algorithms only to the linear ordering problem, they are more widely applicable. Search methods based on grammars apply to the general ABC model, which subsumes the traveling salesman problem. The constant time per neighbor property applies to the BlockInsert_n neighborhood for the three-dimensional C model, if it satisfies the cyclicity constraint, and the VLSN BlockInsert_n^* can also be searched in $\Theta(n^3)$ time for constrained C. There are no known natural applications of this model, but it could certainly be applied to machine translation, at least. It is no more outlandish than the linear ordering problem. Many of the learning algorithms applied here to the LOP should also transfer to the ABC model.

5.1 Future Work

The work of this dissertation constitutes only a small contribution to each of the fields that it touches upon—namely combinatorial optimization, natural language processing, and machine learning. However extensive the work, it cannot even aspire to exhaustivity. More important, though, the connections it makes between these fields open up numerous opportunities for further research. The space of ideas that remain for exploration is vast, but there are some clear directions for future work:

- The linear ordering problem has already seen wide application across a variety of disciplines, but ordering is ubiquitous and the LOP will no doubt find other applications. Advanced techniques for search in the LOP seem to be applied almost exclusively to the economics application. There is no reason why other applications should not make use of the best tools available for solving this problem.

- The poor performance of exact decoding algorithms for the ABC model of machine translation is disappointing. Finding an appropriate approximate algorithm to use for integrating the linear ordering problem into the decoder is an important future step. The model has proven useful for preprocessing, but has not yet been tested in conjunction with the target language model.

- Features often make or break a model. The features applied to learning for the linear ordering problem are derived directly from the task of dependency parsing which, while somewhat analogous, is certainly not an exact match. Designing features especially for the LOP may lead to substantial improvements in learning for this task.

In addition to these major directions, there are possibilities that the dissertation already suggests, either explicitly or implicitly. Section 2.7.3 proposed evaluating Block $\mathcal{LS}_f$ in the context of memetic algorithms, as well as learning which neighborhood to apply when. Section 2.7.4 proposed simulating annealing as another way of avoiding the local maximum problem.

Moving on to machine translation, Section 3.3.3 mentioned both segment choice models and the BTG constituent reordering model, which reorder phrases rather than single words. The linear ordering problem model could be applied in this way as well, with features computed over phrases instead of words. A single phrase segmentation, or possibly a lattice of them, could serve as the input.

An alternative to this fixed phrase segmentation is the addition of phrase brackets symbols to the input sequence. Section 3.6.1 discussed a few possibilities for their use.

An especially attractive possibility is to adapt the grammar from Figure 4.5 in Section 4.11.4, accounting for preservation of adjacent items in the original permutation. This could make a very lightweight alternative to the A model—it doesn't change the attractive runtime of BlockInsert$_n^*$ for the LOP at all. This would mean introduction of additional features and parameters to the model. It would also require constraint of search to the neighborhood of the identity permutation. Fortunately, that constraint already leads to better translation performance.

Finally, the untested learning algorithms from Chapter 4, namely the several instances of MIRA in Sections 4.9 and 4.10, and sampling for computing the true partition function in Section 4.11.1, would make interesting further work. The MIRA methods are close enough to the perceptrons that they should have good performance, and the fact that they take more information into account at each update might lead to improvements. The attempted likelihood-based methods all led to disappointing results, but the Metropolis-Hastings algorithms has the important virtue that it attempts to compute expectations under the true probability distribution, rather than crude approximations.

For Future Reference

The latest version of this dissertation, including the inevitable errata, can be found at `http://nlp.cs.jhu.edu/~royt/`. This will also serve as the home for software implementing the algorithms described herein.

Appendix A

MT Evaluation and Significance Tests

A.1 The BLEU Score

The BLEU score, introduced by Papineni et al. (2002), is an automatic evaluation measure for machine translation. It is compatible with one or more reference translations. It is basically a geometric mean of n-gram precisions, but has an additional term to control recall.

Definition A.1 *Let c be the total candidate word length, r the effective reference word length, N a maximum n-gram length, w_n an arbitrary weight, and p_n the total modified n-gram precision of the candidate output. The* **BLEU score** *of the candidate output is*

$$\log \textsc{Bleu} = \min(1 - \frac{r}{c},\ 0) + \sum_{n=1}^{N} w_n \log p_n. \tag{A.1}$$

This definition requires some explanation.

- The "effective reference length" r is the sum of best-match reference lengths for each sentence. In this dissertation, there is only one reference translation or ordering, so r is trivially the total length of the references.
- $N = 4$ is standard.
- The weights w_n must sum to one. $\frac{1}{N}$ is standard.
- The modified precision p_n is the fraction of n-grams in the candidate that occur in at least one of the reference translations, with the following limitation: if an n-gram occurs more than once in a candidate sentence, *clipping* limits the number that count as correct to the maximum number that occur in a single reference sentence.

This dissertation uses the standard $N = 4$ and $w_n = \frac{1}{4}$.

Clipping is not an issue for permutations, because all of the items are unique. However, except for the correlations in Sections 4.11.4 and 4.11.5, BLEU scores of reorderings are measured with respect to words rather than word positions. This means that when a word appears more than once in a sentence, the "wrong" one may be used in an n-gram and still count as correct.

Machine translation evaluation is an important and difficult task in its own right. Some of the methods of this dissertation might even be adapted to this task. The BLEU score is still widely used, in spite of many criticisms.

A.2 Significance Tests

This dissertation reports many results using BLEU score. An important question for these results is whether observed differences are statistically significant, or whether they are likely to have happened by chance. The *null hypothesis* H_0 is always that two translation systems are equally good. The purpose of significance testing is to reject the null hypothesis when sufficient evidence is available.

BLEU score is a little problematic for significance testing because it is an aggregate measure. As Section 4.11.4 mentions, BLEU score on a single sentence is frequently meaningless because high-order n-grams have low precision that is often zero. As a result, there is frequently only a single sample of the BLEU score of two competing systems—that of their respective outputs on some test corpus. Statistical significance testing from a single sample is hopeless in the absence of very strong distributional assumptions.

A.2.1 Bootstrap Resampling

Several authors (Germann, 2003; Koehn, 2004; Zhang and Vogel, 2004), have proposed bootstrap resampling (Efron and Tibshirani, 1993) as a solution to this problem. (Koehn, 2004) described the procedure in detail:

1. Translate a corpus of N sentences with both system 1 and system 2.
2. S times, sample N sentences, *with replacement*, from the corpus.
3. Measure the BLEU scores of systems 1 and 2 on each of the size-N sample corpora.
4. The confidence that system 2 is better than system 1 is the fraction of the S samples on which it has higher BLEU score.

What is the alternative hypothesis H_a that this procedure tests? Each boostrap sample tests *whether* system 2 scores better than system 1, but disregards the *amount* of the difference.

A.2.2 Sign Test

Collins et al. (2005) proposed a clever sign test construction as an alternative to the paired bootstrap resampling test just described. This new test has the alternative hypothesis H_a that the probability that system 2 translates a given sentence better than system 1 is greater than $\frac{1}{2}$.

1. Translate a corpus of N sentences with both system 1 and system 2.
2. For each $i \in \{1, 2, \ldots, N\}$, replace system 1's translation of sentence i with system 2's translation, and measure the resulting BLEU score.
3. Record the sign of the difference between the new BLEU score, and system 1's BLEU score on the corpus.
4. Perform a standard sign test on the results of the previous step.

The problem with this procedure is that H_a is not exactly the right criterion. It ignores the amount of the difference between the BLEU scores of the two systems, which may even be negative. It is possible for system 2 to satisfy H_a but nonetheless have worse expected corpus BLEU score than system 1. This would be the case, for example, if system 2 made small improvements over system 1 on a significant majority of sentences, but made even larger degradations on the rest.

The next section argues for a paired *permutation* test, which has a more attractive alternative hypothesis.

A.2.3 Paired Permutation Test

A paired permutation test is a bootstrapping procedure similar to the paired resampling described above, but uses all of the available data for each sample. Rather than sample pairs with replacement, permute pairs between the systems according to the flip of a coin. The resulting sample has, in expectation, half of its outputs from the first translation system, and half from the second. Computing the difference between the BLEU scores that result for many such samples leads to an estimate of the percentile of the true system BLEU difference with respect to the distribution of the samples. This procedure is not new to this dissertation—Smith and Eisner (2006) mentioned using a paired-sample permutation test without offering a description, for example.

1. Translate a corpus of N sentences with both system 1 and system 2.
2. S times, for each $i \in \{1, 2, \ldots, N\}$, swap the translations of sentence i with probability $\frac{1}{2}$.

3. Measure the difference betweeen the BLEU scores of the two sets of N translations.

4. The confidence that system 1 improves on system 2 is the fraction of samples for which the true difference is larger than the absolute value of the sample difference.

What is the alternative hypothesis H_a of this procedure? It is that the *amount* of the difference between the true scores of systems 1 and 2 is greater than zero. If H_0 is true, then the outputs of the systems should be interchangeable. The test interchanges outputs at random and measures the results. If the observed difference is greater than the differences that occur during the random interchanges, that suggests that the true difference is not zero.

The results reported in this dissertation use both the sign test and the paired permutation test. The two tests usually agree, though confidences often differ by several orders of magnitude. The paired permutation test is usually more confident.

A.3 METEOR

METEOR (Lavie et al., 2004; Banerjee and Lavie, 2005; Lavie and Agarwal, 2007) is a recall-focused translation evaluation measure that uses stemming and synonymy in addition to exact matching. It has been shown to significantly outperform BLEU in terms of correlation with human judgments. Stemming uses Porter's stemmer. Synonymy and alternate stemmings come from WordNet.

For a given sentence, METEOR first computes an alignment between the candidate and the reference translation. This alignment is one-to-one, with some words on both sides left unmatched. Given the alignment, METEOR computes precision P and recall R in the standard way—precision is the ratio of matched words to the length of the candidate, and recall the ratio of matched words to the length of the reference. The basis for the score is then

$$\mathit{Fmean} \overset{\text{def}}{=} \frac{1}{\frac{\alpha}{R} + \frac{1-\alpha}{P}} = \frac{PR}{\alpha P + (1-\alpha)R}, \tag{A.2}$$

a weighted harmonic mean. The remainder of the score is a chunk penalty, which rewards monotone alignments. A *chunk* is a sequence of word alignments that is consecutive in both the candidate and the reference—a higher-order n-gram match. Let c be the number of such chunks and a the total number of word alignments. Then the penalty is

$$\mathit{Penalty} \overset{\text{def}}{=} \gamma \left(\frac{c}{a}\right)^{\beta}. \tag{A.3}$$

Finally,

$$\textsc{Meteor} \overset{\text{def}}{=} \mathit{Fmean} \cdot (1 - \mathit{Penalty}). \tag{A.4}$$

In Banerjee and Lavie (2005), the parameters were heuristically set to $(\alpha, \beta, \gamma) = (0.9, 3, 0.5)$. The METEOR scores reported in this dissertation use $(0.80, 0.83, 0.28)$, the default values of the scorer for translation into English, tuned by Lavie and Agarwal (2007) to maximize the sum of adequacy and fluency. Results also use all four matching criteria: exact, Porter stemmer, WordNet stemmer, and WordNet synonymy.

A.4 TER

The Translation Edit Rate (TER) of Snover et al. (2006) is another automatic evaluation measure, intended to capture the amount of editing necessary to postprocess a candidate translation in order to arrive at the reference translation. Possible edits come in four types, each with identical cost:

- Insertion
- Deletion
- Substitution
- Shift

The first three are standard edit distance operations. *Shift* means moving a subsequence of the candidate to a new location—a block insertion as used in Chapter 2.

Let E be the smallest set of edits needed to produce one of the reference translations, and let $\bar{r}$ be the average length of the reference translations. Then the TER is simply

$$\text{TER} \overset{\text{def}}{=} \frac{|E|}{\bar{r}}. \tag{A.5}$$

TER, unlike BLEU and METEOR, is an error rate, meaning a lower score corresponds to a better translation.

Snover et al. note that the problem of finding optimal edit distance with movement is NP-complete, citing Shapira and Storer (2002). They therefore rely on greedy search to find good move operations, interspersed with computation of traditional edit distance, using only insertion, deletion, and substitution.

This problem itself is a potential application of the methods of this dissertation. This would make use of the A model of Chapter 3 and a modification of the grammars from Chapter 2.

- Implement edit distance as a finite-state transducer with cost 1 for every edit. Compose the transducer with the reference translation(s) and project to the input to get the language of possible candidates.
- Replace the grammar weights with $\vec{\gamma}_{i,j,k} = 0$ and $\overleftarrow{\gamma}_{i,j,k} = 1$.

Because both METEOR and TER's statistics can be broken down sentence by sentence, it is possible to use the same sign and permutation tests for these measures as for BLEU score.

Appendix B

The TIGER Treebank

This dissertation uses annotations from the TIGER treebank (Brants et al., 2002) for both part-of-speech and dependency features, as described in Section 4.4.

This chapter describes the annotations used in the TIGER treebank. Section B.1 describes the part-of-speech tag set, and Section B.2 describes dependency labels.

B.1 Part of Speech Tags

TIGER uses the Stuttgart-Tübingen tagset, described in Schiller, Teufel, and Thielen (1995), for parts of speech. Table B.1 shows the tag set. The descriptions there are taken directly from the English version of a table provided by the authors.[1] TreeTagger uses a slightly different version of the tags, which replaces PAV with PROAV and $(with $*LRB*.

B.2 Dependency Labels

Amit Dubey converted the phrase-structure trees of the TIGER treebank to dependency structures for the CoNLL-X shared task (Buchholz and Marsi, 2006). The labels are the grammatical functions used in the NEGRA corpus (Skut, Krenn, Brants, and Uszkoreit, 1997). Table B.2 shows the label set. The descriptions there come directly from the NEGRA website.[2] In addition, PUNC is used for all punctuation.

[1] `http://www.cl.cam.ac.uk/~sht25/papers/stts.pdf`

[2] `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/kanten.html`

ADJA	attributive adjective	PRF	reflexive personal pronoun
ADJD	adverbially or predicatively used adjective	PWS	substituting interrogative pronoun
ADV	adverb	PWAT	attributive interrogative pronoun
APPR	preposition or left part of circumposition	PWAV	adverbial interrogative or relative pronoun
APPRART	preposition with article	PAV	pronominal adverb
APPO	postposition	PTKZU	*zu* before an infinitive
APZR	right part of circumposition	PTKNEG	negation particle
ART	definite or indefinite article	PTKVZ	separated verbal particle
CARD	cardinal number	PTKANT	answer particle
FM	material of a foreign language	PTKA	particle with adjective or adverb
ITJ	interjection	TRUNC	first (separated) part of composition
KOUI	subordinating conjunction with *zu* and infinitive	VVFIN	finite content verb
KOUS	subordinating conjunction with a sentence	VVIMP	imperative content verb
KON	coordinating conjunction	VVINF	infinitive content verb
KOKOM	comparative conjunction	VVIZU	infinitive of content verb with *zu*
NN	common noun	VVPP	past participle of content verb
NE	proper noun	VAFIN	finite verb, primary auxiliary
PDS	substituting demonstrative pronoun	VAIMP	imperative, primary auxiliary
PDAT	attributive demonstrative pronoun	VAINF	infinitive, primary auxiliary
PIS	substituting indefinite pronoun	VAPP	past participle, primary auxiliary
PIAT	attributive indefinite pronoun without determiner	VMFIN	finite verb, modal auxiliary
PIDAT	attributive indefinite pronoun with a determiner	VMINF	infinitive, modal auxiliary
PPER	irreflexive personal pronoun	VMPP	past participle, modal auxiliary
PPOSS	substituting possessive pronoun	XY	non-word, containing special characters
PPOSAT	attributive possessive pronoun	$,	comma
PRELS	substituting relative pronoun	$.	punctuation at end of sentence
PRELAT	attributive relative pronoun	$(	other punctuation, sentence internal

Table B.1: The Stuttgart-Tübingen Tag Set (STTS).

AC	adpositional case marker	MW	way (directional modifier)
ADC	adjective component	NG	negation
AMS	measure argument of adj	NK	noun kernel modifier
APP	apposition	NMC	numerical component
AVC	adverbial phrase component	OA	accusative object
CC	comparative complement	OA2	second accusative object
CD	coordinating conjunction	OC	clausal object
CJ	conjunct	OG	genitive object
CM	comparative conjunction	PD	predicate
CP	complementizer	PG	pseudo-genitive
DA	dative	PH	placeholder
DH	discourse-level head	PM	morphological particle
DM	discourse marker	PNC	proper noun component
GL	prenominal genitive	RC	relative clause
GR	postnominal genitive	RE	repeated element
HD	head	RS	reported speech
JU	junctor	SB	subject
MC	comitative	SBP	passivized subject (PP)
MI	instrumental	SP	subject or predicate
ML	locative	SVP	separable verb prefix
MNR	postnominal modifier	UC	(idiosyncratic) unit component
MO	modifier	VO	vocative
MR	rhetorical modifier		

Table B.2: The NEGRA grammatical function labels.

Bibliography

Ravindra K. Ahuja, James B. Orlin, and Dushyant Sharma. Very large-scale neighborhood search. *International Transactions in Operational Research*, 7:301–317, 2000.

Yaser Al-Onaizan and Kishore Papineni. Distortion models for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 529–536, Sydney, Australia, July 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P06/P06-1067.

D. Arditti. Un nouvel algorithme de recherche d'un ordre induit par des comparaisons par paires. In E. L. Diday, editor, *Data Analysis and Informatics, III*, pages 323–343. North-Holland, Amsterdam, June 1984.

Abhishek Arun and Philipp Koehn. Online learning methods for discriminative training of phrase based statistical machine translation. In *Proceedings of Machine Translation Summit XI*, pages 15–20, Copenhagen, September 2007.

Franz Aurenhammer. On-line sorting of twisted sequences in linear time. *BIT Numerical Mathematics*, 28(2):194–204, June 1988.

James K. Baker. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W05/W05-0909.

Regina Barzilay, Noemie Elhadad, and Kathleen R. McKeown. Inferring strategies for sentence ordering in multidocument news summarization. *Journal of Artificial Intelligence Research*, 17:35–55, 2002.

O. Becker. Das helmstädtersche Reihenfolgeproblem—die Effizienz verschiedener Näherungsverfahren. *Computer Uses in the Social Sciences*, January 1967.

Adam L. Berger, Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Andrew S. Kehler, and Robert L. Mercer. Language translation apparatus and method of using context-based translation models. United States Patent, Patent Number 5510981, April 1996.

Konrad Boenchendorf. Reigenfolgenprobleme/mean-flow-time sequencing. *Mathematical Systems in Economics*, 74, 1982.

Agustin Bompadre and James B. Orlin. 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, 2005.

Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004. URL http://leon.bottou.org/papers/bottou-mlss-2004.

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

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 24–41, 2002.

Leo Breiman, Jerome H. Friedman, Richard H. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Pacific Grove, CA, 1984.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, June 1990.

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

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W06/W06-2920.

Pi-Chuan Chang and Kristina Toutanova. A discriminative syntactic word order model for machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 9–16, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P07/P07-1002`.

Irène Charon and Olivier Hudry. A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154 (15):2097–2116, October 2006.

Irène Charon and Olivier Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR: A Quarterly Journal of Operations Research*, 5 (1):5–60, 2007.

Hollis B. Chenery and Tsunehiko Watanabe. International comparisons of the structure of production. *Econometrica*, 26(4):487–521, 1958.

William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8, Philadelphia, July 2002. Association for Computational Linguistics.

Michael Collins, Philipp Koehn, and Ivona Kučerová. Clause restructuring for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 531–540, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P05/P05-1066`.

marquis de Condorcet, Marie Jean Antoine Nicolas de Caritat. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie Royale, Paris, 1785.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

Marta R. Costa-jussà and José A. R. Fonollosa. Statistical machine reordering. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 70–76, Sydney, Australia, July 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W06/W06-1609`.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, January 2003.

Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005. URL `http://pub.hal3.name/#daume05laso`.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 2007. Submitted.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April 1997.

John DeNero and Dan Klein. Tailoring word alignments to syntactic machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 17–24, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P07/P07-1003`.

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

Martin Durrell. *Hammer's German Grammar and Usage*. NTC Publishing Group, third edition, 1997.

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

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall, New York, 1993.

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 00–00, University of Pennsylvania, 2002. URL `http://www.aclweb.org/anthology/P02-1001.pdf`.

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

Jason Eisner. Expectation semirings: Flexible EM for finite-state transducers. In Gertjan van Noord, editor, *Proceedings of the ESSLLI Workshop on Finite-State Methods in Natural Language Processing*, 2001. URL `http://cs.jhu.edu/~jason/papers/#fsmnlp01`. Extended abstract (5 pages).

Jason Eisner. Simpler and more general minimization for weighted finite-state automata. In Marti Hearst and Mari Ostendorf, editors, *HLT-NAACL 2003: Main Proceedings*, pages 64–71, Edmonton, Alberta, Canada, May 27 - June 1 2003. Association for Computational Linguistics.

Jason Eisner and John Blatz. Program transformations for optimization of parsing algorithms and other weighted logic programs. In Shuly Wintner, editor, *Proceedings of FG 2006: The 11th Conference on Formal Grammar*, pages 45–85. CSLI Publications, 2007. URL http://cs.jhu.edu/~jason/papers/#fg06.

Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–464, College Park, Maryland, USA, June 1999. Association for Computational Linguistics. doi: 10.3115/1034678.1034748. URL http://www.aclweb.org/anthology/P99-1059.

Jason Eisner and Roy W. Tromble. Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Workshop on computationally hard problems and joint inference in speech and language processing*, New York, June 2006.

Pedro Felzenszwalb and David McAllester. The generalized A* architecture. *Journal of Artificial Intelligence Research*, 29:153–190, 2007.

Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Yoav Freund and Robert E. Schapire. 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, 1998. ACM Press. ISBN 1-58113-057-0. URL http://doi.acm.org/10.1145/279943.279985.

Carlos G. Garcia, Dionisio Pérez-Brito, Vicente Campos, and Rafael Martí. Variable neighborhood search for the linear ordering problem. *Computers & Operations Research*, 33:3549–3565, 2006.

Ulrich Germann. Greedy decoding for statistical machine translation in almost linear time. In Marti Hearst and Mari Ostendorf, editors, *HLT-NAACL 2003: Main Proceedings*, pages 72–79, Edmonton, Alberta, Canada, May 27 - June 1 2003. Association for Computational Linguistics.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 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, 2001. Association for Computational Linguistics.

Fred Glover, T. Klastorin, and D. Klingman. Optimal weighted ancestry relationships. *Management Science*, 20(8):1190–1193, April 1974.

Vaibhava Goel and William Byrne. Minimum Bayes-risk automatic speech recognition. *Computer Speech and Language*, 14(2):115–135, April 2000.

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

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W05/W05-1506.

Liang Huang and David Chiang. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P07/P07-1019.

Liang Huang, Hao Zhang, and Daniel Gildea. Machine translation as lexicalized parsing with hooks. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 65–73, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W05/W05-1507.

David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*. Wiley and Sons, 1997.

Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1(1):1–25, 1997.

Stephan Kanthak and Hermann Ney. FSA: An efficient and flexible C++ toolkit for finite state automata using on-demand computation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 510–517, Barcelona, Spain, July 2004.

Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

John G. Kemeny. Mathematics without numbers. *Daedalus*, 88:575–591, 1959.

Maurice Kendall. A new measure of rank correlation. *Biometrika*, 30:81–89, 1938.

Dan Klein and Christopher D. Manning. 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 2003. Association for Computational Linguistics.

Kevin Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615, December 1999. ISSN 0891-2017.

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

Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 2nd edition, 1973a.

Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 2nd edition, 1973b.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 388–395, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Summit X*, pages 79–86, Phuket, Thailand, September 2005.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 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 2003. Association for Computational Linguistics.

Philipp Koehn, Amittai Axelrod, Alexandra Birch Mayne, Chris Callison-Burch, Miles Osborne, and David Talbot. Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *Proceedings of the International Workshop on Spoken Language Translation*, Pittsburgh, October 2005.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P07/P07-2045.

Bernhard Körte and Walter Oberhofer. Triangularizing input-output matrices and the structure of production. *European Economic Review*, 2(4):493–522, 1971.

Roland Kuhn, Denis Yuen, Michel Simard, Patrick Paul, George Foster, Eric Joanis, and Howard Johnson. Segment choice models: Feature-rich models for global distortion in statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 25–32, New York City, USA, June 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N06/N06-1004.

Shankar Kumar and William Byrne. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In Marti Hearst and Mari Ostendorf, editors, *HLT-NAACL 2003: Main Proceedings*, pages 142–149, Edmonton, Alberta, Canada, May 27 - June 1 2003. Association for Computational Linguistics.

Shankar Kumar and William Byrne. Minimum Bayes-risk decoding for statistical machine translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 169–176, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

Shankar Kumar and William Byrne. 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 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/H/H05/H05-1021.

Manuel Laguna, Rafael Martí, and Vicente Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & Operations Research*, 26(12):1217–1230, October 1999.

Alon Lavie and Abhaya Agarwal. METEOR: An automatic metric for MT evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 228–231, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W07/W07-0734.

Alon Lavie, Kenji Sagae, and Shyamsundar Jayaraman. The signicance of recall in automatic metrics for mt evaluation. In Robert E. Frederking and Kathryn B. Taylor, editors, *Machine Translation: From Real Users to Research*, pages 134–143. Association for Machine Translation in the Americas, Springer, September–October 2004.

Hendrik W. Lenstra, Jr. The acyclic subgraph problem. Report BW26, Mathematisch Centrum, Amsterdam, 1973.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 761–768, Sydney, Australia, July 2006a. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P06/P06-1096.

Percy Liang, Ben Taskar, and Dan Klein. Alignment by agreement. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 104–111, New York City, USA, June 2006b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N06/N06-1014.

Olivier C. Martin and Steve W. Otto. Combining simulated annealing with local search heuristics. Technical Report CS/E 94-016, Oregan Graduate Institute Department of Computer Science and Engineering, 1994.

Andrew McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of UAI*, 2003.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan, June 2005a. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P05/P05-1012.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Spanning tree methods for discriminative training of dependency parsers. Technical Report MS-CIS-05-11, UPenn CIS, 2005b.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 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 2005c. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/H/H05/H05-1066.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220, New York City, June 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W06/W06-2932.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

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

John E. Mitchell and Brian Borchers. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operations Research*, 62(1):253–276, December 1996.

John E. Mitchell and Brian Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In H. L. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 349–366. Kluwer Academic Publishers, Dordrecht, 2000.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, June 1997.

Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, March 2000.

Mehryar Mohri. Generic epsilon-removal algorithm for weighted automata. In Sheng Yu and Andrei Paun, editors, *5th International Conference on Automata (CIAA 2000)*, volume 2088 of *Lecture Notes in Computer Science*, pages 230–242. Springer-Verlag, 2001.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. In A. Kornai, editor, *Proceedings of the ECAI 96 Workshop*, pages 46–50, 1996.

Mehryar Mohri, Fernando Pereira, and Michael Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32, January 2000.

Alexander Nareyek. 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, 2003.

Franz Josef Och. Minimum error rate training in statistical machine translation. In Erhard Hinrichs and Dan Roth, editors, *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, 2003. URL http://www.aclweb.org/anthology/P03-1021.pdf.

Franz Josef Och and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449, December 2004.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, March 2003.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 00–00, University of Pennsylvania, 2002. URL http://www.aclweb.org/anthology/P02-1040.pdf.

Chris N. Potts and Steef L. van de Velde. Dynasearch—Iterative local improvement by dynamic programming. Part I. The Traveling salesman problem. Technical report, University of Twente, The Netherlands, 1995.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

Tommaso Schiavinotto and Thomas Stützle. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, December 2004.

Anne Schiller, Simone Teufel, and Christine Thielen. Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Universität Stuttgart, September 1995.

Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, pages 44–49, Manchester, 1994.

Helmut Schmid. Improvements in part-of-speech tagging with an application to German. In Susan Armstrong, Kenneth Church, Pierre Isabelle, Sandra Manzi, Evelyne Tzoukermann, and David Yarowsky, editors, *Natural Language Processing Using Very Large Corpora*, volume 11 of *Text, Speech and Language Technology*, pages 13–26. Kluwer, Dordrecht, 1999.

Helmut Schmid. Disambiguation of morphological structure using a PCFG. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 515–522, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/H/H05/H05-1065.

Ernst Schröder. Vier combinatorische Probleme. *Zeitschrift für Mathematik und Physik*, 15:361–376, 1870.

Dana Shapira and James A. Storer. Edit distance with move operations. In Alberto Apostolico and Masayuki Takeda, editors, *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching*, number 2373 in LNCS, pages 85–98, Fukuoka, July 2002. Springer.

Louis Shapiro and A. Brooke Stephens. Bootstrap percolation, the Schröder numbers, and the n-kings problem. *SIAM Journal on Discrete Mathematics*, 4(2):275–280, May 1991.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 88–95, Washington, DC, March–April 1997. Association for Computational Linguistics.

Patrick Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 48 (3/4):303–312, 1961.

David A. Smith and Jason Eisner. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794, Sydney, Australia, July 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P06/P06-2101.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 354–362, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P05/P05-1044.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*, 2006.

Andreas Stolcke and Stephen Omohundro. 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, 1993. URL citeseer.ist.psu.edu/stolcke93hidden.html.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. Max-margin parsing. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 1–8, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Christoph Tillmann. A unigram orientation model for statistical machine translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Short Papers*, pages 101–104, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

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

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 836–841, Copenhagen, August 1996.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. Online large-margin training for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 764–773, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D07/D07-1080.

Dekai Wu. An algorithm for simultaneously bracketing parallel texts by aligning words. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 244–251, Cambridge, Massachusetts, USA, June 1995. Association for Computational Linguistics. doi: 10.3115/981658.981691. URL http://www.aclweb.org/anthology/P95-1033.

Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, September 1997. URL http://acl.ldc.upenn.edu/J/J97/J97-3002.pdf.

Fei Xia and Michael McCord. Improving a statistical MT system with automatically learned rewrite patterns. In *Proceedings of Coling 2004*, pages 508–514, Geneva, Switzerland, August 2004. COLING.

Deyi Xiong, Qun Liu, and Shouxun Lin. Maximum entropy based phrase reordering model for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 521–528, Sydney, Australia, July 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P06/P06-1066.

Richard Zens and Hermann Ney. A comparative study on reordering constraints in statistical machine translation. In Erhard Hinrichs and Dan Roth, editors, *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 144–151, 2003. URL http://www.aclweb.org/anthology/P03-1019.pdf.

Ying Zhang and Stephan Vogel. Measuring confidence intervals for the machine translation evaluation metrics. In *Proceedings of the Tenth Conference on Theoretical and Methodological Issues in Machine Translation*, pages 85–94, Baltimore, October 2004.

Vita

Roy Tromble was born in Juneau, Alaska, in November, 1979. He graduated from Juneau-Douglas High School in 1997, one of three co-valedictorians. He attended the University of Idaho, and finished in 2001 with degrees in Computer Science and Mathematics, both *summa cum laude*.

Roy spent a year after graduation working at the Center for Secure and Dependable Software, at the University of Idaho, as a research associate. He started the Ph.D. program at Johns Hopkins University in September, 2002.

Roy married Nicole Taylor, whom he met at the University of Idaho, in August, 2003. Their first child, Rachel, was born at Johns Hopkins Hospital in June, 2008. They reside in Pittsburgh, Pennsylvania, where Roy works as a Software Engineer at Google.