DISSERTATION

EMPIRICAL MODELING AND ANALYSIS OF LOCAL SEARCH ALGORITHMS FOR THE JOB-SHOP SCHEDULING PROBLEM

Submitted by

Jean-Paul Watson

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2003

COLORADO STATE UNIVERSITY

July 31, 2003

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY JEAN-PAUL WATSON ENTITLED EMPIRICAL MODELING AND ANALYSIS OF LOCAL SEARCH ALGORITHMS FOR THE JOB-SHOP SCHEDULING PROBLEM BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Committee Member

Committee Member

Adviser

Co-Adviser

Department Head

ABSTRACT OF DISSERTATION

EMPIRICAL MODELING AND ANALYSIS OF LOCAL SEARCH ALGORITHMS FOR THE JOB-SHOP SCHEDULING PROBLEM

Local search algorithms are among the most effective approaches for locating high-quality solutions to a wide range of combinatorial optimization problems. However, our theoretical understanding of these algorithms is very limited, leading to significant problems for both researchers and practitioners. Specifically, the lack of a theory of local search impedes the development of more effective algorithms, prevents practitioners from identifying the algorithm most appropriate for a given problem, and allows widespread conjecture and misinformation regarding the benefits and/or drawbacks of particular algorithms. This thesis represents a significant step toward a theory of local search. Using empirical methods, we develop theoretical models of the behavior of four well-known local search algorithms: a random walk, tabu search, iterated local search, and simulated annealing. The analysis proceeds in the context of the well-known job-shop scheduling problem, one of the most difficult NP-hard problems encountered in practice. The large volume of prior research on the job-shop scheduling problem provides a diverse range of available algorithms and problem instances, in addition to numerous empirical observations regarding local search algorithm behavior; the latter are used to validate our behavioral models.

We show that all four local search algorithms can be modeled with high fidelity using

straightforward variations of a generalized one-dimensional Markov chain. The states in these models represent sets of solutions a given fixed distance from the nearest optimal solution. The transition probabilities in all of the models are remarkably similar, in that search is consistently biased toward solutions that are roughly equidistant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. Surprisingly, the qualitative form of the transition probabilities is simply due to the structure of the representation used to encode solutions: the binary hypercube. The models account for between 96% and 99% of the variability in the cost required to locate both optimal and sub-optimal solutions to a wide range of problem instances, and provide explanations for numerous phenomena related to problem difficulty for local search in the job-shop scheduling problem. In the course of our analysis, we also disprove many conjectures regarding the behavior and benefits of particular algorithms.

Our research indicates that despite their effectiveness, local search algorithms for the job-shop scheduling problem exhibit surprisingly simple run-time dynamics. Further, we observe minimal differences between the dynamical behavior of different algorithms. As expected given similar run-time dynamics, although contrary to numerous reports appearing in the literature, we also show that the performance of different algorithms is largely indistinguishable. Ultimately, our behavioral models serve to unify and provide explanations for a large body of observations regarding problem difficulty for local search in the job-shop scheduling problem, and identify new research areas for the development of more effective local search algorithms.

Jean-Paul Watson
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
Fall 2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Optimization problems are ubiquitous in both scientific and industrial settings. Given an instance Π of an optimization problem, the objective is to find a solution s to Π such that some real-valued cost function $\mathcal{F}(s)$ is either maximized or minimized, depending on the application. Algorithms that are guaranteed to find such extremal values are known as *exact* algorithms. Many important optimization problems (e.g., the Traveling Salesman Problem (TSP) and the Maximum Satisfiability Problem (MAX-SAT)) are known to be NP-hard [GJ79], such that locating optimal solutions is, unless $P = NP$, computationally prohibitive: in the worst case, run-time is an exponential function of the size of Π. Given the apparently intractable nature of most optimization problems, the obvious practical alternative is to employ *approximation* algorithms, which typically locate sub-optimal solutions to Π, but in reasonable run-times.

Both exact and approximation algorithms for NP-hard optimization problems are generally based on one of the following two paradigms: constructive search and local search. *Constructive* search algorithms build solutions incrementally, often using a significant amount of problem-specific knowledge to guide the search process. Well-known examples of constructive search algorithms are branch-and-bound and constraint programming. Exact algorithms are almost exclusively based on constructive search, using complete backtracking mechanisms to systematically explore the entire search space. In contrast, *local* search algorithms operate by performing iterative modifications to one or more fully specified solutions, and generally rely on little problem-specific knowledge to guide search. Genetic algorithms, simulated annealing, and tabu search are all well-known examples of local search algorithms. Local search algorithms are generally haphazard in their exploration of the search space, and as a consequence, application of the paradigm is restricted to development of approximation algorithms.

Until the mid-1980s, both exact and approximation state-of-the-art algorithms for nearly all NP-hard optimization problems were based on constructive search. These algorithms typically required problem-specific knowledge to both guide search and to eliminate large regions of the search space from consideration, knowledge that often required years of research to develop. With few exceptions, e.g., the Iterated Lin-Jungian

algorithm for the TSP [LK73], local search algorithms were generally ignored because they failed to achieve performance that was competitive with the best constructive search algorithms. This situation started to change in the mid-to-late 1980s with the introduction of two new local search methods: simulated annealing and tabu search. Researchers found that local search algorithms based on simulated annealing and tabu search often equaled or outperformed the best constructive algorithms for a given NP-hard problem, relied on significantly less problem-specific knowledge, and were far easier to implement. The combination of simplicity and good performance especially interested practitioners, which, in turn, fueled a subsequent explosion of interest in local search, which continues unabated. Today, state-of-the-art approximation algorithms for any given optimization problem are frequently based on local search.

Despite widespread success, very little is known about *why* local search algorithms work so well and under what conditions. This situation is largely due to the fact that researchers typically focus on demonstrating, and not analyzing, algorithm performance. Most local search algorithms are developed in an ad-hoc manner. A researcher devises a new search strategy or a modification to an existing strategy, typically arrived at via intuition. The algorithm is implemented, and performance is compared with that of existing algorithms on sets of widely available benchmark problems. If the new algorithm outperforms existing algorithms, the results are published, advancing the state-of-the-art. Unfortunately, most researchers fail to actually prove that the proposed enhancement(s) actually led to the observed performance increase (typically, multiple new features are introduced simultaneously) or whether the increase was due to fine-tuning of the algorithm, implementation tricks, flaws in the comparative methodology, or some other factor(s). Hooker [Hoo95] refers to this approach to algorithm development as the *competitive testing* paradigm, arguing that "this *modus operandi* spawns a host of evils that have become depressingly familiar to the algorithmic research community" and that the "emphasis on competition is fundamentally anti-intellectual" [Hoo95], (p. 33). However, although few would argue with Hooker's criticisms, the competitive testing paradigm remains the dominant paradigm for developing new algorithms – independent of whether they are exact or approximate, or based on constructive or local search.

Due largely to the widespread practice of competitive testing, theoretical results concerning the operation of local search algorithms are very limited. In particular, we currently lack fundamental models of local search algorithm behavior. The importance of behavioral models cannot be understated. Ideally, behavioral models enable practitioners to identify those problems for which a particular local search algorithm is likely to be effective and those problem instances that are likely to be more difficult than others. The availability of behavioral models also allows researchers to identify fundamental similarities and differences between different local search algorithms and identify new research directions in order to improve the performance of existing local search algorithms. In contrast, the current lack of behavioral models has led to several undesirable side-effects, including widespread conjecture and mythology regarding the benefits and/or operation of particular local search algorithms.

This thesis develops several behavioral models of local search algorithms. The approach taken is both *theoretical* and *empirical*. Many in the algorithmic research community view these two terms as incongruent, instead associating the terms *theoretical* and *analytical*. Hooker [Hoo94] argues that there is no justifiable reason to favor analytical models over empirical models in the analysis of algorithms. In the hard sciences such as physics and chemistry, a theory is a model that provides a cohesive explanation for a set of existing observations, and ideally makes predictions about the outcomes of new experiments, i.e., a theory is falsifiable. Thus, there is really nothing contradictory about developing theories using empirical methods; as Hooker notes ([Hoo94], p. 203), "Anyone who thinks that empirical science is non-theoretical should take a look at quantum electrodynamics." The main objection to the empirical analysis of algorithms appears largely due to the fact that an algorithm is really a formal system, and therefore, the consequences (i.e., behavior) of the system should be deducible via an analytical approach. Why favor an empirical over analytic approach to developing a theory of local search? The problem with this analytic, reductionist approach is that the *scale* of the analysis is inappropriate. For example, consider the scenario in which biologists attempted to model cell dynamics in terms of quantum mechanics; even if it was feasible to develop such a model, the detail and scale of the model would likely prevent any significant insights into the higher-level dynamics of the cell life-cycle. Returning to the present context, "Even if one can in principle *deduce* what the algorithms are going to do, it is beyond human powers to do so, and even if we did, we would not *understand why* they behave as they do" ([Hoo94], p. 205).

In modeling local search algorithms, the primary behavior of interest is the cost required to locate globally optimal solutions to a specific problem; the ultimate objective of a local search algorithm is to solve optimization problems. For a given optimization problem, different sub-classes of problem instances can be defined, for example in terms of either dimensionality or through the specification of individual problem features. Problem difficulty can vary tremendously, typically by many orders of magnitude, both between and within such sub-classes. Because local search algorithms are typically used to solve broad sub-classes of problem instances, and not individual instances, our objective in modeling the behavior of local search algorithms is to *account for the variability in problem difficulty observed in problem instances belonging to a particular sub-class*. If a single model accounts for the variability in problem difficulty observed within multiple sub-classes, any differences in difficulty between the sub-classes can be explained in terms of differences in the distribution of problem instances; otherwise, differences must also be explained in terms of the discrepancies between multiple models.

Extant models of local search algorithms are typically expressed in terms of features present in the underlying search space. Here, the goal is to identify those features that are most highly correlated with search cost, i.e., problem difficulty. Yet, despite their simplistic form – algorithm run-time behavior is explicitly ignored – the accuracy of the resulting models, if they are rigorously quantified at all, is generally quite poor. With the possible exception of specialized local search algorithms for MAX-SAT, an optimiza-

tion formulation of the well-known Boolean Satisfiability Problem, those search space features that most heavily influence problem difficulty for local search remain elusive. Research on more complex models of local search algorithms, which capture at least some aspects of algorithm run-time behavior and, consequently, allow deeper insights, is nearly non-existent.

The primary goal of this thesis is to develop behavioral models of a number of well-known and widely-applied local search algorithms. The proposed models vary in complexity, ranging from simple models based on search space features to those that explicitly model run-time dynamics with very high fidelity. In addition to accounting for variability in the difficulty of different problem instances, the models provide significant insight into why local search algorithms can be so effective. Further, by explicitly modeling algorithm run-time dynamics, it is possible to compare and contrast different local search algorithms in terms of their actual behavior, as opposed to more superficial characteristics. Ultimately, the availability of accurate behavioral models should eliminate much of conjecture surrounding local search and place the field on a firmer scientific footing.

One current limitation of the behavioral models developed in this thesis bears immediate disclosure: the models are primarily descriptive, in that they provide *a posteriori* explanations for the observed variability in problem difficulty. The reason is simply that to achieve exceptionally high accuracy levels, it appears necessary to base the behavioral models on information that is, in general, computationally intractable to obtain, specifically the set of optimal solutions to a problem instance. Consequently, there is currently no feasible way to predict the difficulty of a given problem instance. However, this does not limit the contribution of the models, which unambiguously identify both (1) those search space features that are highly correlated with problem difficulty and (2) the dynamics governing local search processes. Identification of the former provides sharp focus for future research on predictive models of problem difficulty: Can these features be estimated with any reasonable level of accuracy, and with what cost? Models of search processes enable cross-comparisons of different local search algorithms based on their search dynamics and identify those areas required to improve search efficiency. Further, the models are also used to form various hypotheses – which are subsequently verified – regarding other behavioral aspects of local search algorithms.

To develop behavioral models of local search algorithms, it is first necessary to select a specific context, i.e., optimization problem. The problem selected for this thesis is the Job-Shop Scheduling Problem, or JSP. Given the vast range of alternatives, the obvious question is "Why?" The motivation is three-fold. First, the JSP is an interesting optimization problem in its own right, as it is one of the most difficult NP-hard problems encountered in practice, and despite years of intense study, little is known about what makes this problem so difficult. Second, in contrast to many other NP-hard problems, numerous local search algorithms have been introduced for the JSP, enabling a relatively strict focus on algorithm analysis, as opposed to development *and* analysis. Third, the volume of research devoted to the JSP has resulted in numerous observations relating to

both the relative difficulty of problem instances and the behavior of local search algorithms. Most of these observations currently lack satisfactory explanations and provide a further opportunity to validate any behavioral models of local search: more robust models account for a wider range of observations.

The remainder of this thesis is organized as follows. Background on the JSP and local search algorithms for the JSP are provided in Chapters 2 and 3, respectively. The specific types of behavioral models considered and the empirical methodology used to develop these models are described in Chapter 4. Ultimately, local search algorithm behavior, and, consequently, any model of this behavior is a function of the interaction of the search strategy with the underlying search space. A detailed analysis of the primary structural characteristics of the JSP search space is described in Chapter 5. The core of the thesis is presented in Chapters 6 through 9, where behavioral models for the following local search algorithms are developed: a random walk, tabu search, iterated local search, and simulated annealing. The latter three algorithms have seen widespread application on a wide variety of combinational optimization problems; a pure random walk is included as a baseline, which, despite its ineffectiveness, is closely related to the other algorithms. Most research on the JSP, including that in much of this thesis, is based on randomly generated problem instances. An obvious question is then the generalization of the behavioral models to more realistic, structured problem instances; this issue is analyzed in Chapter 10. The current state-of-the-art algorithm for the JSP is a variant of tabu search that re-intensifies search around previously encountered high-quality solutions. It is an open question whether such re-intensification can equally benefit other local search algorithms; the answer to this question is provided in Chapter 11. The thesis concludes in Chapter 12 with a discussion of the implications of this research, a laundry-list of open questions, and a summary of the overall contribution to the algorithmic research community.

Chapter 2

The Job-Shop Scheduling Problem

In the most general terms, a scheduling problem specifies a set of tasks to be performed using a finite set of resources. The objective is to schedule processing of all the tasks such that some overall measure of efficiency is maximized, e.g., the time required to complete all tasks. Scheduling is ubiquitous in real-world applications, ranging from operating systems to large-scale military logistics problems. The details of particular scheduling problems are highly variable, which led researchers to develop "distilled" scheduling problems, each abstractly representing a particular class of real-world problem. Although numerous abstract scheduling problems have been introduced (e.g., see the recent books by Blażewicz et al. [BSE$^+$96], Brucker [Bru01], and Pinedo [Pin01]), the most widely studied problem is the static, deterministic job-shop scheduling problem, hereafter referred to simply as the JSP. In the JSP there are n jobs and m machines. Each job must be processed on each machine exactly once, for a fixed duration, and in a pre-specified order; the processing of a job on a machine is called an operation. Each machine can only process a single operation at a time, and once initiated, an operation must be processed to completion. Various scheduling objectives have been introduced for the JSP. However, most research considers the objective of *makespan minimization*, i.e., minimizing the maximal completion time of all jobs [BDP96].

Although the JSP is the most widely studied abstract scheduling problem, it is also, paradoxically, one of the least realistic. Mattfeld [Mat96] notes the following restrictions inherent in the definition of the JSP, many of which need not hold in any given real-world scheduling problem (French [Fre82] provides a similar list):

- No two operations of one job may be processed simultaneously.
- No preemption (i.e., process interruption) of operations is allowed.
- No job is processed twice on the same machine.
- Each job must be processed to completion.
- Jobs may be started at any time, i.e., no release times exist.
- Jobs may be finished at any time, i.e., no due dates exist.

- Jobs must wait for the next machine in the processing order to become available.
- No machine may process more than one operation at a time.
- Machine setup times are negligible.
- There is only one type of each machine.
- Machines may be idle within the schedule period.
- Machines are available at any time.
- The machine processing orders of each job is known in advance and is immutable.

Consequently, algorithms for solving the JSP are of little direct use to practitioners, who must deal with the intricacies of their particular scheduling problem. McKay et al. [MSB88] even argue that "the [JSP] problem definition is so far removed from job-shop reality that perhaps a different name for the research should be considered." Yet, despite the lack of realism, the JSP has drawn intense attention from researchers in a variety of disciplines, ranging from operations research to computer science and mathematics; many of which are at best peripherally involved with real-world scheduling problems.

Why the intense attention on a problem of little practical significance? Mattfeld (p. 8, 1996) [Mat96] argues that "... benefit from previous research can only be obtained if a widely accepted standard model exists." However, the primary draw for most researchers is the fact that the JSP is notoriously difficult, widely accepted as empirically one of the most difficult NP-hard problems, and consequently poses a constant intellectual challenge: despite over 40 years of effort, resulting in hundreds of published journal articles and tens of dissertations, even state-of-the-art algorithms often fail to consistently locate optimal solutions to relatively small problem instances (e.g., see Jain [Jai98], Tables 2.4 through 2.10). The inherent intractability (unless $P = NP$) of the JSP is often illustrated by considering the history surrounding attempts to locate optimal solutions to a small benchmark problem instance introduced in 1963 by Fisher and Thompson [FT63], which was not solved to optimality until 1986. Jain and Meeran [JM99] provide an overview of the algorithmic developments leading to the solution of this famous problem instance.

In the remainder of this chapter, we introduce the job-shop scheduling problem and a number of related concepts that are integral to understanding and developing local search algorithms for its solution. We begin in Section 2.1 and introduce key sub-classes of problem instances in Section 2.2; empirical observations regarding the relative difficulty of JSP instances are discussed in Section 2.3. In Sections 2.4 through 2.7, we discuss the concepts, specification, and visualization of solutions and schedules, in addition to methods for quantifying the similarities between them. We conclude in Section 2.8 by discussing the notion of a critical path, a concept central to the design of local search algorithms for the JSP, which we discuss in Section 3.

2.1 Definition, Notation, and Computational Complexity

In the $n \times m$ job-shop scheduling problem (JSP), there are n jobs, each of which must be processed exactly once on each of m machines. Each job i is routed through the m machines in some pre-defined order π_i, where $\pi_i(j)$ denotes the jth machine in the routing order; clearly, $1 \leq i \leq n$ and $1 \leq j \leq m$. Conversely, let $\pi_i^{-1}(j)$ denote the position of machine j in the routing order of job i. The processing of job i on machine $\pi_i(j)$ is denoted o_{ij} and is called an *operation*. An operation o_{ij} is processed on machine $\pi_i(j)$ for an integral duration $\tau_{ij} \geq 0$. For $2 \leq j \leq m$, o_{ij} cannot initiate processing until o_{ij-1} has completed. The constraints imposed by the job routing orders and unit capacity restrictions are respectively referred to as *precedence constraints* and *resource constraints*. In contexts where the job and routing order indices of an operation are irrelevant, we drop the dual subscript notation in favor of the single subscript notation, e.g., o_k or τ_k.

Each solution s to an instance Ω of the $n \times m$ JSP specifies a processing order for all of the jobs on each machine. There are $n!$ possible processing orders for each machine, resulting in $(n!)^m$ possible solutions to Ω. However, these solutions are often *infeasible* such that the combination of job routing orders π_i and the machine processing orders results in a cyclic ordering dependency between at least one pair of operations. Clearly, realizable job-shop schedules cannot be obtained from infeasible solutions. A solution s with no such cyclic dependencies is said to be *feasible*. We denote the set of all feasible solutions to an instance Ω by S_Ω.

Each solution $s \in S_\Omega$ implicitly specifies an earliest possible start time est_{ij} for each operation o_{ij} such that all precedence and resource constraints are satisfied; algorithms for computing the set of est_{ij} are discussed in Section 3. The earliest possible completion time ect_{ij} for an operation o_{ij} is then given by $ect_{ij} = est_{ij} + \tau_{ij}$. The makespan $C_{max}(s)$ of a solution $s \in S_\Omega$ is the maximum earliest completion time of the last operation of any job i: i.e., $C_{max}(s) = max(ect_{1m}, ect_{2m}, \cdots, ect_{nm})$. The optimal makespan, denoted C^*_{max}, is equal to the minimal makespan of any solution in the set of all feasible solutions, i.e., $C^*_{max} = min_{\forall s \in S_\Omega}(C_{max}(s))$.

The decision problem of finding a solution to the JSP with a makespan less than or equal to some constant L is known to be NP-complete for $m \geq 2$ and $n \geq 3$ [GJS76]. Consequently, the optimization problem of locating a solution with a makespan equal to C^*_{max} is (strongly) NP-hard. Even accurate approximation in the JSP is provably intractable: unless $\mathcal{P} = \mathcal{NP}$, no polynomial-time algorithm exists that can guarantee a solution to the JSP within 20% of the optimal makespan [WHH$^+$97].

	Operation					
Job	o_{i1}		o_{i2}		o_{i3}	
(i)	τ_{i1}	$\pi_i(1)$	τ_{i2}	$\pi_i(2)$	τ_{i3}	$\pi_i(3)$
1	5	1	8	2	2	3
2	7	3	3	1	9	2
3	1	1	7	3	10	2
4	2	2	3	3	1	1

Table 2.1: Specification of a 4×3 random JSP.

2.2 Specification and Classification of Problem Instances

An instance Ω of the $n \times m$ JSP is completely specified by the set of nm operation durations τ_{ij} and n job routing orders π_i. We show the specification for a small 4×3 instance originally introduced by Foo and Takefuji [FT88] in Table 2.1. Beginning with Fisher and Thompson [FT63], researchers typically generate problem instances by sampling the τ_{ij} independently and uniformly from a fixed-width interval $[\mathit{LB}, \mathit{UB}]$. Most often $\mathit{LB} = 1$ and $\mathit{UB} = 99$, e.g., see Taillard [Tai93] or Demirkol et al. [DMU98]. The definition of the JSP places no *a priori* restrictions on the form of the job routing orders π_i. Consequently, researchers frequently construct the π_i by generating random permutations of the integers $[1..m]$. We define *random* JSPs as instances constructed by (1) sampling the τ_{ij} uniformly from a fixed-width interval and (2) generating the π_i from random permutations of the integers $[1..m]$. Most well-known JSP benchmark instances are random JSPs. The instance shown in Table 2.1 is a random JSP, with the τ_{ij} sampled from the interval $[1, 10]$.

Researchers have introduced specializations of the random JSP by imposing specific constraints on the job routing orders that more accurately reflect features found in some real-world job shops. In these more structured JSPs, the set of m machines is partitioned into *wf* contiguous, equally-sized subsets called *workflow partitions*. Clearly, $\mathit{wf} = 1$ in random JSPs. In contrast, when $\mathit{wf} = 2$, the set of m machines is partitioned into two subsets containing the machines 1 through $m/2$ and $m/2+1$ through m, respectively. In these problems, every job must be processed on all machines in the first partition before proceeding to any machine in the second partition. No constraints are placed on the job routing orders within each partition. We refer to JSPs with $\mathit{wf} = 2$ simply as *workflow* JSPs.

In a *flowshop* JSP, $\mathit{wf} = m$, and each job must visit the machines in an identical pre-determined order. The flowshop JSP is closely related to the Permutation Flow-Shop Problem (PFSP), in which the job processing orders for all machines is identical. However, the optimal makespan to a PFSP instance is typically higher than the optimal makespan of the corresponding flowshop JSP, where the job processing orders of differ-

ent machines can be variable. For most benchmark problems, the difference is relatively small. The worst-case difference is substantially larger; Potts et al. [PSW91] show that the optimal makespan of a PFSP can exceed that of the corresponding flowshop JSP by a factor of more than $\frac{1}{2}\sqrt{m}$.

To date, no research has been directed toward generating problem instances with structured τ_{ij}, although it is well-known that real-world scheduling problems typically possess non-random operation processing times [PDS73] [DPS92]. Further, there is evidence that problem structure can have a significant impact on the algorithm performance [WBWH02]. Although we previously introduced a method for generating structured permutation flow-shop scheduling problems [WBWH02], which can be easily extended to the JSP, we do not consider such problem instances here. The present goal is to analyze and understand the large body of research already performed on the JSP, which necessarily constrains analysis to existing classes of problem instances.

2.3 The Relative Difficulty of Problem Instances

While developing algorithms for solving the JSP, researchers have made numerous qualitative observations regarding the relative difficulty of different problem instances. With few exceptions, these observations appear to be algorithm-independent, such that relative difficulty with respect to constructive algorithms such as branch-and-bound also holds for local search. For our purposes, these observations serve to validate the models of problem difficulty for local search that we develop in Chapters 6 through 9. In particular, our models should at *least* be consistent with these observations, and ideally should provide *explanations* (relative to local search algorithms) for the observed behavior.

The primary observations regarding the relative difficulty of JSP instances, which have emerged from nearly 50 years of research, are as follows:

- Given fixed n and m, workflow JSPs are typically more difficult than random JSPs.
- Given fixed n and m, flowshop JSPs are typically more difficult than workflow JSPs.
- Given fixed m and *wf*, *square* (i.e., $n/m \approx 1$) JSPs are typically more difficult than *rectangular* (i.e., $n/m \gg 1$) JSPs.
- Given fixed n, m, and *wf*, relative problem difficulty is largely algorithm-independent.

We now describe each of these observations in more detail, noting in advance that although researchers have proposed intuitive explanations for each of these observations, rigorous causal explanations remain elusive.

Workflow and flowshop JSPs differ from random JSPs in that the job routing orders π_i are more structured. While the presence of structure often makes problems *easier*

to solve, assuming the existence of an algorithm that exploits this structure, this is not the case in the JSP. Specifically, given fixed, arbitrary n and m, the average difficulty of problem instances, as measured either by the cost required to locate an optimal solution or to *prove* optimality of a solution, is proportional to *wf*, i.e., those instances that possess more workflow partitions are likely, on average, to be more difficult. In particular, the most difficult instances are flowshop JSPs, where $wf = m$. Evidence for this observation stems from a wide variety of sources. For example, Storer et al. [SWV92] introduced sets of 50×10 random and workflow JSPs in 1992; since then, the random instances have been solved to optimality, while the optimal makespans of all but one of the workflow instances are unknown. Similarly, the most difficult 10×10 benchmark instances, Fisher and Thompson's infamous 10×10 instance and the `orb` instances introduced by Applegate and Cook [AC91], are all "nearly" workflow or flowshop instances, in that the requirement that a job be processed on all machines in one workflow partition before proceeding to any machine in the next workflow partition is occasionally violated.

Large differences in the relative difficulty of square (i.e., $n/m \approx 1$) versus rectangular (i.e., $n/m > 1$) JSPs are easily illustrated by considering the best-known makespans for the problems in Taillard's benchmark suite of JSP instances, which we discuss in Section 4.6. Here, the optimal makespans of all the relatively small 20×20 and 30×20 instances are currently unknown, while the optimal makespan for all of the larger 50×15, 50×20, and 100×20 instances are known, despite astronomical differences in the sizes of the search space. Taillard [Tai94] performed an in-depth analysis of changes in relative difficulty as n and m are varied, concluding that for a tabu search algorithm, the cost of locating optimal solutions grows *polynomially* with increases in n and m when $n/m \geq 6$. In contrast, when $n/m \approx 1$, increases in n (or equivalently m) yield apparently exponential increases in the cost required by the same tabu search algorithm to locate optimal solutions.

The third observation stems from the fact that in a set of fixed-sized problem instances with a given *wf*, the easy (difficult) instances are easy (difficult) for *all* search algorithms. So-called "open" problem instances, i.e., those for which the optimal makespan is not known, have resisted solution by numerous algorithms based on dramatically different computational paradigms. Similarly, when the optimal makespan is known, the computational effort required to locate an optimal solution is consistently predictable from the historical reputation of the difficulty of a given instance.

The class of JSP instances that are considered "intractable", in that the optimal makespan has not been determined, clearly depends on the state-of-the-art in available computing power. Jain and Meeran [JM99] indicate that a problem instance can be considered hard or intractable if it possesses the following properties:

- $n \cdot m \geq 200$
- $n \geq 15$
- $m \geq 10$

- $n < 2.5m$

They further observe that "The problem is made more intractable when [*wf*=2] and in such a situation n need not be less than or equal to $2.5m$" ([JM99], p. 407). Although this classification was introduced in 1998, it is still valid: significant advances in computing power has *not* significantly expanded the range of tractable JSP instances. It is currently beyond the power of exact algorithms to consistently determine optimal makespans to 20×15, 20×20, and 30×15 benchmark instances, and approximation algorithms have great difficulty locating optimal or even high-quality sub-optimal solutions to these same instances. Even though smaller instances, e.g., 15×15 are now considered tractable, this does *not* necessarily imply that it is easy for approximation algorithms to locate optimal solutions; rather, it is possible for *some* approximation algorithms, likely after extended periods of computation, to *occasionally* locate an optimal solution.

2.4 Solutions: Specification, Properties, and Notation

One approach to specifying solutions to an $n \times m$ instance Ω of the JSP is to simply list the job processing orders γ_j for each machine j, $1 \leq j \leq m$. For example, an optimal solution to the 4×3 instance shown in Table 2.1 is given as follows:

- $\gamma_1 = (1, 3, 2, 4)$
- $\gamma_2 = (4, 1, 2, 3)$
- $\gamma_3 = (2, 3, 4, 1)$

Each machine processing order γ_j is simply an ordered permutation of the integers $[1..n]$ such that $\gamma_j(i)$ denotes the ith job in the processing order of machine j. Conversely, let $\gamma_j^{-1}(i)$ denote the position of job i in the processing order γ_j of machine j.

The precedence constraints imposed by the specification of the instance Ω introduce at most one predecessor and one successor operation for any operation o_{ij}. We denote the corresponding "job" predecessors and successors of o_{ij} respectively by JP_{ij} and JS_{ij}, which are defined as follows:

- $\forall i \neq 1$, $JP_{ij} = o_{i(j-1)}$
- $\forall i \neq m$, $JS_{ij} = o_{i(j+1)}$
- $JP_{i1} = JS_{im} = *$

where $*$ denotes a "null" or non-existent operation.

Similarly, for any operation o_{ij}, the set of machine processing orders γ_j introduce at most one additional predecessor and successor operation apiece. We denote the corresponding "machine" predecessors and successors of o_{ij} respectively by MP_{ij} and MS_{ij},

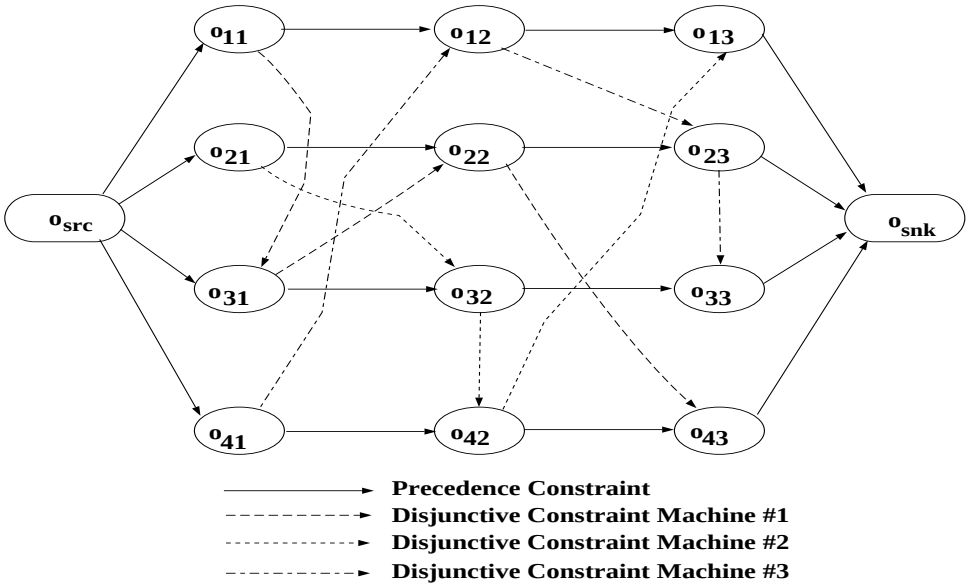

Figure 2.1: The disjunctive graph for an optimal solution to the 4×3 JSP instance shown in Table 2.1.

which are defined as follows, given that o_{ij} is processed on machine $x = \pi_i(j)$ at position $y = \gamma_x^{-1}(i)$:

- $\forall y \neq 1$, $MP_{ij} = o_{\gamma_x(y-1)\pi_{y-1}^{-1}(x)}$
- $\forall y \neq n$, $MS_{ij} = o_{\gamma_x(y+1)\pi_{y+1}^{-1}(x)}$
- if $y = 1$, $MP_{ij} = *$
- if $y = n$, $MS_{ij} = *$

where $*$ again denotes a non-existent operation.

An alternative method for specifying solutions is via a *disjunctive graph*. Instead of explicitly specifying a job processing sequence for each machine, a disjunctive graph specifies the relative processing order for each *pair* of jobs on a machine. Further, a disjunctive graph (redundantly) specifies the precedence constraints imposed by the job processing orders π_j. Formally, a disjunctive graph is a directed graph $G = (V, C \cup D)$. The vertex set V contains $nm + 2$ vertices, where nm of these vertices represent the operations of the n jobs, while the remaining two vertices represent dummy operations known as the *source* and the *sink*, which we respectively denote o_{src} and o_{snk}. Both o_{src} and o_{snk} are orphaned, in the sense that they do not belong to any job and are not processed on any machine, such that $\tau_{src} = \tau_{snk} = 0$. A precedence constraint is defined between (1) o_{src} and each operation o_{i1} and (2) each operation o_{im} and o_{snk}.

The edge set C contains a directed edge for each precedence constraint imposed by the job routing orders and the o_{src} and o_{snk} operations. The edge set D contains a

directed edge between each distinct pair of operations that are processed on the same machine. The edges $(o_{ij},o_{ik}) \in C$ are oriented such that $j < k$. Edges in C imposed by the source and sink operations are respectively oriented as (o_{src}, o_{i1}) and (o_{im}, o_{snk}). The edges $(o_{ij},o_{kl}) \in D$, by definition processed on the same machine $x = \pi_i(j) = \pi_k(l)$, are oriented such that $\gamma_x^{-1}(i) < \gamma_x^{-1}(k)$. In Figure 2.1, we show a disjunctive graph for an optimal solution, identical to the example solution specified via $\gamma_1 \cdots \gamma_3$ above, to the instance specified in Table 2.1.

The job processing order and disjunctive graph specifications are equivalent, in that one can be easily transformed into the other. Both methods can also be used to *represent* solutions in a search algorithm. For example, the source and sink operations serve to equip the disjunctive graph with a single entry and exit point, which in turn facilitates computation of various solutions attributes, including feasibility and operation start times. Although neither method is *directly* used to represent solutions for the local search algorithms we consider, aspects of both methods are incorporated, e.g., see Section 3.5.1. Further, the methods are used to compare and contrast various solutions, as discussed below in Section 2.5.

2.5 Quantifying Solution Similarity

The disjunctive graph specification of a solution also provides a straightforward measure of the similarity or "distance" between two solutions. Recall that the precedence constraints are independent of any particular solution to a given problem instance. Thus, two solutions differ only in the orientation of their disjunctive (directed) edges. For any solution $s \in S_\Omega$, the orientation of a disjunctive edge between two jobs i and j on machine k can be represented as a Boolean variable, which we denote $\mathit{precedes}_{i,j,k}(s)$. If job i is processed before job j on machine k in solution s, $\mathit{precedes}_{i,j,k}(s) = \mathit{true}$; otherwise, it is *false*. The distance $D(s_1, s_2)$ between two solutions $s_1, s_2 \in S_\Omega$ to an $n \times m$ JSP Ω is then given by

$$\sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \mathit{precedes}_{i,j,k}(s_1) \oplus \mathit{precedes}_{i,j,k}(s_2) \tag{2.1}$$

where the symbol $\oplus$ denotes the Boolean XOR operator. We denote the normalized distance $\frac{2 \cdot D(s_1,s_2)}{n(n-1)}$ by $\overline{D}(s_1, s_2)$ ($0 \leq \overline{D}(s_1, s_2) \leq 1$). Equation 2.1 can also be used to compute the distance between pairs of infeasible solutions or between infeasible and feasible solutions. Finally, nothing in the definition of Equation 2.1 is specific to the disjunctive graph – the distance can also be computed when solutions are specified as machine permutations; the term "disjunctive graph distance" is used primarily for historical reasons.

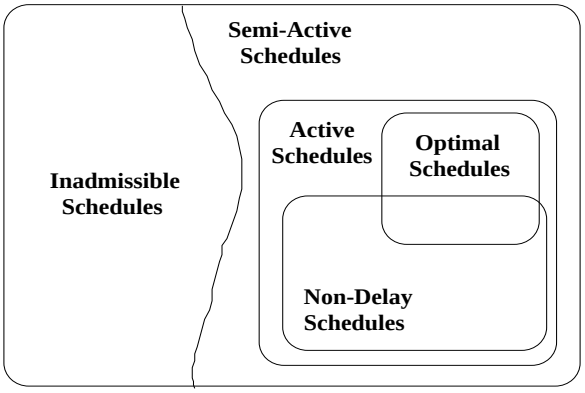

Figure 2.2: The hierarchy of schedules in the JSP.

2.6 Solutions versus Schedules and Schedule Taxonomy

A feasible *solution* $s \in S_\Omega$ specifies a job processing order for each machine in which there exist no cyclic ordering dependencies between pairs of operations. Any reference to time in a solution is implicit – an earliest possible start time can be computed for each operation. In contrast, a (feasible) *schedule* specifies a start time for each operation of each job such that the precedence and resource constraints are satisfied. In general, there is a one-to-many relationship between solutions and schedules, as there is no requirement to start operations at the earliest possible time; assuming an infinite time horizon, there are infinitely many possible schedules arising from any given solution. We refer to schedules in which operations are started later than their earliest possible start time as *inadmissible*. However, to achieve the makespan $C_{max}(s)$ it is necessary to schedule some subset of the operations at their earliest possible start time; the exact subset is introduced in Section 2.8. Consequently, operations are in practice assigned to their earliest possible start time; the resulting schedules are known as admissible or *semi-active*. Clearly, there exists a one-to-one relationship between solutions and (admissible) semi-active schedules; we use the two terms interchangeably.

In a semi-active schedule, the only way to reduce the makespan is to re-order the job processing order on at least one machine. In particular, it may be possible to move an operation earlier than its assigned start time *without* delaying the start time of any other operation, e.g., if there is idle time in the middle of a machine sequence. Such a move is called a *global left shift*, and a schedule in which no global left shifts are possible is called an *active* schedule. A schedule in which no machine is kept idle if there is an operation available for processing is called a *non-delay* schedule. By definition, all non-delay schedules are also active schedules.

We show the taxonomy of schedules in the JSP in Figure 2.2. At least one optimal schedule is necessarily active [GT60], but there may exist no optimal schedule that is non-delay; Fang [Fan94] provides counter-examples. Although restricting search to the set of active solutions appears advantageous, it is not always practical; the problem of

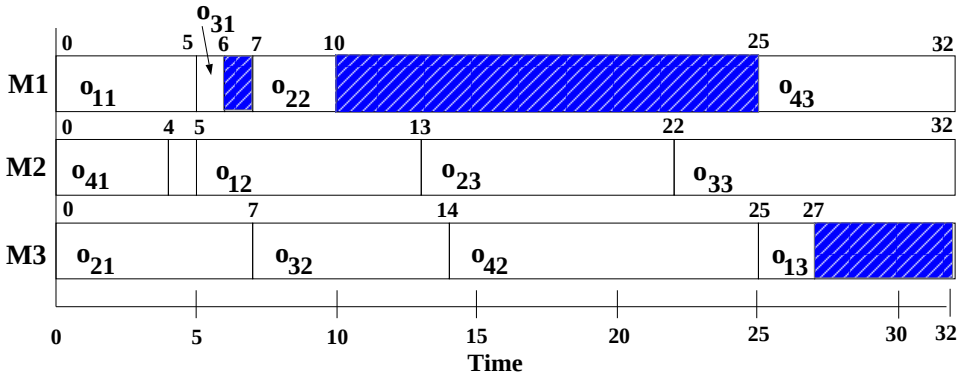

Figure 2.3: The Gantt chart visualization of the earliest start-time schedule for an optimal solution to the 4×3 JSP instance specified in Table 2.1. Shaded regions represent machine idle time.

verifying that a solution is active, i.e., checking that no global left-shifts are possible, is NP-complete [Vae95]. Consequently, most local search algorithms for the JSP restrict search to the space of semi-active solutions. Nevertheless, active and non-delay schedules do play an important role in local search, as we discuss in Section 3.5.3.

2.7 Visualizing Schedules

Solutions can be easily specified via a set of job processing orders or a disjunctive graph; however, in both cases the notion of time is only implicit. Although we could easily annotate these specifications with the assigned start time in the corresponding schedule (again, we assume semi-active scheduling), the result is difficult to interpret and fails to provide immediate insight into the qualitative nature of the overall schedule. To more effectively visualize schedules, we turn to a scheme known as a Gantt chart [Gan19], which simply records the activity of each machine over the duration of the schedule. In Figure 2.3, we provide a Gantt chart visualization of the earliest start-time schedule for an optimal solution, identical to that shown in Figure 2.1, to the 4×3 instance specified in Table 2.1; here, $C^*_{max} = 32$. The activity of each machine is indicated on a horizontal "time-line", where gray and white areas respectively represent active and idle times. Gantt charts can be used to identify bottleneck machines, potential flexibility in the schedule, and critical paths, as we discuss next in Section 2.8.

2.8 Critical Paths and Critical Blocks

A semi-active schedule is generated by starting operations in a solution s at their earliest possible time. However, it is possible to start some subset of operations later, yet still

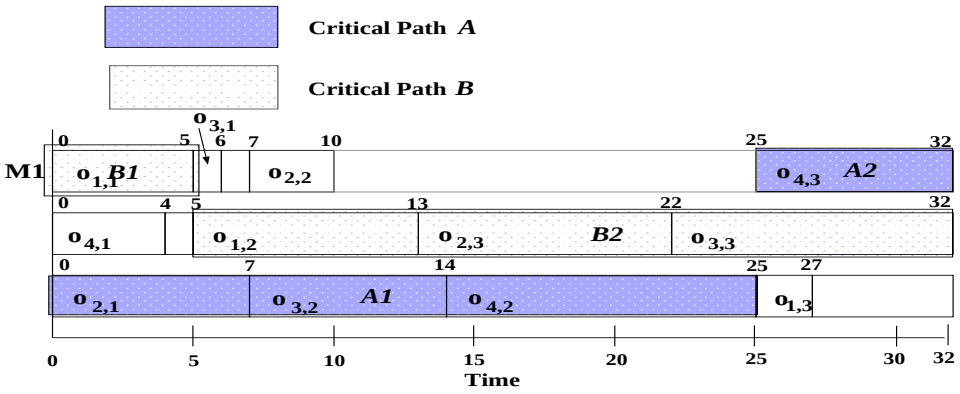

Figure 2.4: An earliest-start-time schedule for an optimal solution to the JSP instance shown in Table 2.1. The solution has two critical paths, labeled **A** and **B**.

yield a schedule with makespan $C_{max}(s)$. Let lft_{ij} denote the *latest finish time* of operation o_{ij} such that if all operations o_{ij} complete processing by lft_{ij} the resulting solution will have a makespan equal to $C_{max}(s)$; as with the earliest start times, the latest finish times are implicitly specified by s. Define the *latest start time* lst_{ij} of an operation o_{ij} as $lft_{ij} - \tau_{ij}$. In order to generate a schedule from s with makespan $C_{max}(s)$, all of the o_{ij} must be assigned a start time in the interval $[est_{ij}, lst_{ij}]$ such that the selected times are mutually consistent, i.e., all precedence and disjunctive constraints are satisfied.

The set of operations o_{ij} such that $est_{ij} = lst_{ij}$ are known as *critical* operations. If the actual start time of a critical operation is delayed by even a single time unit, then the makespan of the resulting schedule is guaranteed to be larger than $C_{max}(s)$. A contiguous sequence of critical operations starting at time $t = 0$ and ending at time $t = C_{max}(s)$ is called a *critical path*, such that no machine idle time exists along the path. In general, a solution s can have multiple critical paths, and a single critical operation can be shared by multiple critical paths.

Alternatively, we can define the *time-to-end* tte_{ij} of an operation o_{ij} in s as the difference between $C_{max}(s)$ and the latest possible end time of o_{ij} such that the schedule s is still feasible and has a makespan not exceeding $C_{max}(s)$. Consider a solution s and the *inverse* s' of s, obtained by inverting all of the edges in the disjunctive graph of s, such that the roles of o_{src} and o_{snk} are exchanged. The tte_{ij} of s are then equivalent to the est_{ij} of s'. Under this notation, an operation is critical if and only if $est_{ij}+\tau_{ij}+tte_{ij} = C_{max}(s)$, i.e., when there is no "slack" in the scheduling of o_{ij}.

In Figure 2.4, we show the earliest-start-time schedule for an optimal solution (identical to that shown in Figure 2.3) to the JSP instance shown in Table 2.1. The solution has two critical paths, **A** and **B**. Each critical path is composed of sub-sequences of critical operations on the same machine; these sub-sequences are called *critical blocks*. Critical path **A** is composed of critical blocks on machines 3 and 1, containing three operations and a single operation, respectively. Analogous, critical path **B** is composed of critical blocks on machines 1 and 3, respectively containing one and three operations.

No critical operation is shared by more than one critical path.

Critical paths play a central role in most high-performance algorithms for the JSP, for the following reason: given a complete and feasible solution $s \in S_\Omega$, the *only* way to reduce $C_{max}(s)$ is to re-order the sequence of operations on all critical paths [vLAL92]. Re-ordering the operations along a *single* critical path is *not* guaranteed to reduce $C_{max}(s)$, although several researchers implicitly make this assumption. The latter only holds when some of the critical operations on the path under consideration are shared by all critical paths; by re-ordering the operations on a random critical path, we are implicitly re-ordering the operations on all critical paths.

Chapter 3

Local Search and the Job-Shop Scheduling Problem

Local search is a general algorithmic paradigm for solving combinatorial optimization problems, including the JSP. Although local search algorithms first appeared in the 1970s, widespread application did not occur until much later. The first local search algorithms for the JSP were introduced in the late 1980s, when simulated annealing and tabu search were shown capable of locating high-quality solutions to notoriously difficult problem instances in relatively short periods of time. The success of these initial explorations fueled an explosion of interest in local search algorithms for the JSP, such that most variants of local search have been applied to the JSP by at least one group of researchers. Currently, algorithms for the JSP based on local search are widely acknowledged as state-of-the-art [BDP96] [JM99] [NS03]. The superiority of local search for the JSP is even more remarkable if one considers the competition: over the last 50 years, researchers have applied nearly every optimization method to the JSP, ranging from approaches based on mixed integer linear programming and Lagrangian relaxation to branch-and-bound and greedy constructive heuristics.

In this chapter, we provide a general overview of local search for combinatorial optimization, and detail its application to the JSP. We begin with a general definition of a combinatorial optimization problem in Section 3.1. The key concepts underlying local search algorithms, and how they are used to solve combinatorial optimization problems, are introduced in Section 3.2. The performance of local search depends to a large degree on the fitness landscape, a concept defined in Section 3.3. The introduction to local search concludes in Section 3.4 with a discussion of the general behavioral properties of these algorithms. We then shift to the specifics of local search algorithms for the JSP. The core components found in all of the local search algorithms considered in this thesis are detailed in Section 3.5. Much of the analysis of local search algorithms for the JSP presented in later chapters is based on the set of optimal solutions to a problem instance; the algorithm used for computing these sets is documented in Section 3.6.

3.1 Combinatorial Optimization Problems

Before defining a combinatorial optimization problem (COP), we first distinguish between a *problem* and a *problem instance*. Intuitively, a problem is a general description of a particular type of task. The details of a particular task are specified by a problem instance. For example, the task of determining whether a graph contains a Hamiltonian Cycle is a problem, while an instance of the Hamiltonian Cycle problem specifies the vertex and edge sets of a particular graph of interest. We denote a combinatorial optimization problem by Π and an instance of Π by Ω. Ω is drawn from some (possibly infinite) universe of possible problem instances, which we denote $\mathcal{U}_\Pi$.

An instance Ω of a combinatorial optimization problem Π is fully specified by two components: the state space and the objective function. The state space S is a finite, although typically astronomically large, set of possible solutions to Ω. The objective function F assigns a numeric 'worth' to each state $s \in S$. The only formal restriction on F is that there must exist a total order of the co-domain, such that a maximal or minimal value is well-defined. Typically, $F : S \rightarrow \mathbb{R}^+$ or $F : S \rightarrow \mathbb{Z}^+$. The objective function is commonly referred to as a *fitness function*.

Given a COP, the ultimate objective is to locate a solution $s \in S$ such that $F(s)$ is optimal, i.e., minimal or maximal. Without loss of generality, we assume the objective is minimization unless otherwise noted. Most COPs of practical interest are NP-hard, such that optimization is inherently intractable, or at the very least extremely expensive. Depending on the problem, sub-optimal solutions are acceptable, and the search objective is relaxed to locating the highest-quality solution possible in the alloted run-time. For example, in many scheduling, logistics, and transportation problems, all feasible sub-optimal solutions are executable, although they incur larger costs than absolutely necessary. In other problems, sub-optimal solutions are useless. For example, in the protein structure prediction problem, the objective is to predict the three-dimensional structure of a sequence of amino acids. Because fully folded natural proteins possess minimal free energy, solutions that yield sub-optimal free energy may exhibit very different three-dimensional structure than that observed in nature.

To solve a particular problem using local search, the problem must first be formulated as a COP. For many well-known and widely-studied NP-hard problems, formulation as a COP is straightforward. For example, in the Traveling Salesman Problem (TSP) a salesman is given a set of n distinct cities c_i, $1 \leq i \leq n$, and the distances $d(c_i, c_j) \in R^+$ between all distinct pairs of cities c_i and c_j. The problem for the salesman is to find a tour (i.e., a Hamiltonian cycle) such that the tour length is minimal. A COP formulation of the TSP is trivial: the state space S consists of the set of $n!$ permutations of the n cities, and the objective function F simply maps a permutation π to the total tour distance $\Sigma_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_n, c_1)$ (in this formulation, some elements of S are redundant, as there are only $(n-1)!/2$ unique tours).

Local search algorithms can also be applied to NP-complete decision problems. However, in contrast to the TSP, these COP formulations are often somewhat forced, as

the notion of solution quality is not immediately relevant in a decision problem. For example, in the Boolean Satisfiability Problem (SAT) we are given a formula Φ in propositional logic containing n variables and a conjunction of m clauses. A clause is a disjunction of some number of literals, each of which is either one of the n Boolean variables or their negation. The problem is to determine whether there exists a set of assignments to the n Boolean variables such that each of the m clauses is satisfied, i.e., at least one literal in each clause is true. To formulate SAT as a COP, we define the state space S as the set of all 2^n possible assignments to the set of n Boolean variables. To define the objective function, we let F map a variable assignment s to $\sum_{i=1}^{m} \mathit{satisfied}(i, \Phi, s)$, where $\mathit{satisfied}(i, \Phi, s)$ returns 1 if the ith clause in Φ is satisfied under s, and 0 otherwise. The resulting formulation is known as MAX-SAT, which enables local search algorithms to solve certain instances (specifically, satisfiable instances) of the SAT decision problem. If a local search algorithm is guaranteed to locate an optimal solution s, and $F(s) = m$, then the problem is satisfiable; otherwise, it is not satisfiable. However, as we discuss later in Section 3.4, the behavioral properties of local search severely limit the practical applicability of local search to SAT and other decision problems.

Numeric parameter optimization problems can also be formulated as COPs for solution by local search algorithms. Pervasive in engineering fields, the objective is to find a minimal or maximal value of some high-dimensional continuous function. To formulate these problems as COPs, we simply discretize each input parameter. The objective function is then identical to the original function, with the exception that the domain of the function is restricted to the set of values allowable under the chosen discretization.

3.2 Local Search and Combinatorial Optimization

Local search proceeds via iterative modifications to complete solutions, in contrast to more traditional constructive optimization algorithms (e.g., branch-and-bound) that incrementally extend partial solutions into complete solutions. We restrict our attention to the sub-set of local search algorithms that operate via iterative modifications to a *single* complete solution s. Well-known examples of such single-solution local search algorithms are simulated annealing, tabu search, and iterated local search. A significant number of local search algorithms maintain a set or population of solutions, including genetic algorithms [Mit98], optima linking [RW97], and path relinking [GL97]. However, because the behavior of single-solution local search algorithms is poorly understood, it is only pragmatic to develop a full understanding of the simplest (albeit still powerful) class of local search algorithms before tackling more complex derivatives.

All single-solution local search algorithms consist of the following four core components: the state space, the objective function, the move operator, and the navigation strategy. Search begins from an initial solution $s = s_{\mathit{init}}$ that is generated either at random or via some heuristic procedure, and proceeds via a sequence of iterations. The move operator specifies the set of allowable modifications to the current solution s at

any given iteration, one of which is selected by the navigation strategy to serve as the basis for the next iteration. The best solution encountered in any iteration is stored and returned when the algorithm terminates. We now explore each of these four components in more detail, providing simple examples of each as applied to both the TSP and MAX-SAT.

3.2.1 The State Space and the Objective Function

Both the state space S and the objective function F are taken directly from Ω, the problem instance under consideration. For example, in an n-city TSP instance, the state space consists of the set of $n!$ permutations, each representing a possible tour; the objective function simply returns the total length of the input tour. In an n-variable, m-clause MAX-SAT instance, the state space consists of the set of 2^n Boolean vectors of length n; the objective function returns the number of the m clauses that are satisfied. In general, the details of how solutions are represented can impact the definition of both the move operator and the navigation strategy. However, this is not the case for many well-known combinatorial optimization problems – including MAX-SAT, the TSP, and the JSP. Consequently, details of the representation can be ignored when discussing a particular local search algorithm, although those details can have a major impact on implementation efficiency.

3.2.2 The Move Operator

Given a state space S, the notion of locality in a local search algorithm is provided by the move or neighborhood operator N. N defines the set of allowable modifications to the current solution s in any given iteration. In single-solution local search algorithms, $N : S \rightarrow P(S)$, where $P(S)$ denotes the power set of S. More complicated move operators, e.g., those whose domain and/or codomain are cross-products of S and $P(S)$, respectively, are found primarily in evolutionary algorithms and other related approaches such as optima linking, path relinking, and scatter search. Given a solution s, the set $N(s)$ is known as the *neighborhood* of s. Similarly, if $s' \in N(s)$, then s' is said to be a *neighbor* of s.

Local search algorithms often employ rather simple move operators. For example, the most widely used move operator for MAX-SAT maps each solution $s \in S$ to the subset of n solutions (where n is the total number of Boolean variables) in S that differ from s in the value of a single variable assignment; this is known as the '1-flip' neighborhood. Similarly, most local search algorithms for the TSP are based in part on the 2-opt move operator [LK73], where the neighbors of a solution $s \in S$ are defined as the subset of $\binom{n}{2}$ solutions in S obtained by inverting the sub-tour between any pair of distinct cities on the tour specified by s. More complex move operators can be obtained via straightforward generalization of these basic operators, e.g., k-flip move operators in MAX-SAT and k-opt move operators in the TSP.

Move operators can vary significantly in their attempts to maintain logical locality. Both the 1-flip and 2-opt move operators induce the minimal possible change to the current solution s: 1-flip inverts the value of a single Boolean variable, while 2-opt changes exactly 2 edges. However, in local search algorithms such as iterated local search, the differences between neighboring solutions can be much more substantial, e.g., under the k-opt move operator for the TSP [JM97]. In both cases, however, the perturbation is local in the sense that a neighboring solution is obtained via a *single* application of a move operator. We raise this issue to note that a 'local' search algorithm may in fact proceed by making drastic modifications to individual solutions.

Mathematically, the move operator N induces a relation on the space $S \times S$, and the properties of this relation can influence the performance of local search. For simplicity, we refer to the relation induced by N simply as N. Although both the 1-flip and 2-opt move operators are symmetric, in that $s' \in N(s) \Rightarrow s \in N(s')$, this is generally not required (and is not true in the case of the JSP – see Section 3.5.2). Further, N is necessarily transitive and anti-reflexive. Beyond defining the immediate neighborhood, a move operator also defines the connectivity of the search space, i.e., what solutions can be reached via a finite sequence of moves from an initial solution. A move operator N is said to induce a *connected* search space if from an arbitrary solution there always exists a sequence of moves to an optimal solution. N is said to induce a *fully connected* search space if there exists a sequence of moves between any two arbitrary solutions. Both the 1-flip and 2-opt move operators induce fully connected, and consequently connected, search spaces. However, many powerful, problem-specific move operators induce disconnected search spaces.

3.2.3 The Navigation Strategy

The mechanism for selecting some neighbor $s' \in N(s)$ at each iteration of local search is embodied in the navigation strategy, which we denote by Δ. One of the simplest navigation strategies follows the basic principle of gravity: select a neighbor $s' \in N(s)$ with $F(s') < F(s)$. Two well-known variants of this greedy strategy form the core of all most navigation strategies. In *next-descent* search, the neighbors $N(s)$ are randomly ordered, and the first neighbor $s' \in N(s)$ such that $F(s') < F(s)$ is selected. In *steepest-descent* search, the neighbor that provides the maximal decrease in the objective function value (*argmin*$_{s' \in N(s)} F(s')$) is selected, with ties broken randomly.

By iterating greedy descent, local search will eventually arrive at a solution $s \in S$ from which no immediate improvement in the value of the objective function is possible; s is known as a local optimum, such that $\forall s' \in N(s), F(s') \geq F(s)$. Unless s is also a globally optimal solution, the navigation strategy must then guide search to unexplored regions of the search space. When there exists a neighbor $s' \in N(s)$ such that $F(s) = F(s')$, s is actually contained in a plateau, which may or may not be locally optima; this issue is discussed further in Section 5.2.

Within the local search community, strategies for escaping or avoiding local optima

are commonly referred to as *meta-heuristics*. Formally, a meta-heuristic is a heuristic that dynamically alters the behavior of the core local search heuristics, typically in response to the properties of recently visited solutions. In most local search algorithms, the core heuristic is greedy descent; a meta-heuristic is activated when the descent strategy becomes trapped in a local optimum, and deactivated once search is directed toward new regions of the search space. Conceptually, meta-heuristics and greedy descent are distinct forms of navigation strategies, each operating at a different level of abstraction. However, in practice, the boundary between the two is often fuzzy, for example in simulated annealing. Consequently, meta-heuristics are often viewed as atomic entities, such that the distinction between the core heuristic and the meta heuristic is ignored. We present them as such, while at the same time acknowledging any intended distinction between the core and meta heuristic. Finally, we note that a local search algorithm is in practice synonymous with the meta-heuristic it employs. For example, tabu search frequently refers to a complete local search algorithm, with the remaining components implicitly defined.

Perhaps the most obvious way to escape a local optimum is to generate a new starting solution s_{init} and then re-initiate greedy descent. This process can be iterated until a global optimum is located. The resulting meta-heuristic is commonly referred to as *iterated descent*, which is distinct from the next-descent and steepest-descent procedures. In practice, iterated descent is a simple way to improve the performance of the core greedy descent strategies. Further, iterated descent can locate very high-quality solutions for some combinatorial optimization problems, e.g., see Beveridge et al. [BGS97].

Clearly, the probability of iterated descent locating a global optimum approaches 1 as the number of greedy descents N approaches ∞. However, from a practical standpoint, iterated descent is only effective if the fitness distribution of the local optima assumes a certain form, i.e., one in which the left tail of the distribution is non-negligible. For many well-known combinatorial optimization problems, the fitness distribution of local optima in small problem instances satisfies this requirement. At the same time, it has been empirically demonstrated that such tails typically vanish at larger problem sizes (for example in the TSP), causing iterated descent to perform poorly due to what has come to be known as a "central limit catastrophe" [JM97].

More established and recognized meta-heuristics include simulated annealing (and the related Metropolis sampling procedure), tabu search, and iterated local search. These "big 3" are by no means an exhaustive list of the meta-heuristics available for developing single-solution local search algorithms. Nor do these meta-heuristics necessarily represent the state-of-the-art for a particular combinatorial optimization problem, although this is often the case. Rather, these meta-heuristics are widely applied and well-studied, each providing good, if not exceptional, performance on a wide range of problems, including the JSP. For these reasons, we restrict our attention to this sub-set of meta-heuristics. The general concepts underlying these meta-heuristics are discussed in Chapters 7 through 9, in addition to specific variations as applied to the JSP. With the notable exception of guided local search [BV98], competitive local search algorithms

for the JSP are based either entirely or in part on one of these three meta-heuristics [JM99]. Popular meta-heuristics that are *(*not) currently considered competitive on the JSP are evolutionary algorithms and ant colony optimizers.

Analysis of meta-heuristics is relatively rare. Yet, analysis is critical in making targeted improvements to existing meta-heuristics. Often, the motivation for a new or modified meta-heuristic is ad-hoc, based on some very general observation or intuition. But, as we discuss in Chapter 5, because the factors that influence the performance of existing meta-heuristics are poorly understood, ad-hoc modifications may actually do little to rectify "known" deficiencies. Further, researchers rarely prove, a-posteriori, that the proposed modification is directly responsible for any observed differences in performance. By analyzing well-known, existing meta-heuristics, we hope to establish the deficiencies and benefits of these algorithms, and propose more principled methods for developing new meta-heuristics.

3.3 The Fitness Landscape

Given a local search algorithm A and a combinatorial optimization problem Π, we are interested in determining what makes a particular instance $\Omega \in U_{\Pi}$ easy or difficult for A. Problem difficulty, or equivalently search cost, is dictated by the interaction of A with the underlying search space. For example, suppose all globally optimal solutions to Ω reside in a small region of the search space containing otherwise poor local optima. If A consistently biases search toward regions of the search space containing generally high-quality local optima, then the cost (on average) of locating optimal solutions to Ω using A is likely to be large. In contrast, if A intensifies search in regions of the search space with poor local optima, then A is more likely to locate optimal solutions to Ω in shorter run-times.

Due to the central role of the search space in determining problem difficulty, much of the research on models of problem difficulty has concentrated identifying structural features of the search space that are likely to influence the cost of local search. Given a local search algorithm A, the search space is defined by the combination of (1) the state space S, (2) the move operator N, and (3) the objective function F. Formally, we define the search space $L = (S, N, F)$ as a vertex-weighted directed graph $G = (V, E)$ in which:

1. $V = S$
2. $\forall v \in V$, the weight w_v of v is equal to $F(v)$
3. $E = \{(i, j) | i \neq j \wedge \exists i, j : i \in N(j)\}$

Within the local search community, the graph G is known as a *fitness landscape*, a concept first introduced by the theoretical biologist Sewall Wright in 1932 [Wri32].

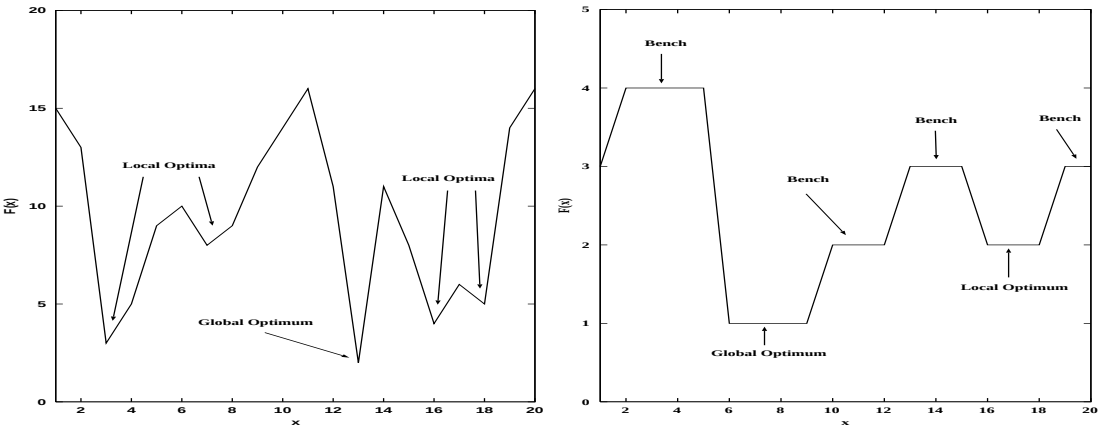

Figure 3.1: Examples of Type I (left figure) and Type II (right figure) fitness landscapes.

We provide two examples of very simple fitness landscapes in Figure 3.1; in general, landscapes are high-dimensional and extremely difficult to visualize. In both examples, $V = \{1, 2, ..., 20\}$ and $N(x) = \{x - 1, x + 1\}$, subject to the boundary conditions $N(1) = \{20, 2\}$ and $N(20) = \{19, 1\}$. ***Type I*** fitness landscapes are characterized by deep, punctuated valleys with abrupt changes in the fitness of neighboring solutions. In contrast, ***Type II*** fitness landscapes are dominated by plateaus of equally fit neighboring solutions, with discrete jumps in fitness between the plateaus. We differentiate between the two types of fitness landscapes for three reasons. First, different terminology is associated with the two landscape types. Second, Type I and Type II landscapes have different implications for the design of navigation strategies. Third, these two types are representative of the fitness landscapes found in most NP-hard optimization problems. For example, the TSP and MAX-SAT respectively possess Type I and Type II fitness landscapes.

In a Type I fitness landscape, the two key features of interest are local optima and global optima. A *local optimum* is a point $x \in S$ such that $\forall y \in N(x)$, $F(x) \leq F(y)$. In our example Type I landscape, the following vertices are local optima: 3, 7, 13, 16, and 18. A *global optimum* is a point $x \in S$ that is both locally optimal and $\forall y \in S$, $F(x) \leq F(y)$. In our example Type I landscape, vertex 13 is the sole global optimum. The *attractor basin* of a local optimum s consists of all $s' \in S$ such that s results with non-zero probability when a descent-based procedure is applied to s'; as first noted by Reeves [Ree98], attractor basin membership may be stochastic due to the different forms of randomization commonly found in descent procedures.

Plateaus are the dominant feature of Type II fitness landscapes. Informally, a plateau is simply an interconnected region of the fitness landscape where all points have equal fitness. Formally, a plateau is defined as a set $P \subseteq S$ such that:

1. $\forall x \in P$, $F(x) = C$ for some constant C

2. For any two points $x, y \in P$ there is a sequence of elements $\{x, a_1, ..., a_n, y\}$ such

that $\forall i$, $a_i \in S$ and $F(x) = F(a_1) = ... = F(a_n) = F(y) = C$

3. (a) $a_1 \in N(x)$, (b) $\forall i \neq n-1, a_{i+1} \in N(a_i)$, and (c) $y \in N(a_n)$

If for some $x \in P$ there exists a $y \in N(x)$ such that $F(y) < C$, the plateau is called a *bench*, and all such solutions y are called *exits*. If there are no exits from a plateau, then the plateau is a local optima. If the plateau is locally optimal and $\forall x \in S, C \leq F(x)$, then the plateau is also globally optima. All benches, local optima, and global optima are labeled in our example Type II fitness landscape. There are many additional nuances regarding the terminology of features found in Type II fitness landscapes; an overview can be found in Frank et al. [FCS97].

The qualitative differences between Type I and Type II fitness landscapes have an important impact on the design of navigation strategies and meta-heuristics for local search. For example, both next-descent and steepest-descent typically terminate once a solution s is located with no lower-fitness neighbors. The implicit, built-in assumption is that the local optimum s is not a member of a plateau, or that if s is a member of a plateau, then the plateau itself is locally optimal. In general, these assumptions do not hold when dealing with Type II fitness landscapes; if greedy descent terminates at a local optimum, it is possible that the optimum resides on a bench, from which an exit may exist. Additionally, the attractor basins in Type II fitness landscapes are often very shallow. For example, Frank et al. [FCS97] have shown that in MAX-SAT, it is often possible to escape a local optimum by accepting a single non-improving move. Consequently, the emphasis on navigation strategies in Type II fitness landscapes is on moving quickly from one plateau to another, either by finding an exit from a bench, or by accepting non-improving moves. In contrast, in Type I fitness landscapes the emphasis is on escaping local optima with potentially large and deep attractor basins (although this emphasis can be misguided, as we show in Chapter 5).

3.4 Convergence Properties of Local Search Algorithms

A defining characteristic of many constructive optimization algorithms is *completeness*: upon termination, an optimal solution is always returned. Completeness is derived from the fact that an algorithm systematically explores the entire search space, although the enumeration is typically implicit, e.g., as occurs in branch-and-bound. Only through such systematic exploration can an algorithm both locate an optimal solution *and* prove the optimality of the solution. In contrast, local search algorithms generally do not perform a systematic exploration of the fitness landscape, and are therefore *incomplete*: although they can locate optimal solutions, they lack the ability to prove the optimality of these solutions. Several researchers have noted that it is at least possible to specify a complete local search algorithm. For example, any single-solution local search algorithm can be made trivially complete by systematically re-starting the algorithm from all possible solutions [Hoo98] – at which point the local search aspect of the algorithm

is largely irrelevant. To date, all practical implementations of local search algorithms are incomplete.

In Chapters 6 through 9, the difficulty of a problem instance for local search is equated with the cost, in terms of the total number of iterations, required to locate an optimal solution. However, because local search algorithms are incomplete, there exists the possibility that the proposed measure is ill-defined, i.e., a global optimum may never be encountered. Although *strictly* incomplete, there are many local search algorithms that will eventually locate an optimal solution *given a sufficiently large run-time*. This observation has led researchers to acknowledge the ability of some local search algorithms to act in some sense as if they are complete.

With very few exceptions, local search algorithms are inherently stochastic. Suppose that a stochastic local search algorithm A is executed on a problem instance Ω of size K, and requires a CPU time of $RT_{A,\Omega,K}$ to locate an optimal solution; $RT_{A,\Omega,K} = \infty$ if an optimal solution is never located. Because A is stochastic, $RT_{A,\Omega,K}$ is a random variable. Given this framework, we follow Hoos [Hoo98] in extending the traditional notion of completeness by classifying the behavior of a stochastic algorithm A (and by inference a local search algorithm A) on problem instances Ω of size K into one of the following three categories:

- **Complete**: A stochastic algorithm A is *complete* if for each problem size K there exists a $T_{max,K}$ such that for all problem instances Ω of size K, $P(RT_{A,\Omega,K} \leq T_{max,K}) = 1$. In other words, for each problem size K, there exists an instance-*independent* but size-dependent CPU bound $T_{max,K}$ such that if A is allowed to run for time $T_{max,K}$, it is guaranteed to locate an optimal solution.

- **Asymptotically Complete**: A stochastic algorithm A is *asymptotically complete* if for each problem instance Ω of size K there exists a $T_{max,\Omega}$ such that $P(RT_{A,\Omega,K} \leq T_{max,\Omega}) = 1$. In other words, there exists an instance-*dependent* CPU bound $T_{max,\Omega}$ such that if A is allowed to run for time $T_{max,\Omega}$, it is guaranteed to locate an optimal solution.

- **Incomplete**: A stochastic algorithm A is *incomplete* if there exists a problem instance Ω of size K for which $\lim_{t\to\infty} P(RT_{A,\Omega,K} \leq t) < 1$. In other words, there exist problem instances for which A will never locate an optimal solution, independent of the allocated CPU time.

No practical implementations of local search algorithms are complete; the only known complete stochastic search algorithms are certain constructive algorithms that employ stochastic backtracking procedures, such as Rapid Randomized Re-Starts [GSK98]. Local search algorithms are therefore either asymptotically complete or incomplete. Two factors are directly responsible for the particular classification: the move operator N and the navigation strategy Δ. A necessary condition for approximate completeness is that the move operator N induce a connected search space. Further, this

condition is *independent* of the navigation strategy, i.e., a navigation strategy can never overcome inherent deficiencies in the move operator. Finally, given a connected search space, the property of asymptotic completeness requires that the navigation strategy must eventually visit every point in the fitness landscape at least once, and avoid becoming trapped in particular regions of the fitness landscape.

3.5 Local Search and the JSP: Core Components

We now describe the set of core components found in all of the local search algorithms for the JSP that we consider in later chapters. Specifically, we detail the state space, representation, objective function, move operators, and initialization method; we defer descriptions of meta-heuristics to those chapters in which we analyze the behavior of specific local search algorithms.

3.5.1 State Space, Representation, and Objective Function

For an $n \times m$ instance Ω of the JSP, all of the local search algorithms we consider restrict search to the sub-space of feasible solutions S_Ω. Solutions are encoded using the "permutation-and-graph" method [NS96], which is a restricted form of the disjunctive graph containing no redundant disjunctive edges. Let $s \in S_\Omega$ be specified via the set of machine processing orders γ_j, $1 \leq j \leq m$. Under the permutation-and-graph method, s is represented as a vertex-weighted directed graph $G(\gamma_1, \ldots, \gamma_m) = (V, C \cup R(\gamma_1, \ldots, \gamma_m))$. The sets V and C are identical to those found in the disjunctive graph, with the exception that vertices and edges involving o_{src} and o_{snk} are omitted. The edge set $R(\gamma_1, \ldots, \gamma_m)$ contains an edge (o_{ij}, o_{jk}) if and only if (1) $x = \pi_i(j) = \pi_j(k)$ and (2) $\gamma_x^{-1}(i) = \gamma_x^{-1}(j) - 1$. In other words, disjunctive edges only exist between adjacent operations in a machine order. Each vertex $o_i \in V$ is weighted by τ_i. In terms of implementation, $G(\gamma_1, \ldots, \gamma_m)$ can be encoded using two vectors of length mn, respectively tracking the machine predecessor and successor operations (if they exist) of each operation o_i, $1 \leq i \leq mn$; the job successor and predecessor operations need not be stored, as they can be computed directly from the operation indices. Additionally, we maintain pointers to the first and last operation to be processed on each machine. It should be clear that the following can be easily extracted from the resulting encoding: (1) the set of machine processing orders γ_i, (2) the machine predecessor and successor operations MP_{ij} and MS_{ij} of any operation o_{ij}.

The objective function $F(s)$ simply returns the longest vertex-weighted path in $G(\gamma_1(s), \ldots, \gamma_m(s))$, i.e., the makespan $C_{max}(s)$. To compute $C_{max}(s)$, we first compute the set of operation earliest start times using the algorithm shown in Figure 3.2, due to Taillard [Tai94]. Let o denote the start operation. If $o = *$, then all of the est_{ij} will be recomputed. If $o \neq *$, then only the est_{ij} of operations o_{ij} "downstream" from o will be re-computed. Recall that $est_{ij} = max(est_{MP_{ij}} + \tau_{MP_{ij}}, est_{JP_{ij}} + \tau_{JP_{ij}})$

```
function computeESTs (o)
        Q = ∅
        ∀i, marked(o_i) = false
        if (o = ∗) then
                Q = {o_i | MP_i = JP_i = ∗}
        else
                Q = {o}
        end
        while Q ≠ ∅ do
                select an arbitrary o_i ∈ Q
                marked(o_i) = true
                Q = Q \ {o_i}
                compute est_i
                if JS_i ≠ ∗ then
                        pred = MP_JS_i
                        if pred ≠ ∗ or marked(pred) then
                                Q = Q ∪ {JS_i}
                        end
                end
                if MS_i ≠ ∗ then
                        pred = JP_MS_i
                        if pred ≠ ∗ or marked(pred) then
                                Q = Q ∪ {MS_i}
                        end
                end
        end
end
```

Figure 3.2: Pseudo-code for computing the set of operation earliest start times. See text for details.

for any o_{ij}; by convention, $est_* = \tau_* = 0$. The makespan C_{max} is then given by $max(ect_{1m}, ect_{2m}, \ldots, ect_{nm})$, with $ect_{ij} = est_{ij} + \tau_{ij}$. If necessary (see Section 3.5.2), the set of operation tte_{ij} can be computed using an analogous algorithm that propagates start-times from the rear of the schedule to the front. Both computeESTs and the algorithm for computing the tte_{ij} are specializations of well-known algorithms for constructing a topological sort of the vertices in a directed graph, and typically consume over 95% of the run-time in local search algorithms for the JSP.

3.5.2 Move Operators

Through purely syntactic manipulation of the machine processing orders, one can easily define a number of problem-independent move operators for the JSP. However, in contrast to other combinatorial optimization problems (e.g., MAX-SAT), local search algorithms for the JSP based on problem-independent move operators generally provide very weak performance. The reasons are three-fold. First, the makespan of the current solution cannot be reduced unless the move operator re-orders some sequence or subsequence of critical operations [vLAL88]. By focusing on arbitrary operations, many of the moves generated by problem-independent operators are provably non-improving. Second, random manipulation of machine processing orders generally leads to infeasible solutions, forcing problem-independent move operators to perform computationally expensive cycle checks. Third, the size of the neighborhood under problem-independent move operators is generally quite large, leading to high-cost evaluation of the full neighborhood.

The explosion of interest in local search algorithms for the JSP largely stems from the development of powerful, problem-specific move operators, which use a deeper structural knowledge of the JSP to focus search on the subset of moves that have the potential to both decrease the makespan of the current solution and when possible avoid infeasible solutions. All of the local search algorithms we consider are based on two well-known problem-specific move operators for the JSP, which we denote $N1$ and $N5$; our notation is borrowed from Blażewicz et al. [BDP96].

3.5.2.0.1 The $N1$ Move Operator The canonical problem-specific move operator for the JSP was introduced by van Laarhoven et al. in 1992 [vLAL92] and is commonly referred to as the $N1$ operator. The neighborhood of a solution s under the $N1$ operator consists of the set of solutions generated by inverting the order of a pair of adjacent operations on the same critical block in s, i.e., two operations o_{ij} and o_{kl} such that (1) $ect_{ij} = est_{kl}$, (2) $\pi_i(j) = \pi_k(l)$, and (3) $\gamma_x^{-1}(i) = \gamma_x^{-1}(k)$ where $x = \pi_i(j) = \pi_k(l)$. By focusing strictly on critical operations, $N1$ avoids a large subset of the moves that cannot immediately reduce the makespan of the input solution s. Further, all neighbors of s under $N1$ are feasible [vLAL92], eliminating the need for cycle checking. van Laarhoven et al. also prove that the search space under $N1$ is connected: from an arbitrary (feasible) solution, there exists a sequence of moves that leads to some global optimum. Consequently, any local search algorithm based on the $N1$ operator at least has the *potential* to exhibit asymptotically complete behavior. In contrast to move operators for many well-known combinatorial optimization problems, the $N1$ operator is *asymmetric* [Kol99]: $s' \in N1(s) \not\Rightarrow s \in N1(s')$. Nowicki and Smutnicki [NS96] show that if $N1(s) = \emptyset$, then $C_{max}(s) = C^*_{max}$. In other words, if no move is possible from s, which can occur if each critical block consists of a single operation, then s is optimal.

The neighborhood under $N1$ can be defined relative to either a single (typically random) critical path, or all critical paths. As indicated in Section 2.8, re-sequencing the

order of operations on a single critical path is *not* guaranteed to reduce the makespan, although this is commonly assumed. In our use of $N1$, we follow Taillard [Tai94] in inverting the order of critical operations appearing on *any* critical path.

Given a current solution s with the associated est_i and tte_i, the makespan $C_{max}(s')$ of neighboring solutions $s' \in N1(s)$ can either be computed exactly or it can be estimated. Suppose s' is obtained by inverting the order of two critical operations o_k and o_l; by convention, assume that o_k appears before o_l in the machine processing order. To compute $C_{max}(s')$ exactly, we first construct the machine processing orders of s', subsequently invoking computeESTs(o_l) and computeTTEs(o_k) to respectively update the est_i and tte_i of operations in s'. Note that only the est_i of operations downstream (and including) o_l and the tte_i of operations upstream (and including) o_k can be altered by inverting the order of o_k and o_l. In contrast, estimation of $C_{max}(s')$ relies only on the est_i and tte_i of the current solution s. Following Taillard [Tai94], we define the following:

- $est'_l = max(est_{MP_k} + \tau_{MP_k}, est_{JP_l} + \tau_{JP_l})$
- $est'_k = max(est'_l + \tau_l, est_{JP_k} + \tau_{JP_k})$
- $tte'_k = max(tte_{MS'_l}, tte_{JS'_k}) + \tau_k$
- $tte'_l = max(tte'_k, tte_{JS_l}) + \tau_l$

The estimated makespan $C_{max}(s')$ is then given by $max(est'_l + tte'_l, est'_k + tte'_k)$.

In Chapters 6 through 9, we develop cost models for various local search algorithms based on the $N1$ operator. Although no longer considered state-of-the-art, the connectivity property of the $N1$ operator enables these local search algorithms to exhibit asymptotic completeness, a property that greatly simplifies the development of our cost models. The *distance* between two solutions also plays a major role in our cost models; ideally, such a distance should be expressed in terms of the underlying move operator. Although computing the exact number of moves under a given move operator to transform solution s_1 into solution s_2 is NP-hard [Vae95], the disjunctive graph distance $D(s_1, s_2)$ does provide a lower bound on the number of moves under $N1$.

3.5.2.0.2 The $N5$ Move Operator Since its introduction, researchers have identified several deficiencies of the $N1$ operator. Specifically, there exist many pairs of adjacent critical operations for which swapping their relative order cannot improve the makespan of the current solution. Based on these observations, more constrained variations of the $N1$ operator have been developed. Here, the design goal is to eliminate provably poor moves while retaining the ability to locate improving solutions [JRM00]. The key observations were made by Matsuo et al. [MSS88] and Nowicki and Smutnicki [NS96]. Matsuo et al. proved that the makespan of the current solution can only be improved by swapping adjacent pairs of operations that include either the first or last operations of a critical block. Nowicki and Smutnicki further showed that swapping either the first two operations on the first critical block or the last two operations on the last critical block

can never improve the makespan of the current solution. Following Blazewicz et al. [BDP96], we refer to the move operator enforcing both restrictions as the $N5$ operator.

The $N5$ operator is noteworthy because it is an integral component of state-of-the-art local search algorithms for the JSP [NS96] [NS01][NS02][NS03]. Relative to $N1$, the neighborhood size under $N5$ is very constrained. Further, the fitness landscape under $N5$ is disconnected, such that local search algorithms based on $N5$ can never be asymptotically complete. Yet, counter-intuitively, this drawback has not impeded the performance of local search algorithms based on $N5$.

In Chapters 8 and 9, we introduce several variations of local search algorithms based on the $N5$ operator. Following Nowicki and Smutnicki [NS96], we restrict the pairs of adjacent critical operations to those appearing on a *single* critical path. Experimental results reported by Jain et al. [JRM00] indicate that the performance of local search algorithms based on $N5$ is independent of the details of the method for selecting a critical path (even if all critical paths are identified). Consequently, we extract the sequence of operations along a *random* critical path, beginning at an arbitrary operation o_y with $ect_y = C_{max}(s)$ and traversing backward along critical job/machine predecessors until an operation o_x is encountered with $est_x = 0$. Under this methodology, we only need to maintain the set est_i of operation earliest start times. Nowicki and Smutnicki [NS01] indicate that estimation of neighboring solution makespans can detrimentally impact the performance of tabu search algorithms based on $N5$. Thus, we compute the makespan of neighboring solutions exactly; when swapping a pair of critical operations o_k and o_l, we update the operation est_i by invoking computeESTs(o_l).

3.5.2.0.3 Other Move Operators Another trend in the design of move operators for the JSP is to expand the neighborhood size by swapping either multiple pairs of adjacent critical operations or pairs of non-adjacent critical operations. By increasing the number of neighbors, the goal is to increase the probability of locating an improving solution. Although diametrically opposed to the trend toward more constrained variants of the $N1$ operator, high-performance local search algorithms based on such operators do exist, e.g., the algorithm by Balas and Vazacopoulos [BV95]. Blazewicz et al. [BDP96] provide an excellent overview of move operators for the JSP, including $N1$ and $N5$.

3.5.3 The Initial Solution

We initiate search in all of our algorithms from either random semi-active solutions or random local optima. To construct a random semi-active solution, we use a method introduced by Mattfeld [Mat96]. Given an $n \times m$ problem instance, we first construct a vector v consisting of m '1's, followed by m '2's, ..., terminated by m 'n's. For example, given a 4×3 instance, $v = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]$. We then construct a random permutation of the indices 1 through mn, and re-order v accordingly. To build a solution, v is scanned from indices 1 to mn. Let $x = v[i]$ denote the job located at position i in v. At each iteration i, the next available operation o_{xy} of job x is added to the

end of the machine order $\gamma_{\pi_x(y)}$. Upon termination, the resulting solution is guaranteed to be acyclic. We construct a random local optimum by applying steepest-descent local search to a random semi-active solution; ties between equally good moves are broken randomly.

Researchers have introduced a wide variety of initialization methods for the JSP. We use random solutions as a baseline, as it is unclear what benefit, if any, more complex initialization methods have on the performance of local search. We introduce some of these additional methods and analyze their impact on performance in Section 7.10.3.

3.6 Locating Globally Optimal Solutions

Although the focus in this thesis is on *local* search algorithms for the JSP, complete constructive algorithms are required to identify the optimal makespans of the test problems introduced in Section 4.6. Further, much of our analysis requires knowledge of *all* optimal solutions to a problem instance, which again requires a complete constructive algorithm for the enumeration. Complete, constructive algorithms for the JSP fall into two broad categories: branch-and-bound and constraint programming. Both types of algorithm are equally effective on the relatively small (e.g., 10×10 and smaller) problem instances we consider. For our experiments, we (admittedly arbitrarily) selected a state-of-the-art constraint programming algorithm.

The history of constraint programming (CP) and the JSP is relatively short, although very effective search algorithms have been developed in this time. CP algorithms for the JSP search the space of $2^{(m \cdot \frac{n(n-1)}{2})}$ possible orientations for the $m \cdot \frac{n(n-1)}{2}$ disjunctive edges of D in the disjunctive graph $G = (V, C \cup D)$. CP algorithms are formulated to solve decision problems. Thus, we must first specify a target makespan L, such that the goal is to identify a solution with a makespan $\leq L$. To determine the optimal makespan of a JSP instance, we begin with L equal to the makespan of a random semi-active solution, and iterate the CP algorithm for successively smaller values of L. The process terminates once a makespan K is identified for which no solution exists, such that the optimal makespan C^*_{max} is given by $K + 1$.

Search in CP is constructive. At each step there exists a partial solution s where some set $D' \subseteq D$ of disjunctive edges remain unoriented; each such edge specifies a pair of operations which have yet to be sequenced. CP algorithms for the JSP use variable ordering heuristics to select an edge $x \in D'$ to orient, and use value ordering heuristics to select a particular orientation for x. After selecting an orientation for an undirected edge $x \in D'$, CP algorithms for the JSP perform constraint propagation, which reduces the range of feasible operation start and end times. These ranges can then be used to determine whether a specific orientation is required to produce a feasible solution, or if any orientation leads to an infeasible solution. If there exists an edge $x \in D'$ with no orientation leading to a feasible solution, search backtracks. If there exists an edge $x \in D'$ requiring a specific orientation, that orientation is selected. When

neither case is applicable (both orientations are feasible for all $x \in D'$), the variable and value orderings heuristics are used to select and orient a disjunctive edge. The effectiveness of CP algorithms for the JSP is largely due to the development of very strong, problem-specific constraint propagation methods. These methods are able to substantially reduce the range of feasible operation start and end times, which allows search to quickly identify both forced and infeasible orientations, leading to a potentially dramatic reduction in the size of the search space. A detailed overview of both constraint propagation and variable/value ordering heuristics in the JSP is provided in Beck and Fox [BF00].

The specific CP algorithm we use is also documented in Beck and Fox [BF00], and possesses the following key features:

- The *Sum Height* texture-based variable and value ordering heuristics.
- Limited Discrepancy Search (LDS)[HG95] for backtracking.
- Arc-B-consistency [Lho93] constraint propagation for both precedence and disjunctive constraints.
- Edge-Finding Exclusion[Nui94] constraint propagation for subsets of operations on the same machine.
- Edge-Finding Not-First/Not-Last[Nui94] constraint propagation for subsets of operations on the same machine.

To enumerate the set of optimal solutions to a problem instance, we first compute the optimal makespan using the CP algorithm and then re-initiate search with $L = C^*_{max}$. However, we modify the basic algorithm as follows: whenever a complete solution $s \in S_\Omega$ with $C_{max}(s) = C^*_{max}$ is identified, the solution is recorded and the algorithm backtracks as if an infeasible solution was detected. While the modification is slight, the run-time ramifications are not: enumeration of optimal solutions costs *at least* as much as proving optimality. Finally, we also have the requirement (e.g., see Section 7.6.1) to enumerate the set of all *sub-optimal* solutions to an instance with a makespan between C^*_{max} and $C^*_{max} + X$, which is accomplished simply by letting $L = C^*_{max} + X$.

Chapter 4

Developing and Validating Cost Models of Local Search: Methodological Issues

Most research on local search focuses on developing newer, better-performing algorithms. The goal in such research is to *demonstrate* algorithm performance. Paul Cohen notes in his book *Empirical Methods for Artificial Intelligence* ([Coh95], p. 249) that "It is good to demonstrate performance, but it is even better to *explain* [emphasis added] performance." The hard sciences advance via the development of accurate models of the object or objects of interest, models that are both consistent with existing observations and suggest new behavioral hypotheses. Currently, models of local search algorithms for *any* combinatorial optimization problem are rare to non-existent.

In developing a model of a given object, we generally concentrate on capturing specific behaviors or small sets of behaviors. In the context of local search algorithms for combinatorial optimization, the behavior of interest is generally the cost required by an algorithm to locate an optimal solution to a problem instance. Due to the stochastic nature of local search, search cost is a random variable with a particular distribution. *Cost models* of local search are behavioral models that capture various aspects of the cost distribution. Most often, we focus on the *average* or typical search cost, as defined by either the distribution mean or median. As we show in Chapters 6–10, average cost can vary by as much as *eight* orders of magnitude for sets of even relatively small JSPs. One objective in developing cost models is to account for a significant proportion, and ideally all, of this variability. A more aggressive, penultimate objective is to develop cost models that account for the entire distribution of search cost.

In this chapter, we discuss our general approach to developing the cost models of local search algorithms introduced in Chapters 6 through 9. We investigate three different types of cost models, differing in both the type of information upon which they are based and the extent to which they attempt to capture algorithm run-time dynamics. *Static* cost models, described in Section 4.1, are functions of one or more features of the fitness landscape, and only implicitly consider algorithm dynamics. In contrast, *quasi-dynamic* and *dynamic* cost models are based on analyses of algorithm run-time

behavior. Quasi-dynamic cost models, described in Section 4.2, are functions of simple summary statistics of algorithm behavior. Dynamic cost models, discussed in Section 4.3, explicitly model low-level algorithm behavior using Markov chains. In addition to defining the primary characteristics of each model type, we also discuss what has been already been achieved for other combinatorial optimization problems, and what constitutes reasonable expectations. Despite the behavioral insights that cost models can provide, their power is also fundamentally limited; we investigate this issue in Section 4.4. Although our goal is to account for the observed variability in average search cost, our dynamic cost models can potentially account for the full distribution of search cost, via the methodology discussed in Section 4.5. We conclude in Section 4.6 by introducing the various test problems that we use to develop, test, and validate our cost models.

4.1 Static Cost Models

Static cost models are strictly based on fitness landscape features; algorithm dynamics are completely and explicitly ignored. In a static cost model, the independent variables are fitness landscapes features, or combinations thereof, and the dependent variable is the mean or median search cost. To facilitate model evaluation, static cost models are expressed as linear or multiple regression models. Under this formulation, the accuracy of a static cost model can be naturally quantified as the r^2 value of the corresponding regression model, i.e., the proportion of the total variability accounted for by the model. Most of the static cost models we consider are based on a single feature of the fitness landscape. For purposes of brevity, we often denote a static cost model based on the feature X as the X static cost model, or simply the X model. Similarly, given the close relationship between static cost models and regression models, we frequently use the two terms interchangeably. Finally, regression methods make certain assumptions (e.g., model errors are homogeneous across the range of the independent variable) in order to generate valid statistical inferences concerning model parameters. These assumptions are generally not satisfied in our research. Our motivation in using regression models is to (1) quantify overall model accuracy using the associated r^2 value and (2) analyze worst-case deviations from a predicted/expected value. Failure to satisfy regression assumptions does not impact our ability to achieve either of these objectives.

The quality of a static cost model is tied to the model r^2: models with larger r^2 values are more accurate. However, there are limits on the absolute level of accuracy that we can reasonably expect to achieve. As discussed in Chapter 5, the most accurate static cost models of local search for other NP-hard problems only yield $r^2 \approx 0.5$ in the worst case, which is typically observed for the most difficult sets of problem instances. Although failure to develop more accurate static cost models, despite intense research effort, is not evidence for their impossibility, there does appear to be a practical limit on what can be achieved. Because static cost models ignore algorithm dynamics, the

existence of models with even $r^2 \approx 0.5$ is in some sense surprising. In expressing fitness landscape features as atomic numeric quantities, there is also the obvious potential for loss of information. Further, there are practical (although not theoretical) limits on the accuracy with which we can measure various quantities, including search cost.

4.2 Quasi-Dynamic Cost Models

A first-order approach to improving static cost models is to incorporate coarse-grained information concerning algorithm run-time behavior. For example, we might track simple summary statistics that capture defining characteristics of the set of solutions generated by an algorithm. Given such summary statistics, we can then construct regression models relating these summary statistics to search cost, and quantify model accuracy as the resulting r^2. We refer to such cost models as *quasi-dynamic* cost models. The "quasi-dynamic" modifier derives from the fact that the model is based on aggregate statistics relating to run-time behavior, as opposed to an explicit model of algorithm run-time dynamics. The sole difference between static and quasi-dynamic cost models is in the nature of the information captured in the independent variable(s).

Most of the issues relating to possible limitations on the accuracy of static cost models equally apply to quasi-dynamic cost models. However, because they account for some aspects of run-time behavior, we would expect in some sense the accuracy of quasi-dynamic cost models to be higher than that of static cost models, although less than the fully dynamic cost models considered below. The most accurate cost models of local search algorithms developed to date are quasi-dynamic [SGS00], and achieve a worst-case accuracy of $r^2 \approx 0.65$.

4.3 Dynamic Cost Models

Because they are respectively based on fitness landscape features and summary statistics of run-time behavior, static and quasi-dynamic cost models yield no *direct* insight into the dynamical behavior of local search. To gain insight as to why particular landscape or run-time features are highly correlated with search cost, we turn to *dynamic* cost models. Dynamic cost models are high-resolution models of the run-time behavior of local search algorithms. Recently, Hoos [Hoo02] used dynamic models similar to those developed in Chapters 6- 9 to posit an explanation for certain run-time behaviors observed for Walk-SAT and other local search algorithms in the MAX-SAT phase transition region. However, the ability of these models to account for variability in problem difficulty was not considered.

The dynamic cost models we develop are instances of a Markov chain. Each state of the Markov chain captures the distance i to the nearest *target* solution, in addition to other algorithm-specific attributes. Unless otherwise noted, the target solutions are optimal solutions. Transitions in the Markov chain correspond to iterations of the local

search algorithm. A dynamic cost model is constructed by specifying a set of states, and then estimating the various transition probabilities between the states. The details of the estimation process are algorithm-dependent, and are discussed in Chapters 6–9. The search cost predicted by a dynamic cost model is defined as the mean number of iterations until an absorbing state (i.e., a state with $i = 0$) is encountered. For some Markov chains, analytic formulas for the mean time-to-absorption are easily derived. When analytic formulas are not immediately available, we resort to simulation of the cost model: search cost is defined as the mean number of iterations (typically out of 10,000 independent trials) required to reach an absorbing state.

To quantify the accuracy of a dynamic cost model, we use straightforward linear regression models, in which the predicted and actual search costs serve as the independent and dependent variables, respectively; model accuracy is then quantified by the r^2 value of the linear model. Dynamic cost models differ from their static counterparts in that they explicitly consider the meta-heuristic, and move beyond simple numeric characterizations of either fitness landscape features or run-time behavior. Consequently, we *a priori* anticipate higher levels of accuracy than are possible for static and quasi-dynamic cost models. This conjecture is verified in our analyses: the *worst-case* r^2 for any of our dynamic cost models is 0.96. However, the near-perfect accuracy does not come without costs: dynamic cost models are generally more expensive to construct than static or quasi-dynamic cost models, and are generally far less intuitive.

4.4 Descriptive Versus Predictive Cost Models

For all practical purposes, the cost models we develop are purely descriptive, in that they provide *a posteriori* explanations for why one problem instance is more difficult than another for local search. In principle, our models could be used to compute a relatively tight confidence interval, via standard regression techniques, for the expected cost required to locate an optimal solution to a new problem instance. However, because the most accurate models are functions of the set of *all* optimal solutions to a problem instance, the effort required to generate the prediction actually exceeds that of simply locating an optimal solution. Given an accurate cost model, the problem of run-time prediction is essentially equivalent to the problem of estimating the value of the model parameters. The nature of the cost-accuracy trade-off in model parameter estimation is currently an open research question.

This does *not* imply that cost models are a scientific curiosity, useless in practice. Our cost models *do* make specific predictions regarding the behavior of local search algorithms (e.g., see Chapter 11), which we confirm using empirical testing. Further, and perhaps most importantly, our models explicitly identify those features of the fitness landscape that are overwhelmingly responsible for problem difficulty in local search. By identifying such features, we are enabling algorithm designers to focus on the areas most likely to yield performance improvements, and to move beyond the ad-hoc, benchmark-

driven design methodology that is current employed [Hoo95].

4.5 Run-Length Distributions

We follow Hoos [Hoo98] in referring to the full distribution of search cost as the *run-length distribution*, or RLD. Ideally, we want to account for the full RLD, and not just variability in the average search cost. Static and quasi-static cost models are structured specifically to account for variability in average search cost, and cannot be immediately extended to model the full RLD. In contrast, dynamic cost models, at least in theory, can account for the full behavioral range of an algorithm. Consequently, it is relatively straightforward via simulation to generate a predicted RLD from a dynamic cost model.

For a given problem instance, we can sample both the predicted and actual search costs from a variety of initial solutions, and from this data generate empirical RLDs in the form of cumulative density functions, or CDFs. We are then interested in testing two specific hypotheses: (1) What is the distribution underlying a particular RLD? and (2) Are the predicted and actual RLDs identically distributed? Both hypotheses can be tested via a goodness-of-fit test. Although empirical RLDs are obviously discrete, we treat them as continuous random variables, primarily in order to avoid issues related to specification of the bin size in the standard χ-square goodness-of-fit test for discrete random variables. Instead, we use a two-sample Kolmogorov-Smirnov (KS) goodness-of-fit test for both hypotheses. In either case, the null hypothesis is that the distributions underlying both samples are identically distributed. The KS test statistic then quantifies the maximal distance between the (hypothesized or estimated) CDFs, and the null hypothesis is rejected if the distance between the two CDFs is sufficiently large [SM90].

Our primary goal in investigating the ability of our cost models to account for the full RLD is to fully disclose (as opposed to mask) any deficiencies in those cost models. While accurate prediction of the full RLD is the ideal objective, we note in advance that we do not achieve this goal. However, the predicted and actual RLDs are generally qualitatively (and occasionally quantitatively) identical, providing additional evidence that our dynamic cost models do accurately reflect algorithm run-time behavior. Further, as we discuss in Chapters 6–9, we believe the discrepancies are largely due to deficiencies in our transition probability estimation process.

4.6 Test Problems

Obviously, cost models must be constructed, tested, and validated using one or more sets of problem instances. It is therefore necessary to define a selection criterion. Research on algorithms for the JSP is nearly without exception performed using test problems

available from either the OR Library [1][Bea90] or Demirkol et al. [2][DMU98]. These two benchmarks contain an aggregate 242 problem instances, of variable size and difficulty [Jai98].

The immediate temptation is to select subsets of the most challenging benchmark problems: high accuracy on such benchmarks would strongly validate any cost model of local search. However, for a variety of both methodological and technical reasons, it is instead necessary to use sets of relatively small test instances. Many of the cost models developed in Chapters 6-9 are functions of all optimal solutions to a problem instance, which number in the millions even for small problem instances (e.g., see Section 5.3.1). Additionally, because search cost is a random variable, numerous samples are typically required to achieve stable estimates of the average search cost, leading to substantial aggregate run-times. Large sets of problem instances are also necessary to generate reliable statistics; for any given problem size, typically fewer than 10 benchmark instances are available.

The approach to developing cost models taken in this thesis is as follows. First, large sets of small-dimensional problem instances are used to construct the cost models and to perform preliminary validation. Second, scalability of the cost models is assessed using smaller sets of larger-dimensional problem instances. In order to link results to those previously reported in the literature, the cost models are also validated using benchmark instances whenever possible. In Chapters 6-9, the primary focus is on developing cost models of local search algorithms for *random* JSPs. Later, in Chapter 10, the resulting models are extended to more structured instances, i.e., workflow and flowshop JSPs. Below, the sets of problem instances used throughout this thesis, including both random and structured JSPs, are detailed.

4.6.1 Random JSPs

We construct our cost models and perform a first-order validation using sets of 6×4 and 6×6 problems instances, each containing 1,000 problem instances. For each instance, the τ_{ij} were uniformly sampled from the interval $[1, 99]$ and the job routing orders π_i were constructed using random permutations of the integers $[1..m]$. In contrast to Taillard [Tai93] and other researchers, we do not filter for difficult problems. Given that we are interested in cost models for typical problem instances, and any filtering would bias the resulting problem sets. While small in comparison to benchmark instances, these two problem sets respectively represent the largest rectangular and square sizes for which we can currently develop (due to the limitations of computing technology)

[1] Available from: http://www.ms.ic.ac.uk/info.html

[2] Available from: http://palette.ecn.purdue.edu/ uzsoy2/benchmark/problems.html

cost models using a large number of samples, i.e., problem instances.

To address scalability concerns, we also evaluate the accuracy of our models on a set of 100 random 10×10 instances (generated using the same procedures and parameter settings associated with the smaller problem sets). As indicated in Section 2.3, 10×10 instances are no longer viewed as particularly challenging for state-of-the-art JSP algorithms. However, these are the largest problems for which optimality can be routinely established by complete (i.e., branch-and-bound or constraint programming) algorithms, and most algorithms for the JSP still have difficulty in *consistently* locating optimal solutions to these problems in reasonable periods of time.

To tie our results to the JSP literature, we also analyze the accuracy of our cost models on the following well-known 10×10 benchmark problems, all of which are available from the OR Library: `la16-la20`, `abz`, and `abz6`. Lawrence [Law84] introduced `la16-la20`, which were generated by uniformly sampling the τ_{ij} from the interval $[5, 99]$. Adams et al. [ABZ88] introduced `abz5` and `abz6`; here, the τ_{ij} were uniformly sampled from the interval $[50, 100]$. With a single exception (Fisher and Thompson's `ft06` 6×6 instance), these instances are the smallest random JSPs in widespread use.

Although the bulk of our research is based on the 6×4, 6×6, and 10×10 problem sets introduced above, we also consider other sets of random JSPs at various points throughout this thesis (e.g., in our analysis of the JSP fitness landscape presented in Chapter 5). For each such problem set, we again generate the instances by uniformly sampling the τ_{ij} from the interval $[1, 99]$ and generating the π_i from random permutations of the integers $[1..m]$.

4.6.2 Sub-Optimal Random JSPs

Although we are primarily interested in modeling the cost required to locate optimal solutions, we do assess, at various points in Chapters 7 and 8, the ability of our models to account for the variability in the cost required to locate *sub-optimal* solutions to random JSPs. To enable such testing, we extend our 6×4 and 6×6 problem sets as follows: for each instance π with an optimal makespan C^*_{max}, we generate 25 new instances with a 'virtual' optimal makespan of $C^*_{max} + x$ by varying x from 1 to 25. Each such sub-optimal problem set contains 25,000 problem instances. The maximal offset of 25 was imposed due to the extreme numbers of solutions (e.g., ≥ 5 million) at $x > 25$.

4.6.3 Workflow and Flowshop JSPs

We analyze the ability of our cost models to generalize beyond the random JSP using sets of 6×4, 6×6, and 10×10 workflow and flowshop JSPs. The 6×4 and 6×6 sets contain 1,000 instances apiece, while the 10×10 sets contain 100 instances apiece. In all cases, the τ_{ij} were uniformly sampled from the interval $[1, 99]$. The job routing orders π_i for the workflow JSPs were constructed by concatenating random permutations of the integers $[1..m/2]$ and $[m/2 + 1..m]$; in flowshop JSPs, the π_i are deterministic.

As with cost models for the random JSP, we also consider well-known structured benchmark JSPs in order to link our results with the literature. Although no pure workflow or flowshop 10×10 benchmark instances are available, several instances do possess structured job routing orders that mirror workflow or flowshop partitions. For example, the job routing orders in Fisher and Thompson's infamous `ft10` instance are very similar to those found in workflow JSPs. Similarly, Applegate and Cook [AC91] introduced ten 10×10 instances, denoted `orb01-orb10`, whose job routing orders bear strong similarity with workflow or flowshop JSPs. All 11 of these instances were generated by uniformly sampling the d_{ij} from the interval $[1, 99]$.

Chapter 5

Structural Characteristics of the JSP Fitness Landscape

The performance of any local search algorithm is dictated by the interaction of the meta-heuristic with the underlying fitness landscape. Toward understanding this interaction, researchers have initiated numerous investigations of the structural characteristics of the fitness landscapes of various combinational optimization problems. As a result, several fitness landscape features have been identified that have been shown, via abstract argument or in specific circumstances, to influence problem difficulty for local search. Examples of such features include[1]:

- The number and/or distribution of local optima
- The strength and size of local optima attractor basins
- The size and extension of the search space

Although the importance of these features is widely acknowledged, little or no empirical evidence exists to substantiate the *extent* to which any of these features, or combination thereof, is actually correlated with local search cost. Because the strength of the relationships have not been quantified, it is possible or even likely that the prime factor(s) dictating problem difficulty for local search have either yet to be identified or remain largely unexplored.

Structural features of the fitness landscape also have, or at least *should* have, a major influence on the design of local search algorithms. Meta-heuristics differ largely in their approach to escaping the attractor basins of local optima, and the complexity

[1]Kauffman (p. 44, [Kau93]) provides a more comprehensive list, developed for adaptive local search algorithms.

of the proposed escape mechanisms - in terms of algorithmic details - is highly variable. Ideally, designers tailor a meta-heuristic to the class of fitness landscapes that the algorithm is likely to encounter. Yet, very few concrete details are known about attractor basin strength, i.e., the expected computational effort required to escape local optima. This is true for nearly all combinatorial optimization problems, including the JSP. Consequently, it is unclear whether further attention on novel escape mechanisms is warranted, or if researchers should shift their focus to designing more effective high-level search strategies, such as those associated with advanced implementations of tabu search.

In this chapter, we analyze the structure of the fitness landscape of *random* JSPs; we defer discussion of the differences between the fitness landscapes of random, workflow, and flowshop JSPs to Chapter 10. In particular, we (1) quantify the attractor basin strength of local optima in the JSP and (2) assess the potential for various features to serve as a basis for accurate static cost models of local search algorithms for the JSP. A key result of prior research on fitness landscape structure is that the landscapes of different combinatorial optimization problems often exhibit strong structural similarities. As a result, we are able to both leverage this research and identify new links between the JSP fitness landscape and that of other combinatorial optimization problems. We only present results obtained under the $N1$ move operator; qualitatively similar results hold for the $N5$ operator.

The rest of this chapter is organized as follows. First, we provide a high-level overview of prior research on the JSP fitness landscape in Section 5.1; specific details are provided as needed throughout the remainder of this chapter. The structure and strength of attractor basins in the JSP are analyzed in Section 5.2. Those landscape features we use to develop our static cost models are analyzed in Section 5.3. Much of the prior research on both problem difficulty and fitness landscapes performed for other combinatorial optimization problems are inappropriate for our research; we discuss these omissions in Section 5.4.

5.1 Prior Research

Prior to 2000, only Dirk Mattfeld and his colleagues had performed detailed analyses of the JSP fitness landscape. In his book *Evolutionary Search and the Job Shop* [Mat96], Mattfeld exhaustively analyzed the fitness landscape of Fisher and Thompson's `ft10` instance. In particular, he showed that (1) local optima in the JSP are distributed uniformly throughout the fitness landscape and (2) high-quality local optima share more features with each other and with globally optimal solutions than do mediocre local optima. In a related article, Mattfeld et al. [MBK99] extend this analysis to develop possible explanations for differences in the relative difficulty of random versus workflow JSPs, and why genetic algorithms (and, more generally, adaptive search algorithms) typically perform poorly on the JSP. Most recently, Nowicki and Smutnicki [NS01]

analyzed the fitness-distance relationship between local optima and globally optimal solutions in Taillard's benchmark problems, confirming Mattfeld's original observations. We detail many of Mattfeld's results in the remainder of this chapter. However, we defer discussion of Mattfeld's comparative analysis of the fitness landscapes of random and workflow JSPs to Chapter 10.

5.2 The Attractor Basin Structure of Local Optima in the JSP

A key decision in the design of any meta-heuristic for local search is how to escape the attractor basins of local optima. For example, Glover notes that tabu search overcomes the deficiencies of descent procedures by allowing local search to "continue exploration without becoming confounded by an absence of improving moves, and without falling back into a local optimum from which it previously emerged" ([Glo89], p. 191). Further, the escape mechanism is often the defining feature of any particular meta-heuristic. With few exceptions (e.g., some variants of tabu search), little attention is paid to higher-level search processes. Consequently, the global search strategy is typically implicit – and if a particular strategy is effective, the performance is necessarily fortuitous. The success of any escape mechanism (and by inference, meta-heuristic) depends largely on the strength and/or structure of the attractor basins in a given problem or problem instance. We now analyze the attractor basin strength of local optima in random JSPs, focusing on effective strategies for escape and the computational effort required.

5.2.1 Attractor Basin Size

Intuitively, the strength of the attractor basin of a local optimum s can be quantified by its size. A solution s' lies within the attractor basin of s if s results with non-zero probability when a descent procedure is applied to s'; basin membership is generally stochastic due to the use of randomization in most descent procedures. We denote the set of solutions in the attractor basin of s by $AB(s)$. The size of the attractor basin of s is then defined as $|AB(s)|$. It is theoretically possible to compute $|AB(s)|$ for any local optimum using reverse-hillclimbing algorithms, such as those introduced by Jones and Rawlins [JR93] or Jones [Jon95]. However, even for 6×6 JSPs, basin sizes can easily exceed 100 million solutions, making exact computation of $|AB(s)|$ intractable (reverse hill-climbing requires caching and/or hashing of solutions to identify duplicates). A related approach to quantifying attractor basin size is via its width or diameter.

Alternatively, we can estimate $|AB(s)|$ by computing the number of iterations k required by a descent procedure to reach s from a random initial solution; under this methodology, s is determined post-hoc. We refer to k as the *basin depth*. The intuition is that $|AB(s)|$ is proportional to k. Following Mattfeld [Mat96], we apply randomized descent (next or steepest) to 10,000 random semi-active solutions, and record both the

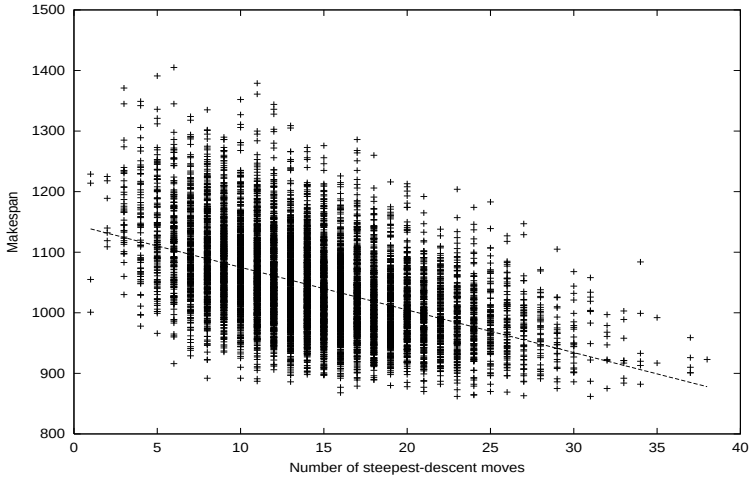

Figure 5.1: Scatter-plot of the number of iterations k of steepest-descent required to transform random semi-active solutions into local optimum versus the makespan of the resulting optimum.

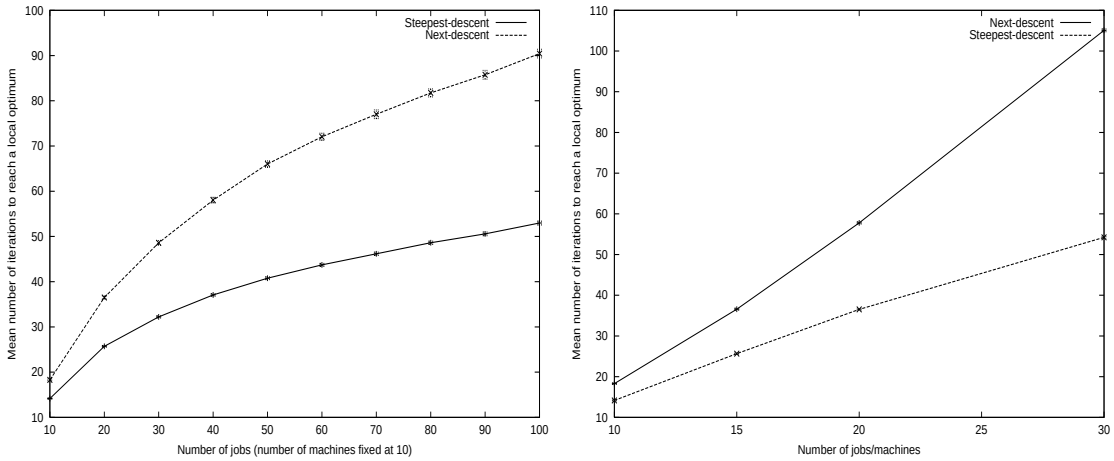

Figure 5.2: The mean basin depths for 100 instances of various rectangular (left figure) and square (right figure) random JSPs; data points are annotated with 95% confidence intervals.

number of iterations k required to reach a local optimum s and the makespan $C_{max}(s)$. We show a scatter-plot of k versus $C_{max}(s)$ under steepest-descent for a typical 10×10 instance in Figure 5.1. On average, roughly 15 iterations of steepest-descent are required to convert a random semi-active solution into a local optimum. Further, k is inversely correlated ($r = 0.2378$) with $C_{max}(s)$, suggesting that more solutions drain into high-quality local optima. We observe qualitatively similar results under next-descent, with a slight ($\approx 10\%$) increase in k.

Next, we analyze the impact of problem dimension on attractor basin size. We define basin depth for a problem *instance* as the mean basin depth under next- and steepest-

descent, with statistics taken over 1,000 independent trials. The mean basin depths for a range of rectangular (m fixed at 10, n varied from 10 to 100 in steps of 10) and square ($n = m$ equal to 10, 15, 20, and 30) are shown in the left and right sides of Figure 5.2, respectively. Each data point summarizes results for 100 instances, i.e., we compute a mean of means. The confidence intervals indicate that variability in the mean basin depth across different instances of a given problem size is small. With the exception of next-descent in the square problems, the mean basin depth appears to be approaching a linear asymptote (although there is a slight jump at $n = 100$). The results indicate that attractor basin size increases with problem size, and attractor basins in the JSP are generally quite large. However, the relationship between attractor basin size (as estimated by basin depth) and problem difficulty is less clear. Specifically, the mean basin depth fails to account for the variability in problem difficulty observed for JSPs with different dimensions. Specifically, if basin size is correlated with problem difficulty for local search, we would expect basin size to *decrease* in rectangular JSPs as $n/m \rightarrow \infty$.

5.2.2 Plateaus Versus Local Optima: Plateau Size

Measures such as depth and size only indirectly quantify attractor basin strength. A more operational or direct approach is to measure the level of computational effort actually required to escape. In general, a local optimum s in the JSP is a member of a plateau P of equally fit solutions, i.e., all with makespan equal to a constant C. Consequently, there exist two straightforward strategies for escaping local optima in the JSP. The first alternative is to search the containing plateau for an *exit*, i.e., a solution possessing a neighbor with an improving makespan; no less-fit neighbors are ever considered. The success of this strategy depends on two related factors: the size of the plateau and the proportion of member solutions possessing exits. The second alternative is to accept a sequence of less-fit neighbors such that when greedy descent is re-initiated, search terminates at a different local optimum with a non-zero probability. Here, success is dictated by the number of dis-improving moves that must be accepted before descent achieves a specified probability of terminating at a different local optimum.

We first analyze the two factors dictating the success of the plateau exit strategy. Below, we examine the size of plateaus in the JSPs; we consider exit probabilities later in Section 5.2.3. We investigate the factors dictating the effectiveness second strategy in Section 5.2.4.

To quantify plateau size, we generated 10,000 random local optima for each of our 10×10 random JSPs. For each local optimum s, we enumerated the set of solutions P such that for all $s' \in P$ the following conditions hold:

1. There exists a path $s = s_1, s_2, ..., s_k = s'$ such that $s_i \in N1(s_{i-1})$ for all $k \geq i \geq 2$.

2. $\forall s_i$ on the path from s to s', $C_{max}(s_i) = C_{max}(s)$.

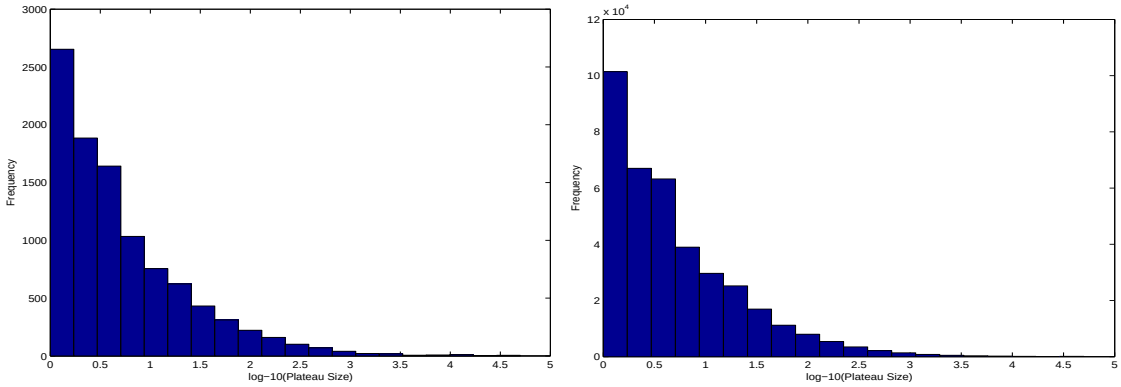

Figure 5.3: Histograms of plateau sizes for random 10×10 JSPs. Left figure: distribution for a typical problem instance. Right figure: aggregate distribution for 100 problem instances.

In contrast to many combinatorial optimization problems, the composition of a plateau P in the JSP is a function of the initial solution s; due to the asymmetry of the $N1$ operator and other critical-path move operators, a path from s to s' does not imply the existence of a path from s' to s.

We show the distribution of $|P|$ for a typical 10×10 JSP in the left side of Figure 5.3. The smallest plateau contained a single solution, while the largest contained 25,251 solutions. As indicated in the figure, most plateaus are very small: 7,824 plateaus contained 10 or fewer solutions, while 2,994 contained a single solution. The distribution of $|P|$ was similar for the other instances in our 10×10 problem set; we show the aggregate distribution in the right side of Figure 5.3. For larger problem instances, we observed significantly larger plateaus, such that exact computation of $|P|$ is generally prohibitive. However, the median value of $|P|$ remains relatively small for rectangular and square problem sizes ranging up to 70×10 and 30×30, respectively. We found no significant correlation between $|P|$ and C_{max}. Our initial expectation was that plateaus under the $N1$ operator would be large, as many moves fail to yield an improvement in the makespan. However, empirical results indicate that non-improving moves are often strictly dis-improving.

Assuming that exits from plateaus exist and are relatively numerous (an issue we examine next in Section 5.2.3), our results indicate that a plateau-intensive search strategy, e.g., restricting search to the plateau until an exit is located, could be quite effective, as most plateaus are very small. However, extremely large plateaus do exist, which would be costly to search unless the relative frequency of exits from solutions on the plateau is large.

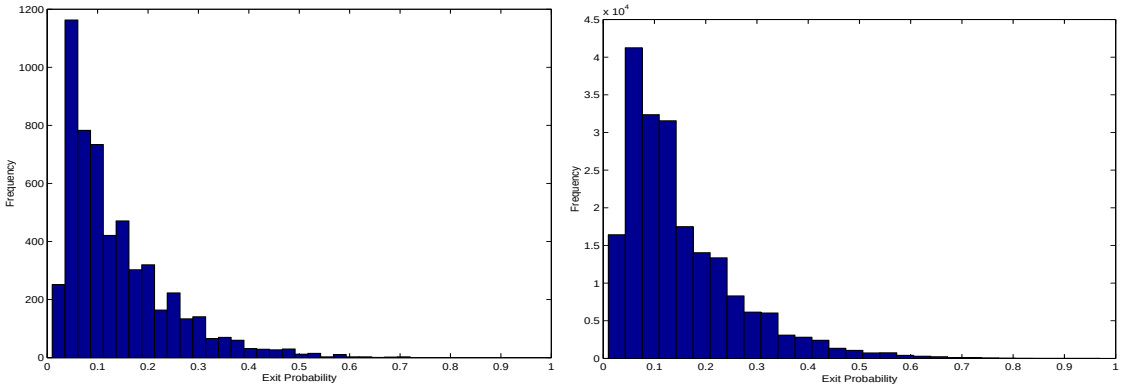

Figure 5.4: Histograms of bench exit probabilities for random 10×10 JSPs. Left figure: distribution for a typical problem instance. Right figure: aggregation distribution for 100 problem instances.

5.2.3 Plateaus Versus Local Optima: Exit Probabilities

As indicated above, the effectiveness of plateau-based search is predominantly a function of the relative frequency of exits from a plateau. We define the exit probability $Pr_{ext}(P)$ from a plateau P as the proportion of member solutions that possess at least one neighbor with a lower makespan than that of the plateau. We compute $Pr_{ext}(P)$ for each of the plateaus in all of the random 10×10 JSPs previously identified in Section 5.2.2. The first question is whether exits exist at all, i.e., whether a plateau P is locally optimal. Of the aggregate one million plateaus, we observed $Pr_{ext}(P) = 0$ in 578,239 cases; analogous results hold for individual instances. Consequently, in order to escape local optima in the JSP, it is generally necessary to accept a sequence of dis-improving moves.

Next, we analyze the distribution of Pr_{ext} for plateaus that are not locally optimal, i.e., that are *benches* (see Section 3.3). We show the distribution of Pr_{ext} for the 5,476 bench plateaus (the remaining 4,524 plateaus are locally optimal) of a typical 10×10 JSP in the left side of Figure 5.4. The distribution is roughly log-normal, such that $Pr_{ext} \leq 0.1$ for 2,706 of the benches. Similar results hold in our other 10×10 instances; we show the aggregate distribution in the right side of Figure 5.4. The results indicate that when a local optimum is a bench, only a few of the member solutions possess exits. Consequently, unless the bench is small, plateau-based search is unlikely to be effective. Finally, we find only weak correlation between Pr_{ext} and $|P|$ ($r = 0.1852$), and no statistically significant correlation between Pr_{ext} and C_{max}.

5.2.4 A Perturbation Analysis of Attractor Basin Strength

We have established that, in general, local search algorithms for the JSP must accept dis-improving moves to escape the attractor basins of local optima. The question is

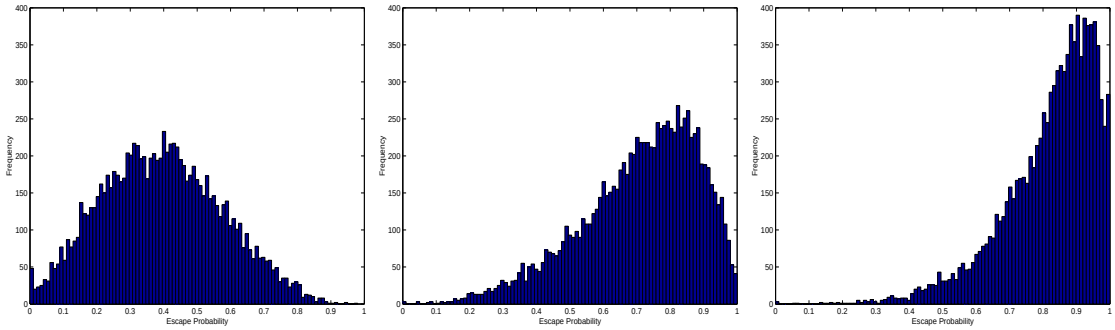

Figure 5.5: Distribution of local optima escape probabilities under next-descent for a typical random 10×10 JSP after accepting a random sequence of 1 (left figure), 3 (center figure), and 5 (right figure) less-fit neighbors.

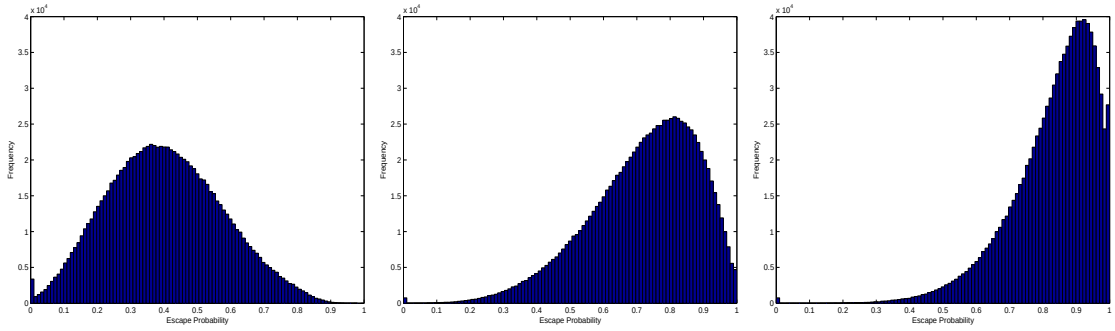

Figure 5.6: Distribution of local optima escape probabilities under next-descent for 100 random 10×10 JSPs after accepting a random sequence of 1 (left figure), 3 (center figure), and 5 (right figure) less-fit neighbors.

then: How *many* dis-improving moves are required? Consider a local optimum s and suppose that we accept a random sequence $S = s_1, s_2, ..., s_x = s'$ of monotonically less-fit neighbors of at most length k, such that the following conditions hold:

1. $x \leq k$
2. For $1 \leq i \leq x - 1$, $C_{max}(s_i) < C_{max}(s_{i+1})$

The 'at most' qualifier is necessary because local search can encounter a local *maximum* before k less-fit moves have been accepted. By applying randomized next-descent to s', we generate a local optimum s''. If $s'' \neq s$, then local search has escaped from s. Otherwise, local search has failed to escape the attractor basin of s. In effect, we are performing at most k iterations of next-*ascent*, followed by the maximum number of iterations of next-*descent*, i.e., the number of iterations required to reach a local optimum. Because the ascent and descent phases are stochastic, we can straightforwardly define the *escape probability* from a local optimum s as the fraction of N independent trials that terminate at a local optimum $s'' \neq s$. In all of our experiments, we let $N = 100$.

One complicating factor in estimating the escape probability from a local optimum s is the fact that s is generally a member of a plateau P of equally fit solutions. Using the aforementioned procedure, it is possible to escape to a new local optimum s'', yet $s, s'' \in P$. In other words, we must consider the problem of escaping both s and the plateau P containing s. In order to compute escape probabilities for large random JSPs (for which plateau enumeration is computationally prohibitive), we indicate that local search has escaped the attractor basin of the plateau P containing s if $C_{max}(s) \neq C_{max}(s'')$. This definition is conservative, as s'' may be a member of a different plateau than s, although two plateaus could have an equivalent makespan; in practice, this situation is rarely observed. To estimate the escape probability from a local optimum s, we use the next-ascent/next-descent procedure in conjunction with the makespan-based plateau escape criterion; we denote the result by P_{escp}.

We show the distribution of escape probabilities for 10,000 random local optima of a typical random 10×10 JSP in Figure 5.5, using $k = 1$, $k = 3$, and $k = 5$. Analogous aggregate results for 100 random 10×10 JSPs are shown in Figure 5.6. Surprisingly, it is possible to escape local optima with a non-negligible probability by accepting a *single* dis-improving move. Increasing k from 1 to 3 to 5 yields further, substantial increases in the escape probability. At $k = 5$, over 90% of the local optima can be escaped with probability ≥ 0.5; by $k = 10$, the probability increases to ≥ 0.9. However, independently of k, there exist local optima for which the escape probability is 0. Upon closer examination, these local optima often reside on plateaus that are also locally maximal (or near-maximal), such that it is not possible to accept any (or a large number of) dis-improving moves. In these situations, the only way to escape is to locate an exit from the containing plateau – which is guaranteed to exist, as the $N1$ operator induces a connected search space. In other cases, the local optimum s resides in the center of a very large plateau; by increasing k even further, it is then possible to escape.

We also analyzed the escape probabilities for larger square and more rectangular random JSPs, using 2,000 random local optima for each instance due to the increased computational requirements. We show the impact of problem size on the escape probability from local optima of square JSPs in Figure 5.7 with $k = 3$; we show the results for typical problem instances, as the aggregate distributions are similar. We observe that as n is increased, the distribution mass quickly shifts toward 1. For the 30×30 instance, the escape probability is $\geq .8$ for 90% of the local optima. Computational requirements currently prevent us from analyzing larger problem instances, although it appears that $\lim_{n \to \infty} P_{escp} = 1$. Similar results hold for rectangular JSPs as m/n is increased. For the 70×10 instance, $P_{escp} \geq 0.8$ for over 99% of the local optima. These results suggest that for large rectangular or square JSPs, problem difficulty for local search is largely *independent* of attractor basin strength, in that any meta-heuristic should be able to consistently escape local optima.

For all of the problem instances we analyzed, we found at best only moderate and unstructured correlation (e.g., $r \approx 0.35$) between P_{escp} and C_{max}. The lack of correlation may in part be due to our sampling methodology. Random local optima are

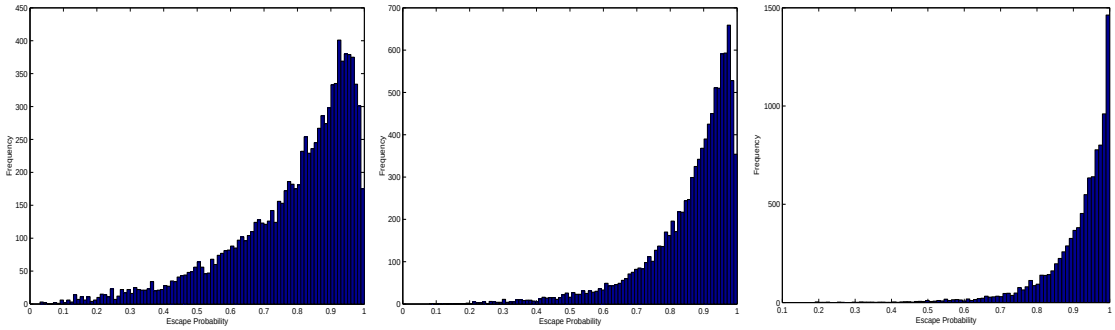

Figure 5.7: Distribution of local optima escape probabilities under next-descent after accepting 3 less-fit neighbors for typical 30×10 (left figure), 50×10 (center figure), and 70×10 (right figure) random JSPs.

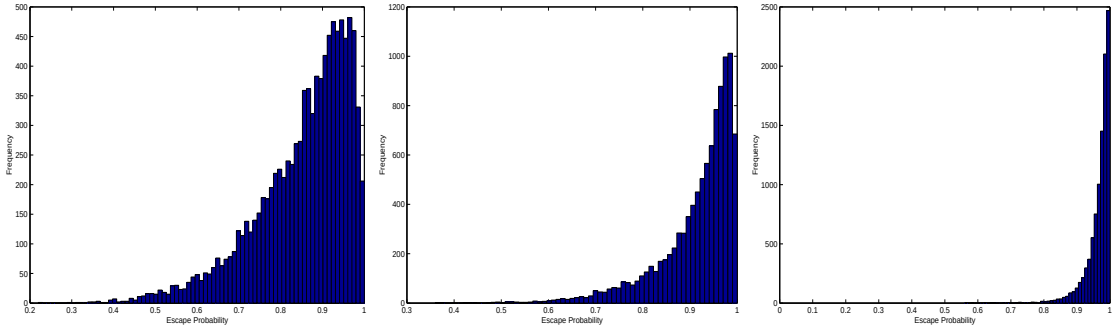

Figure 5.8: Distribution of local optima escape probabilities under next-descent after accepting 3 less-fit neighbors for typical 15×15 (left figure), 20×20 (center figure), and 30×30 (right figure) random JSPs.

generally poor-quality, and the probability of generating near-optimal solutions is very low; further, the probability decreases as problem size is increased. Consequently, a sampling method capable of consistently generating both 'average' and near-optimal solutions may enable identification of a stronger correlation between escape probability and solution quality. In Chapter 8, we introduce one such sampling methodology and extend the analysis introduced above.

Based on the analysis presented thus far, it is unclear whether the unexpectedly large escape probabilities we observed are an inherent property of the fitness landscape, a function of the next-descent strategy, or some combination thereof. To explore this issue, we extend our analysis by substituting a steepest-descent procedure for the randomized next-descent used to convert intermediate solutions (i.e., s') into local optima (i.e., s''). At each iteration, our implementation of steepest-descent simply selects the neighbor with the lowest makespan, with ties broken randomly; the procedure terminates when a local optimum is encountered. While steepest-descent is randomized, it is significantly less so than next-descent.

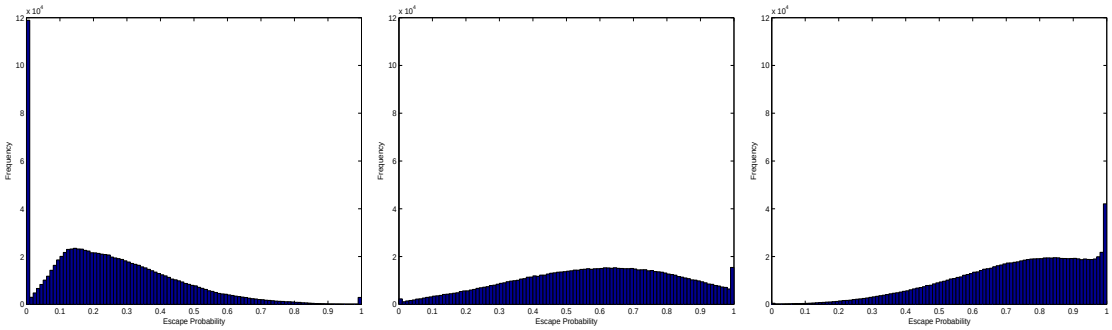

Figure 5.9: Distribution of local optima escape probabilities under steepest-descent for random 10×10 JSPs after accepting a random sequence of 1 (left figure), 3 (center figure), and 5 (right figure) less-fit neighbors.

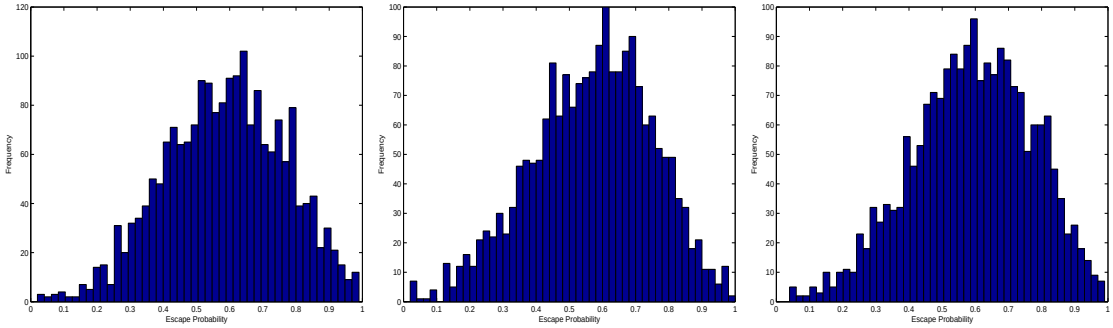

Figure 5.10: Distribution of local optima escape probabilities under steepest-descent after accepting 3 less-fit neighbors for typical 30×10 (left figure), 50×10 (center figure), and 70×10 (right figure) random JSPs.

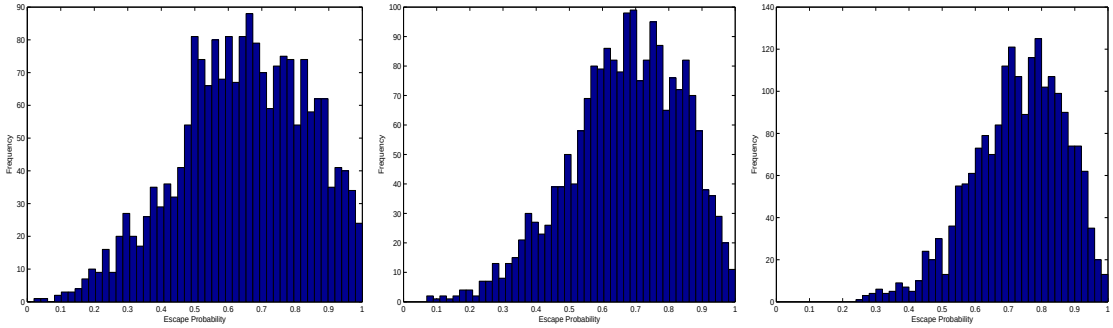

Figure 5.11: Distribution of local optima escape probabilities under steepest-descent after accepting 3 less-fit neighbors for typical 15×15 (left figure), 20×20 (center figure), and 30×30 (right figure) random JSPs.

We show the aggregate distribution of escape probabilities under steepest-descent for 100 random 10×10 JSPs in Figure 5.9, using $k = 1$, $k = 3$, and $k = 5$; for each instance, we generated 10,000 random local optima. Although generally lower than under next-descent, P_{escp} is still non-negligible for 90% of the local optima: ≥ 0.1 for $k = 3$ and ≥ 0.3 for $k = 5$. With respect to next-descent, the largest differences are due to the large numbers of local optima at $k = 1$ for which $P_{escp} = 0$. In addition to cases when a local optima is also a local maximum, the latter occurs for a local optimum s when the makespan of the best neighbor of *any* solution $s' \in N1(s)$ is equal to $C_{max}(s)$. Similar results hold for larger square (see Figure 5.11) and rectangular (see Figure 5.10) problem instances, although the asymptotic approach toward $P_{escp} = 0$ appears significantly more delayed. In conclusion, the weakness of attractor basins in the JSP is largely a structural feature of the fitness landscape, although it can be influenced by the choice of descent strategy. It is currently unclear to what extent the weakness is due to the asymmetry of the $N1$ move operator, as asymmetry can prevent search from re-descending to a local optimum.

5.2.5 Implications

Our results concerning attractor basin structure suggest several immediate hypotheses regarding the design and behavior of local search algorithms for the JSP. First, we observe *no* correlation between the size of an attractor basin and its strength. Consequently, models of problem difficulty based on attractor basin size are of very limited practical use. Second, we can use the fact that a very small number of moves is required to escape local optima to guide the design of new meta-heuristics. We introduce one such algorithm in Chapter 8, and demonstrate that elaborate escape mechanisms are not necessary for effective local search. Third, by comparing the average basin depths with the number of moves required to escape local optima, it is clear that any meta-heuristic with a strong bias toward local optima (e.g., tabu search, iterated local search, and moderate-to-low-temperature simulated annealing) is effectively restricting search to a small fraction of the search space. This observation forms the basis for our development of cost models in Chapters 7 through 9, and may account for a substantial proportion of the difference in performance between well-known local search algorithms and a pure random walk. Fourth, by expressing attractor basin strength in terms of the number of *moves* required for escape, we identify possible deficiencies with fitness-based escape mechanisms. At low temperatures, even a single dis-improving move may yield a significant increase in the makespan, providing a possible explanation for the typically inferior performance of simulated annealing; we explore this issue in Chapter 9. Our results also suggest that the height of the barriers between local optima (i.e., the decrease in fitness that must be accepted in order to escape local optima) need not have an impact on local search [RS01]; it may be possible (as discussed in Chapter 8) to develop escape mechanisms based strictly on the number of dis-improving moves, and not in terms of changes in relative fitness.

5.2.6 Related Research

Frank et al. [FCS97] perform a comprehensive analysis of attractor basin strength in MAX-SAT. More so than in the JSP, the MAX-SAT fitness landscape is dominated by plateaus of equally fit solutions. Our plateau-related terminology and the methodologies employed in Sections 5.2.2 and 5.2.3 are adapted from their research. The distribution of plateau sizes in both the JSP and MAX-SAT are very similar. In contrast, exit probabilities from benches in MAX-SAT are significantly higher than in the JSP. Further, solution quality and exit probability are significantly correlated in MAX-SAT.

5.3 The Global Structure of the JSP Fitness Landscape

Once equipped with an effective escape mechanism, a local search algorithm must decide how to move through the fitness landscape from one local optimum to another. The effectiveness of this process depends to a large extent on overall global structure of the fitness landscape.

We now introduce those global structural features of the fitness landscape that have been proposed to account for the variability in problem difficulty for local search. In most cases, the features have been explored in the context of other NP-hard problems, primarily MAX-SAT and the TSP. We present the motivation behind each feature, summarize prior research, identify limitations, and analyze the extent of the feature in random JSPs. We generally restrict our attention to those problem sets that we use to develop our cost models (6×4, 6×6, and 10×10), although when computationally feasible, we additionally consider larger square and rectangular JSPs. Our goal is to introduce various features and to assess the potential of these features to serve as the basis of accurate static cost models of local search in the JSP, e.g., as developed in Chapters 6 - 9.

5.3.1 The Number of Optimal Solutions

One of the most intuitive measures of problem difficulty is the number of globally optimal solutions to a problem instance. It should be difficult to locate a global optimum if they are relatively rare. Conversely, if global optima are numerous, then it should be relatively easy for local search to find one. In the JSP, the makespan C^*_{max} of any optimal solution is completely determined by the set of operations on the critical path. However, it is often possible to re-order one or more sequences of non-critical operations in an optimal solution without increasing the makespan, and multiple critical paths can induce C^*_{max}. Consequently, the number of optimal solutions is proportional to the flexibility in sequences of non-critical operations, and is therefore likely to be highly variable.

For a given problem instance, we compute the set of optimal solutions using the constraint programming algorithm described in Section 3.6; we denote the result by $|optsols|$. Histograms of $log_{10}(|optsols|)$ for our 6×4 and 6×6 problem sets are shown

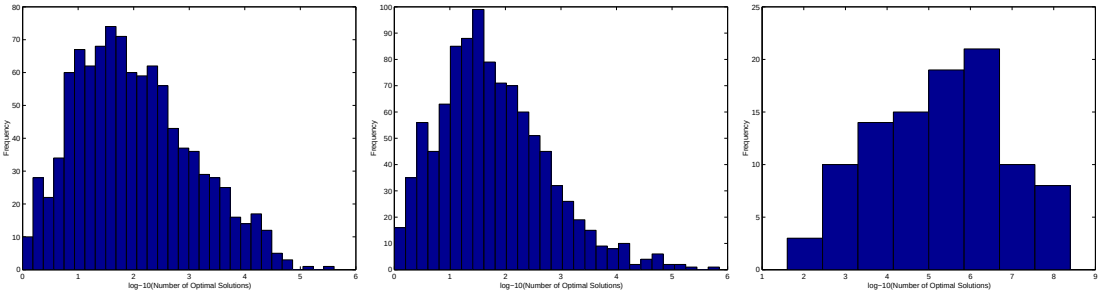

Figure 5.12: Histograms of the number of optimal solutions ($|optsols|$) for 6×4 (left figure), 6×6 (center figure), and 10×10 random JSPs.

Instance	abz5	abz6	la16	la17	la18	la19	la20
$\|optsols\|$	580	2,159	6,753,442	11,787,154	42,158	960	14,106

Table 5.1: The number of optimal solutions ($|optsols|$) for 10×10 benchmark JSPs.

in the left and center portions of Figure 5.12, respectively. In both cases, $|optsols|$ varies over six orders of magnitude, ranging from 1 to 406,073 for the 6×4 instances and from 1 to 710,627 for the 6×6 instances. On average, 6×6 instances possess fewer optimal solutions than 6×4 instances, despite a substantial increase in the size of the search space. Specifically, there are more 6×6 instances with $1 \leq |optsols| \leq 100$, and the right-tail density is lower than that observed for the 6×4 problem instances.

We also computed $|optsols|$ for our 10×10 instances, terminating the enumeration process once $|optsols|$ exceeded 250 million; the limit was reached for 3 instances. We show the resulting distribution of $|optsols|$ in the right side of Figure 5.12. As expected (due to the relative size of the search spaces), increasing the problem size inflates $|optsols|$, which ranges from 40 to over 250 million, or over nearly eight orders of magnitude. For reference, we also computed $|optsols|$ for each of the 10×10 random JSPs found in the OR Library; we report the results in Table 5.1. Both `abz5` and `la19` are generally considered the more difficult 10×10 benchmark instances, and correspond to the instances with the fewest optimal solutions. However, we do not anticipate perfect correlation between $|optsols|$ and search cost, as `la16` is generally more difficult than `abz6`.

5.3.1.1 Related Research

The relationship between the number of globally optimal solutions and problem difficulty for local search was originally analyzed in the context of MAX-SAT and the more general MAX-CSP [CFG+96]. The motivation behind this research was to develop an explanation for the easy-hard-easy pattern in problem difficulty observed in the phase transition regions of these problems [KS94, Pro94]. It was initially conjectured that

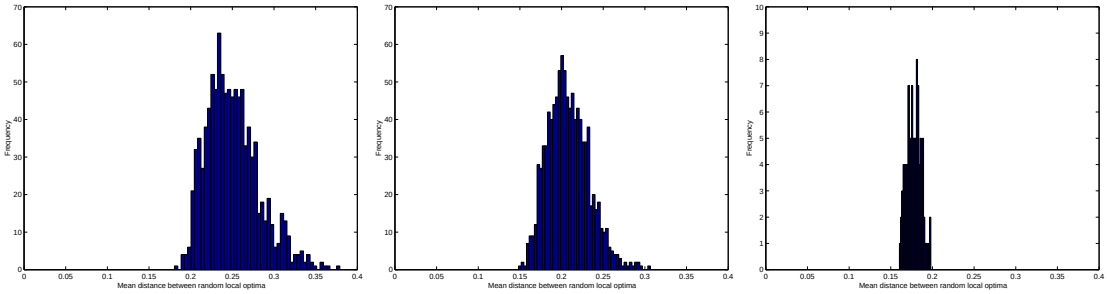

Figure 5.13: Histograms of mean distance between random local optima ($\overline{d}_{lopt\text{-}lopt}$) for 6×4 (left figure), 6×6 (center figure), and 10×10 random JSPs.

the peak in search cost was due to changes in the number of optimal solutions. Yokoo [Yok97] proved that this was not the case, by showing that the mean number of optimal solutions varies in no special way near the phase transition region. In contrast, when the problem size is held constant, Clark et al. [CFG+96] demonstrated a relatively strong (≈ 0.91) $log_{10} - log_{10}$ correlation between the number of optimal solutions and local search cost.

The distribution of the number of optimal solutions depends in large part on the form of the objective function, specifically whether all or a fraction of solution attributes dictate solution fitness. For example, the number of optimal solutions to instances of the 2-D integer Euclidean Traveling Salesman Problem is generally very small, and is frequently equal to 1 [SW01]. The reason is straightforward: tour length is a function of *all* the cities in the instance, and the likelihood of two tours having identical lengths is relatively small given randomly sampled inter-city distances. The likelihood of a single optimal solution is even higher if real-valued city coordinates are allowed. A similar situation is observed in the Permutation Flow-Shop Problem [Stu01]. In contrast, the fitness of solutions in the JSP is dictated by a subset of job orderings, i.e., those on the critical path.

5.3.2 The Mean Distance Between Random Local Optima

The cost of local search is also influenced by the size of the search space. Search in iterated local search, tabu search, and low-to-moderate temperature simulated annealing is heavily biased toward local optima. Coupled with the fact that attractor basins in the JSP are relatively weak, intuition suggests that problem difficulty is in part a function of the size of the sub-space containing local optima. A straightforward approach to quantifying the size of the local optima sub-space is to simply measure the mean distance between a sample of random local optima; large distances should be indicative of large sub-spaces.

For a given problem instance, we generate 5,000 random local optima and compute the mean disjunctive graph distance between all pairs of optima, normalized by the

Prb. Set	Mean $\overline{d}_{\text{rand-rand}}$	Mean $\overline{d}_{\text{lopt-lopt}}$	Min. $\overline{d}_{\text{rand-rand}} - \overline{d}_{\text{lopt-lopt}}$	Mean $\overline{d}_{\text{rand-rand}} - \overline{d}_{\text{lopt-lopt}}$	Max. $\overline{d}_{\text{rand-rand}} - \overline{d}_{\text{lopt-lopt}}$
6×4	0.2922	0.2504	0.0070	0.0419	0.0731
6×6	0.2442	0.2080	0.0063	0.0362	0.0611
10×10	0.1926	0.1780	0.0099	0.0146	0.0190

Table 5.2: Statistics for the difference between (1) the mean distance between random solutions ($\overline{d}_{\text{rand-rand}}$) and (2) the mean distance between random local optima ($\overline{d}_{\text{lopt-lopt}}$).

the maximum possible distance $m \cdot \binom{n}{2}$. We denote the result by $\overline{d}_{\text{lopt-lopt}}$, such that $\overline{d}_{\text{lopt-lopt}} \in [0, 1]$. For problem instances in which local optima are distributed evenly throughout the fitness landscape, we would expect $\overline{d}_{\text{lopt-lopt}} \approx 0.5$.

We show the distribution of $\overline{d}_{\text{lopt-lopt}}$ for our 6×4, 6×6, and 10×10 problem sets in Figure 5.13. Without exception, the estimated $\overline{d}_{\text{lopt-lopt}}$ is significantly less than the maximal value of 0.5, indicating that local optima in the random JSP are clustered in a restricted region of the fitness landscape. Mattfeld et al. [MBK99] indicate such clustering is due to the constraints imposed by the job routing orders, and show that the clustering is less significant in workflow JSPs (we demonstrate similar results for both workflow and flowshop JSPs in Chapter 10).

Although we expected larger $\overline{d}_{\text{lopt-lopt}}$ in the more difficult problem sets, we actually observe the largest range of $\overline{d}_{\text{lopt-lopt}}$ in rectangular 6×4 JSPs. We also found no significant differences in $\overline{d}_{\text{lopt-lopt}}$ between 10×10 JSPs and larger rectangular (50×10) and square (30×30) instances. Further, the variability in $\overline{d}_{\text{lopt-lopt}}$ quickly approaches 0 as the problem size is increased, while the mean remains constant at ≈ 0.19. All of these results run counter to intuition, raising the strong possibility that $\overline{d}_{\text{lopt-lopt}}$ may fail to account for any significant proportion of the variability in problem difficulty for local search.

In both a random walk and high-temperature simulated annealing, search proceeds in the space of all feasible solutions, as opposed to the local optima sub-space. To model problem difficulty for these algorithms, we introduce a simple extension of the $\overline{d}_{\text{lopt-lopt}}$ measure. Based on random semi-active solutions instead of random local optima, the new measure, which we denote $\overline{d}_{\text{rand-rand}}$, captures the size of the sub-space of feasible solutions. Analogous to the case for $\overline{d}_{\text{lopt-lopt}}$, we estimate $\overline{d}_{\text{rand-rand}}$ using 5,000 random semi-active solutions. The distributions of $\overline{d}_{\text{rand-rand}}$ are qualitatively identical (they possess roughly identical means/variances and similar tails) to those shown in Figure 5.13.

In many combinatorial optimization problems, local optima occupy a significantly smaller region of the fitness landscape than random solutions. In Table 5.2, we report statistics for the per-instance difference between $\overline{d}_{\text{rand-rand}}$ and $\overline{d}_{\text{lopt-lopt}}$ for each of our problem sets. The results indicate that for the 6×4 and 6×6 instances, the local optima sub-space is slightly compressed relative to the space of random solutions. We observe even less compression in the 10×10 problem set and in a limited sampling of larger square and rectangular problem instances. These results suggest that local optima in the JSP are distributed uniformly throughout the space of random (feasible) solutions,

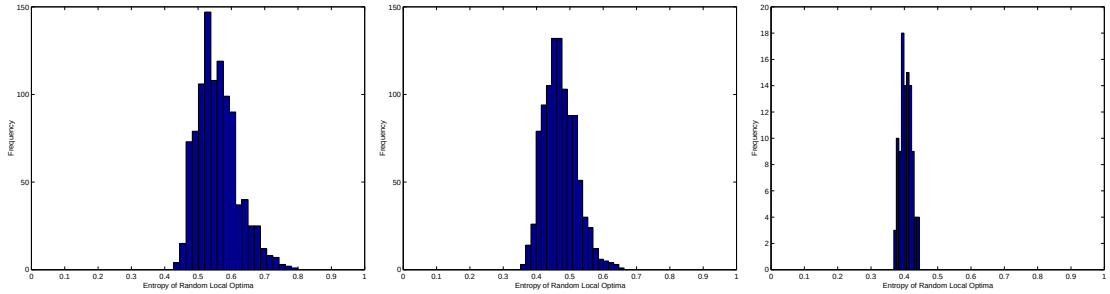

Figure 5.14: Histograms of entropy of random local optima ($\overline{e}_{lopt}$) for 6×4 (left figure), 6×6 (center figure), and 10×10 random JSPs.

in contrast to other well-known combinatorial optimization problems such as the TSP [MÖ90] and the quadratic assignment problem [FF94].

5.3.2.1 Related Research

Mattfeld et al. [MBK99] first introduced the notion of $\overline{d}_{lopt\text{-}lopt}$ as a measure of the size of the local optima sub-space; we have simply extended their analysis to a broader range of problem instances. However, Mattfeld et al. did not investigate the ability of $\overline{d}_{lopt\text{-}lopt}$ to account for variability in the difficulty of fixed-size problem instances. Rather, they demonstrated mean differences in $\overline{d}_{lopt\text{-}lopt}$ between random and workflow JSPs, and argued that differences in relative difficulty were due to differences in $\overline{d}_{lopt\text{-}lopt}$.

5.3.3 Entropy of Random Local Optima

Entropy [Sha48] measures the disorder in a set of solutions. If a set of solutions shares many attributes, then they are likely to be located in a restricted region of the fitness landscape. Conversely, if solutions are very disparate, the size of the fitness landscape is necessarily large. Suppose a particular attribute X of any solution can take on one of n possible values $x_1, x_2, \ldots, x_n$. Given a set of solutions, let $P(X = x_i)$ denote the fraction of member solutions in which attribute X is assigned value x_i. The entropy $E(X)$ of the *attribute* X is defined as $E(X) = -\sum_{i=1}^{n} P(X = x_i) \cdot log(P(X = x_i))$. Minimal entropy occurs when $\exists j : P(X = x_j) = 1$ and $\forall i \neq j : P(X = x_j) = 0$, in which case $E(x) = 0$. Maximal entropy occurs when all values x_i are equally likely ($P(X = x_i) = 1/n$), in which case $E(X) = log(N)$. $E(X)$ is typically normalized by $log(N)$, forcing $0 \leq E(X) \leq 1$. The entropy of a set of *solutions* is defined as the average entropy of individual solution attributes.

In the JSP, we take as individual solution attributes the precedence relations between pairs of jobs on the same machine. Let the predicate $\textit{precedes}_{ijk}(s)$ indicate whether job j precedes job k on machine i in a solution s. Let p_{ijk} denote the fraction of solutions s in a given set S in which $\textit{precedes}_{ijk}(s)$ is true. The entropy E_{ijk} of the corresponding precedence relation is then given by $E_{ijk} = p_{ijk} \cdot \textit{log}(p_{ijk}) +$

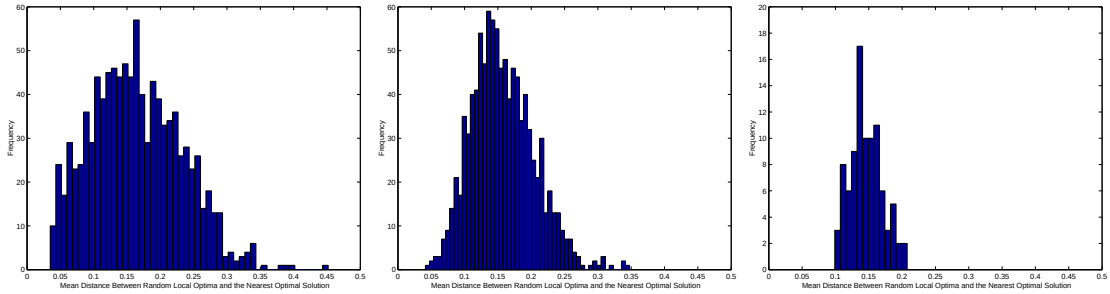

Figure 5.15: Histograms of $\overline{d}_{\textit{lopt-opt}}$ for 6×4 (left figure), 6×6 (center figure), and 10×10 random JSPs.

$(1 - p_{ijk}) \cdot \textit{log}(1 - p_{ijk})$. The entropy $E(S)$ of a set of solutions S is then defined as $E(S) = \frac{1}{n(n-1)m} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1,k\neq j}^{n} E_{ijk}(S)$.

For a given problem instance, we use a set S of 5,000 random local optima to compute an estimate of the mean entropy $\overline{e}_{\textit{lopt}} = E(S)$. We show the distribution of $\overline{e}_{\textit{lopt}}$ for our 6×4, 6×6, and 10×10 problem sets in Figure 5.14. We immediately observe very strong qualitative similarities in the distributions of $\overline{e}_{\textit{lopt}}$ and $\overline{e}_{\textit{rand}}$ (shown in Figure 5.14), raising the possibility that the two measures are essentially redundant, in that they both quantify the similarity of a set of solutions, with identical biases and/or inaccuracies. This hypothesis is easily confirmed by computing the correlation between the two measures, which respectively yield (Pearson's) r-values of 0.9955, 0.9945, and 0.9898 for the 6×4, 6×6, and 10×10 problem sets. Consequently, we can safely ignore one of these measures in developing our static cost models; we base all subsequent analysis on the $\overline{d}_{\textit{lopt-lopt}}$ measure.

5.3.4 Mean Distance Between Random Local Optima and the Nearest Optimal Solution

The number of optimal solutions and the size of the search space S are conceptually (and statistically - see Sections 6.2 and 7.5.6) independent; it is possible to embed as many as $|S|$ optimal solutions within a search space S. Undoubtedly, both factors influence problem difficulty for local search. If we fix $|S|$ and assume that attractor basin strength and size remain relatively constant, we expect problems to become easier as $|\textit{optsols}| \rightarrow |S|$. Analogously, if we fix $|\textit{optsols}|$, it should be more difficult to locate an optimal solution as $|S| \rightarrow \infty$. It follows that both the $|\textit{optsols}|$ and $\overline{d}_{\textit{lopt-lopt}}$ are, in isolation, unlikely to account for a significant proportion of the variability in problem difficulty.

To correct for these flaws, we introduce a measure that simultaneously accounts for the impact of both features on problem difficulty: the mean distance between random local optima and the *nearest* optimal solution. We denote the quantity by $\overline{d}_{\textit{lopt-opt}}$. The intuition is that problem difficulty for local search is proportional to the total distance that must be traversed – between an initial solution (i.e., a random local optima) and a

target solution (i.e., an optimal solution).

To compute an estimate of $\overline{d}_{\textit{lopt-opt}}$ for a given problem instance, we generate a set L of 5,000 random local optima. For each optimum $l \in L$, we then compute the disjunctive graph distance $D(l, s_{\textit{nearopt}})$ between l and the nearest optimal solution $s_{\textit{nearopt}}$. We then take $\overline{d}_{\textit{lopt-opt}}$ as the mean $D(l, s_{\textit{nearopt}})$, subsequently normalized by the maximum possible distance $m \cdot \binom{n}{2}$. The maximal value of $\overline{d}_{\textit{lopt-opt}}$ is equal to 0.5, and can only be achieved under relatively strict conditions: when relatively few optimal solutions are concentrated in a compact region of the fitness landscape and random local optima are distributed throughout the fitness landscape ($\overline{d}_{\textit{lopt-lopt}} = 0.5$).

In Figure 5.15, we show histograms of $\overline{d}_{\textit{lopt-opt}}$ for our 6×4 (left figure), 6×6 (center figure), and 10×10 (right figure) random JSPs. Results for eight 10×10 instances with more than 50 million optimal solutions are omitted, due to the intractability of computing $\overline{d}_{\textit{lopt-opt}}$. The observed $\overline{d}_{\textit{lopt-opt}}$ are typically far less than the theoretical maximum of 0.5, which is expected given that (1) feasible solutions in the random JSP are clustered in sub-space of the fitness landscape and (2) optimal solutions are not always tightly clustered. For the square problem sets, the distribution of $\overline{d}_{\textit{lopt-opt}}$ is qualitatively Gaussian, with mean ≈ 0.15 and a relatively low variance. The variance appears to drop when moving from 6×6 to 10×10 JSPs, but this may be an artifact of the differences in the total number of problem instances. We are unable to determine whether the variance drops to 0 as problem size increases, as occurs for $\overline{d}_{\textit{lopt-lopt}}$, due to astronomical numbers of optimal solutions.

The relative distribution of $\overline{d}_{\textit{lopt-opt}}$ in the square and rectangular problem sets is not entirely consistent with the differences in relative difficulty. Although there are significantly more 6×4 instances with $\overline{d}_{\textit{lopt-opt}} \leq 0.1$ than observed in the 6×6 problem set, there are also *more* 6×4 instances with $\overline{d}_{\textit{lopt-opt}} \geq 0.25$. Due to the number of optimal solutions, we are unable to assess the impact of changes in n/m on the distribution of $\overline{d}_{\textit{lopt-opt}}$ for larger problem sets..

To model problem difficulty in both a random walk and high-temperature simulated annealing, where search proceeds in the space of all feasible solutions, we investigate a simple extension of the $\overline{d}_{\textit{lopt-opt}}$ measure. We denote this measure $\overline{d}_{\textit{rand-opt}}$. The estimation methodology and intuition remain the same; the only difference is that random semi-active solutions are substituted for random local optima. The distributions of $\overline{d}_{\textit{rand-opt}}$ are qualitatively identical to those shown in Figure 5.15.

5.3.4.1 Related Research

Both $\overline{d}_{\textit{lopt-opt}}$ and $\overline{d}_{\textit{rand-opt}}$ are based on an analogous measure proposed to account for variability in problem difficulty for local search in MAX-SAT [SGS00]. Well-known local search algorithms for MAX-SAT rapidly descend from poor-quality initial solutions to near-optimal 'quasi-solutions', and subsequent search is restricted to the space of such quasi-solutions. Singer et al. [SGS00] hypothesized that the search cost was proportional to the size of the quasi-solution sub-space, which in turn could be estimated

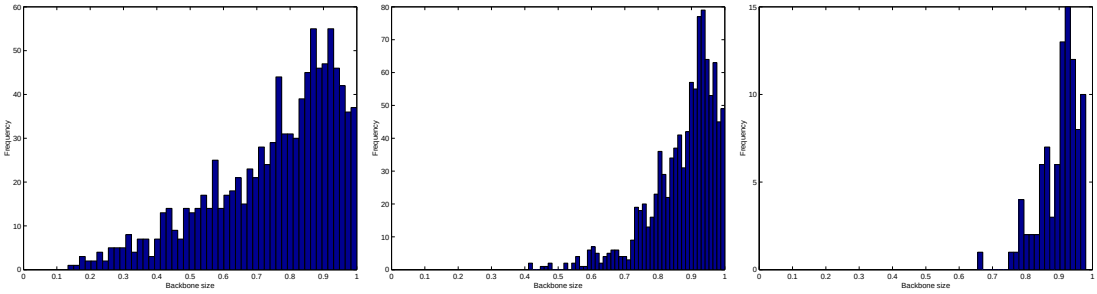

Figure 5.16: Histograms of the backbone size ($|backbone|$) for 6×4 (left figure), 6×6 (center figure), and 10×10 random JSPs.

Instance	abz5	abz6	la16	la17	la18	la19	la20
$\|backbone\|$	0.9733	0.9511	0.8956	0.8822	0.9289	0.9689	0.9244

Table 5.3: The backbone size ($|backbone|$) for 10×10 benchmark JSPs.

by the mean distance between the first quasi-solution encountered and the nearest optimal solution, which we denote $\overline{d}_{quasi\text{-}opt}$. Their experimental results demonstrated a very strong ($r \approx 0.95$) correlation between $\overline{d}_{quasi\text{-}opt}$ and the logarithm of search cost for easy MAX-SAT instances; for more difficult instances, the accuracy degraded only slightly to $r \approx 0.75$.

The quasi-dynamic model proposed by Singer et al. is a landmark achievement, as it represents the first reasonably accurate cost model of *any* local search algorithm, for *any* combinatorial optimization problem. Previously proposed models achieved accuracy of at most $r^2 \approx 0.3$ in the worst case, in contrast to the $r^2 \approx 0.6$ achieved by Singer et al.

5.3.5 Backbone Size

Recently, a number of researchers (e.g., [AGKS00] and [SW01]) have hypothesized that the *backbone* of a problem instance may be correlated with problem difficulty. Informally, the backbone of an instance is the set of solution attributes or variables that possess identical values in *all* optimal solutions; as a consequence, the definition of a backbone depends on the representation scheme used to encode solutions. The intuition behind the backbone measure is that the majority of effort in local search may be spent assigning correct values to backbone variables. Non-backbone variables appear to be significantly less constrained, enabling search to quickly locate an optimal solution once the backbone is located.

We define the backbone of a JSP in terms of a disjunctive graph representation (Section 2.4), in which solutions to $n \times m$ JSP are encoded using $n(n-1)/2$ Boolean 'order' variables for each of the m machines; each order variable represents a precedence relation between a distinct pair of jobs on a machine. Given a problem instance, we first

compute the set of order variables that have the same value in all optimal solutions. We then define the backbone size as the fraction of the possible $mn(n-1)/2$ order variables that are fixed to the same value in all optimal solutions. We denote the result by $|$*backbone*$|$; clearly, $0 \leq |\textit{backbone}| \leq 1$.

In Figure 5.16, we show histograms of $|$*backbone*$|$ for our 6×4 (left figure), 6×6 (center figure), and 10×10 (right figure) problem sets. Data for the three 10×10 instances with $|$*optsols*$| >$ 200 million are omitted. The majority of instances in both square problem sets possess very large backbones, while we observe a larger proportion of small-backboned instances in the rectangular 6×4 problem set. The results suggest that $|$*backbone*$|$ may become more skewed toward 0.0 as $n/m \rightarrow \infty$.

We show the $|$*backbone*$|$ for 10×10 OR Library random JSPs in Table 5.3. The most difficult instances, `abz5` and `la19`, possess the largest backbones, which is again consistent with the hypothesis that backbone size is correlated with problem difficulty. However, as with $|$*optsols*$|$, inconsistencies do exist: the backbone of `la16` is smaller than `abz6`, although the former is the more difficult of the two instances.

5.3.5.1 Related Research

The recent interest in backbones is due in large part to the observation that large-backboned problem instances begin to appear in large quantities near the critical region of the Random 3-SAT phase transition [SC96] [Par97] [MZK+98] [SGS00]; the coincidence of the two observations immediately leads to the hypothesis that backbone size is correlated with problem difficulty. More recently, Achlioptas et al. argue that the shift from small to large-backboned instances in the phase transition region suggests that the most difficult instances may in fact have a backbone size of 0.5 [AGKS00], although this hypothesis has not been verified. Slaney and Walsh [SW01] analyze the correlation between problem difficulty and backbone size for constructive search algorithms for a number of NP-hard optimization problems. For the Traveling Salesman and Number Partitioning Problems, they report a weak-to-moderate correlation (e.g., r between 0.138 and $r = 0.388$) between backbone size and the cost of locating an optimal solution.

5.4 Other Research on Problem Difficulty and Local Search

All of the features we discuss in Section 5.3 are either drawn directly from, or based analogies with, prior research on fitness landscape structure, both for the JSP and other combinatorial optimization problems. Yet, we have also ignored the *majority* of research on landscape structure and problem difficulty, comprising the following three areas: phase transitions in problem difficulty, landscape correlation length, and landscape fitness-distance correlation. These omissions are intentional: the research is either not directly applicable to our goal of modeling variability in problem difficulty, or not rel-

evant to the behavior of the non-adaptive local search algorithms that we consider. We now briefly summarize each of these research areas, and discuss the reasons for their omission.

The majority of research on problem difficulty within the AI community has focused on the identification of so-called *phase transitions* in problem difficulty [HHW96]. A phase transition in a combinatorial optimization problem (or the corresponding decision problem) identifies an order parameter that partitions the universe of problem instances into subsets with differing degrees of expected difficulty. For example, the clause-to-variable ratio m/n in Random 3-SAT induces a clear pattern: as m/n ranges from 0 to ∞, the degree of problem difficulty exhibits a well-known easy-hard-easy pattern [CKT91]. While successful in identifying inter-partition differences in problem difficulty, phase transitions fail to account for the often considerable variability *within* a partition; the latter can vary over many (e.g., 6 or more) orders of magnitude, even for small problem instances. The failure to explain intra-partition variance in problem difficulty should not, however, be viewed as a deficiency of phase transition models; phase transition research was motivated by the desire to generate difficult test problems, and this goal has been achieved.

A number of researchers have hypothesized that the 'ruggedness' of a fitness landscape is likely to be highly correlated with problem difficulty for adaptive search algorithms such as genetic and other evolutionary algorithms [Wei89, Kau93, Sta96]. A fitness landscape is said to be rugged if there is a rapid change in the fitness between nearby solutions in the landscape. If the fitness of nearby solutions is uncorrelated, we cannot expect adaptive search to outperform a random walk, i.e., there is no structure to exploit. Ruggedness is frequently quantified as the landscape correlation length, which captures the maximal distance between two arbitrary solutions for which there still exists significant correlation between their fitness values [Wei89]. We do not consider correlation length in our analyses for two reasons. First, the local search algorithms we examine are not adaptive, such that correlation length is unlikely to have a major impact on problem difficulty. Second, and more importantly, the extensive research on landscape correlation lengths indicate that for a wide range of well-known NP-hard optimization problems, the correlation length is *strictly* a function of problem size [RS01]. For example, the correlation length in an n-city TSP is given by $n/2$, while in an n-vertex Graph Bi-Partitioning Problem, it is given by $(n-3)/8$ [Sta96]. Consequently, correlation length fails to account for *any* of the variability in problem difficulty observed in sets of fixed-sized problem instances.

Another factor hypothesized to influence problem difficulty for *adaptive* local search algorithms (e.g., genetic algorithms) is the correlation between solution fitness and the distance to an optimal solution, often simply denoted as FDC [KT85, MP86, Sou86, MGSK88]. In a problem instance with high FDC, good solutions tend to be tightly clustered or, equivalently, share many solution attributes in common. Consequently, an adaptive search algorithm can exploit these similarities during search. For example, Schneider et al. [SFM$^+$96] introduce an adaptive local search algorithm for the Traveling

Salesman Problem that, after identifying the edges common to a set of high-quality local optima, restricts subsequent search to the sub-space of solutions with *only* those edges. Similarly, Sourlas [Sou86] introduced an adaptive simulated annealing algorithm for the TSP that determines those edges appearing infrequently in high-quality solutions, and prevents subsequent search from generating tours containing those edges. FDC has also been used to account for differences in the relative difficulty of problem instances, e.g., see [JF95]. As with correlation length, we do not consider FDC in the majority of our analysis because we have little reason to believe that FDC has any bearing on problem difficulty for non-adaptive local search algorithms. However, we do show in Chapter 11 that FDC can impact the performance of local search algorithms that employ re-starts – a basic form of adaptive search.

Chapter 6

The Baseline: A Random Walk

We begin our analysis of local search algorithms for the JSP by considering an algorithm based on perhaps the simplest possible navigation strategy: a pure random walk. In practice, random walks are a completely ineffective search strategy, either in an absolute sense or in comparison to local search algorithms such as simulated annealing or tabu search. The obvious question is then: Why bother analyzing the behavior of a random walk? At a fundamental level, a random walk serves as a baseline for other, more complicated navigation strategies[1]. Our goal is to move beyond simple performance-based comparisons of local search algorithms by developing a deeper understanding of the relationship between these algorithms, in terms of both the fitness landscape features that influence performance and the qualitative nature of their dynamical behavior. The relevance of the analysis presented in this chapter will become clear in later chapters, where we demonstrate surprisingly strong similarities between the cost models of a random walk and those of tabu search (Chapter 7), iterated local search (Chapter 8), and simulated annealing (Chapter 9).

This chapter is organized as follows. The random walk algorithm and various methodological issues are discussed in Section 6.1. In Section 6.2, we develop a static cost model for a random walk. The notion of sampling bias is introduced in Section 6.3, which leads into the development of a quasi-dynamic cost model. A dynamic cost model of a random walk for an idealized problem instance is introduced in Section 6.4. Dynamic cost models of a random walk on JSPs are developed in Sections 6.5 and 6.6. The relationship between the various cost models is analyzed in Section 6.7. We conclude with an analysis of the run-length distributions of a random walk in Section 6.8.

---

[1]Technically, a random walk is *not* classifi ed as a meta-heuristic, as it does not modify the behavior of some core search strategy such as next-descent.

6.1 *RW*: Algorithm Definition and Methodological Issues

As with all single-solution local search algorithms, a random walk starts from an initial solution s^* and proceeds via iterative modifications to s^*. Given a current solution s and a move operator N, a random walk assigns a uniform acceptance probability $1/|N(s)|$ to each neighbor $s' \in N(s)$ and selects a move at random. Because a random walk ignores solution fitness, the issue of escaping local optima is irrelevant. If N induces a connected search space, then a random walk is guaranteed to eventually locate a globally optimal solution, i.e., the algorithm is asymptotically complete.

We consider a random walk under the $N1$ move operator. Each trial is initiated from a random semi-active solution and consists of a sequence of iterations. At each iteration, a neighbor $s' \in N1(s)$ of the current solution s is selected at random, with s' serving as the current solution s for the next iteration. The process is iterated until an optimal solution is located. We denote the resulting algorithm by *RW*. The cost of an individual trial of *RW* is defined as the total number of iterations required to locate a global optimum, and is well-defined because the $N1$ operator induces a connected search space. Because *RW* is stochastic, we define the overall search cost for a given problem instance as either the median or mean cost (depending on the context) over 1,000 independent trials, which we respectively denote $\overline{c}$ and c_{Q2}. As we show in Section 6.8, search cost under *RW* is roughly exponentially distributed. Consequently, large sample sizes are required to achieve accurate approximations of these statistics. Empirically, both $\overline{c}$ and c_{Q2} are relatively stable when estimated from 1,000 samples, respectively varying by no more than 5% and 10% under repeated experiments.

We develop cost models of *RW* using 6×4 and 6×6 random JSPs. For our 6×4 problem set, c_{Q2} ranges from 28 to 591,774 iterations, with a mean of 4,860 iterations. For our 6×6 problem set, c_{Q2} ranges from 34 to 846,683 iterations, with a mean of 11,508 iterations. The cost distributions for both problem sets are roughly log-normal, with c_{Q2} varying over 4 to 5 orders of magnitude. Due to the relative ineffectiveness of *RW* as a search strategy, it is currently computationally intractable to assess the scalability of the cost models to larger (e.g., 10×10) problem instances, as is possible for other local search algorithms.

6.2 A Static Cost Model of *RW*

Because the behavior of *RW* is independent of solution fitness, much of the prior research on fitness landscape structure (which focuses on the structure and distribution of local optima) is irrelevant to the development of a static cost model of *RW*. Instead, we are left with the intuition that the cost required by *RW* to locate optimal solutions to problem instances is a function of (1) the size of the search space and (2) the number and/or distribution of optimal solutions within the search space, which can be codified

	Fitness Landscape Feature		
Problem Set	$log_{10}(\|optsols\|)$	$\overline{d}_{rand\text{-}rand}$	$\overline{d}_{rand\text{-}opt}$
6×4	0.4981	0.3098	0.8088
6×6	0.2179	0.3272	0.6557

Table 6.1: The r^2 values for linear regression models relating various landscape features to search cost ($log_{10}(c_{Q2})$) under *RW*.

via the following three landscape measures:

- The average distance between random (feasible) solutions ($\overline{d}_{rand\text{-}rand}$)
- The number of optimal solutions ($|optsols|$)
- The mean distance between random solutions and the nearest optimal solution ($\overline{d}_{rand\text{-}opt}$)

The $\overline{d}_{rand\text{-}rand}$ measure provides an estimate of the size of the search space, while $|optsols|$ captures the relative frequency of target solutions within this space. Although both factors likely influence search cost in *RW*, they are completely independent. In contrast, $\overline{d}_{rand\text{-}opt}$ simultaneously accounts for both factors, and is therefore likely to be more highly correlated with search cost that either of the measures upon which it is based.

In Table 6.1, we report the r^2 values for linear regression models of $log_{10}(|optsols|)$, $\overline{d}_{rand\text{-}rand}$, and $\overline{d}_{rand\text{-}opt}$ versus $log_{10}(c_{Q2})$; we use c_{Q2} instead of $\overline{c}$ because it is less sensitive to the presence of very low or high-cost trials. Both $|optsols|$ and $\overline{d}_{rand\text{-}rand}$ account for at best half of the variability in search cost. In either case, the actual search cost deviates from the predicted value by as much as 2 orders of magnitude. In contrast, $\overline{d}_{rand\text{-}opt}$ accounts for a substantial proportion in the variability in search cost. For the rectangular 6×4 problem set, only 20% of the variance remains unaccounted for, with the actual search costs deviating from the predicted value by no more than 1 order of magnitude. Accuracy is slightly worse for the square 6×6 problem set, where deviations as large as 1.5 orders of magnitude are observed. We provide scatter-plots of $\overline{d}_{rand\text{-}opt}$ versus c_{Q2} for the 6×4 and 6×6 problem sets, respectively, in Figure 6.1. Finally, we observe evidence in both problem sets that the $\overline{d}_{rand\text{-}opt}$ model is least accurate for the most difficult problem instances, i.e., those instances with large $\overline{d}_{rand\text{-}opt}$.

By focusing on only simple linear regression models, we have left open the question of whether some combination of the three features we examined could yield a more accurate static cost model. Using various model selection techniques [Coh95], we analyzed multiple regression models that simultaneously consider both multiple independent features and interactions between features. However, the accuracy of the resulting models (as measured by the model r^2) was never more than 2% higher than that of the simple $\overline{d}_{rand\text{-}opt}$ model. In part, this is due to a high level of redundancy (i.e., colinearity) between these three features, as shown in Table 6.2 for our 6×4 instances; similar results hold for our 6×6 instances.

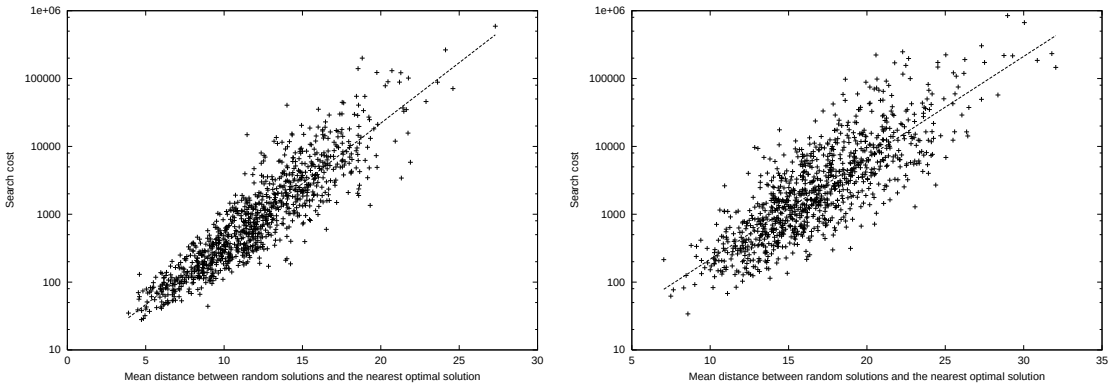

Figure 6.1: Scatter-plots of $\overline{d}_{\textit{rand-opt}}$ versus c_{Q2} for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

	$log_{10}(\lvert optsols \rvert)$	$\overline{d}_{\textit{rand-rand}}$	$\overline{d}_{\textit{rand-opt}}$
$log_{10}(\lvert optsols \rvert)$	1.0	0.0592	.7852
$\overline{d}_{\textit{rand-rand}}$	0.0592	1.0	0.5235
$\overline{d}_{\textit{rand-opt}}$	0.7852	0.5235	1.0

Table 6.2: The correlation (Pearson's) between landscape features for 6×4 random JSPs.

6.3 Sampling Bias and *RW*

In principle, it may be possible to develop more accurate static cost models of *RW* through further analysis of the fitness landscape. At the same time, it appears intuitively obvious that highly accurate cost models must take into account at least some aspects of the dynamic behavior of *RW*. To correct for the deficiencies of the $\overline{d}_{\textit{rand-opt}}$ static cost model, we proceed by analyzing the set of solutions visited by *RW during* search. Consider the distributions of (1) the distance between random solutions and the nearest optimal solution and (2) the distance between solutions actually visited by *RW* and the nearest optimal solution. Intuitively, we would expect these two distributions to exhibit strong similarities. This is often the case, as shown for a particular 6×6 instance in the left side of Figure 6.2; here, the two distributions differ only slightly, primarily in the right-tail density. However, although less common, the two distributions can differ markedly, as shown for a different 6×6 instance in the right side of Figure 6.2. We observe stronger and more frequent differences in our 6×6 problem set than in our 6×4 problem set.

Given the often dramatic differences between these two distributions, we propose the following hypothesis: search cost in *RW* is dictated by the set of solutions visited *during* search, which may be very different from random semi-active solutions. Specifically, we hypothesize that the mean distance between solutions visited by *RW* and the nearest optimal solution, which we denote $\overline{d}_{\textit{rw-opt}}$, is highly correlated with the cost re-

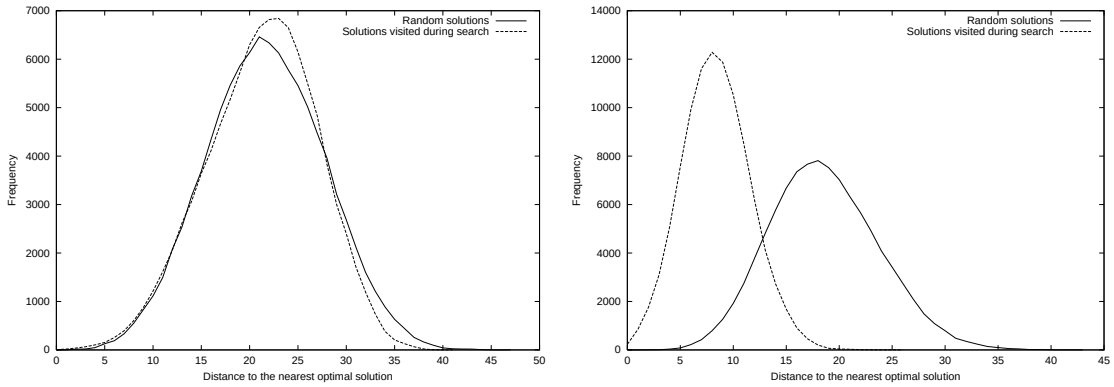

Figure 6.2: Histograms of (1) the distance between random semi-active solutions and the nearest optimal solution and (2) the distance between solutions visited by *RW* and the nearest optimal solution, for two different 6×6 random JSPs.

quired to locate optimal solutions. To test this hypothesis, we computed estimates of $\overline{d}_{\textit{rw-opt}}$ for each 6×4 and 6×6 problem instance using a set of 100,000 solutions visited by *RW* over a variable number of independent trials. Each trial is initiated from a random semi-active solution, and terminated once an optimal solution is located. The hard re-start mechanism is required because under the $N1$ move operator, situations where no moves are possible from optimal solutions can occur (see Section 3.5.2). The process is iterated, and the current trial terminated, once 100,000 solutions have been generated. Given such a large walk length, the distribution of the distance between solutions visited by *RW* and the nearest optimal solution (and hence $\overline{d}_{\textit{rw-opt}}$) is largely insensitive to the choice of initial solution. Consequently, the $\overline{d}_{\textit{rw-opt}}$ statistic is, at least empirically, isotropic.

We show scatter-plots of $\overline{d}_{\textit{rw-opt}}$ versus c_{Q2} for our 6×4 and 6×6 problem sets in the left and right sides of Figure 6.3, respectively. The r^2 values for the corresponding linear regression models are 0.9539 and 0.9074, respectively. The quasi-dynamic $\overline{d}_{\textit{rw-opt}}$ cost model is astonishingly accurate, accounting for over 90% of the variability in search cost. The results are even more surprising given that the model is based on a single summary statistic. Although it provides a significant improvement in overall accuracy, the $\overline{d}_{\textit{rand-opt}}$ cost model retains two drawbacks of the original $\overline{d}_{\textit{rand-opt}}$ model, albeit to a lesser degree. First, the model residuals can deviate from the predicted search cost by as much as an order of magnitude, especially in the 6×6 instances. Second, model accuracy appears to be inversely proportional to $\overline{d}_{\textit{rw-opt}}$, i.e., the model is least accurate for the most difficult problem instances. Third, and perhaps most importantly, the model provides little indication as to *why* $\overline{d}_{\textit{rw-opt}}$ is so highly correlated with search cost.

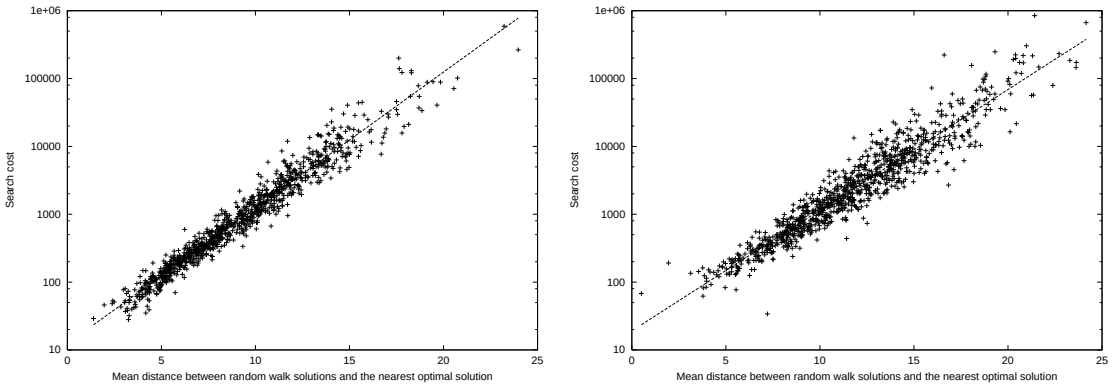

Figure 6.3: Scatter-plots of $\overline{d}_{rw\text{-}opt}$ versus c_{Q2} for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

6.4 A Dynamic Cost Model of *RW*: Preliminaries

To correct for the deficiencies of both the $\overline{d}_{rand\text{-}opt}$ and $\overline{d}_{rw\text{-}opt}$ cost models, we now turn to a more detailed analysis of the dynamical behavior of *RW*. We begin by developing a dynamic Markov model of a random walk (*not RW*) on an idealized binary (non-JSP) problem instance, and investigate the implications of this model. Later, in Sections 6.5 and 6.6, we apply a straightforward relaxation of this model to our 6×4 and 6×6 JSP problem sets.

Consider a problem instance where solutions are encoded using an n-bit representation, with a single optimal solution s^*. Further assume, in contrast to the JSP, that all 2^n solutions are feasible. Without loss of generality, we assume s^* is located at position $000 \cdots 00$ on the corresponding binary hypercube. We now consider the dynamics of a random walk under a Hamming distance-1 neighborhood operator (i.e., for any solution s there are n neighbors, each differing from s in the value of a single bit) for this idealized problem instance. Clearly, search dynamics are completely dictated by the probability of moving closer to, equidistant from, or farther away from s^*, given a current solution s. If s is Hamming distance d from s^*, then there are $\binom{d}{1} = d$ neighboring solutions that are closer to s^* and $\binom{n-d}{1} = \frac{n-i}{n}$ neighboring solutions that are farther from s^*. Under a Hamming distance-1 move operator, there are no equidistant solutions when there is a single optimal solution. Consequently, for any solution Hamming distance i from s^*, $0 \leq i \leq n$, the transition probabilities q_i and p_i of moving closer or farther away from s^* are i/n and $1 - i/n$, respectively. The probability r_i of remaining equidistant from s^* is 0, such that $\forall i \neq 0, n \; p_i + q_i = 1$[2]. Given these transition probabilities, we can model the dynamics of a random walk exactly using a

[2]The notation q_i,r_i, and p_i is borrowed from Feller's [Fel68] classic book on probability theory.

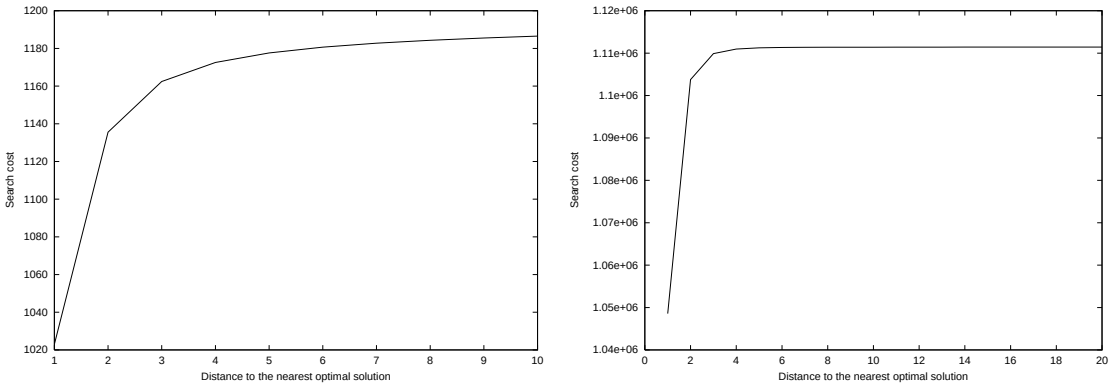

Figure 6.4: Mean cost to locate the optimal solution s^* for an 10-bit (left figure) and 20-bit (right figure) problem instance under a random walk, as a function of the distance i from the initial solution to s^*.

one-dimensional Markov chain with $n+1$ states S_i, $0 \leq i \leq n$, with transition probabilities $q_i = P(S_{i-1}|S_i) = \frac{i}{n}$ and $p_i = P(S_{i+1}|S_i) = \frac{n-i}{n}$. For $|i-j| > 1$ and $i = j$, $P(S_j|S_i) = 0$. We impose a reflecting barrier at S_n such that $P(S_{n-1}|S_n) = 1$ (solutions can be no more than Hamming distance n from s^*) and define S_0 as an absorbing state by imposing $P(S_0|S_0) = 1$ (search is terminated when s^* is located).

In the literature on probability theory, one can find a qualitatively similar Markov chain known as the Ehrenfest model (e.g., see [Fel68], p. 377). One physical interpretation of the Ehrenfest model is as a diffusion process with a central restoring force, as the probability of moving closer to (farther from) s^* is a linearly decreasing (increasing) function of the current Hamming distance from s^*. In other words, a random walk is driven toward solutions that are Hamming distance $n/2$ from s^*. In terms of probability theory, the cost required to locate the optimal solution s^* is given by the mean first passage time to S_0, or equivalently the mean time to absorption at state S_0. We denote the mean first passage time to state S_0 given an initial state S_k by v_k. Analytic formulations for v_k can be easily derived by solving a system of difference equations (we solve for the general case in which $r_i \neq 0$), and are given as follows:

$$v_k = ((\sum_{i=1}^{k-1} \theta_i) + 1)v_1 - (\sum_{i=1}^{k-1} \Phi_i) \tag{6.1}$$

where

$$v_1 = \frac{(r_n + 1)\Phi_n + 1}{(1 - r_n)\theta_{n-1}} \tag{6.2}$$

$$\theta_i = \frac{q_1 q_2 \cdots q_i}{p_1 p_2 \cdots p_i} \tag{6.3}$$

and

$$\Phi_i = \frac{q_2 \cdots q_i}{p_1 \cdots p_i} + \frac{q_3 \cdots q_i}{p_2 \cdots p_i} + \cdots \frac{q_i}{p_{i-1} p_i} + \frac{1}{p_i} \text{ for } 1 \leq i \leq n-1 \tag{6.4}$$

In Figure 6.4, we show plots of i versus v_i for $n = 10$ (left figure) and $n = 20$ (right figure). These plots clearly demonstrate that as $n \to \infty$, v_i becomes roughly constant for all but the smallest values of i. Consequently, unless the initial solution is very close to the optimal solution s^*, the choice of initial solution has little impact on search cost. For large problem instances (e.g., $n = 20$), search cost is only moderately reduced in the *best* case – when initial solutions are *adjacent* to the optimal solution. Finally, we unexpectedly observed in our experimentation that for $1 \leq n \leq 40$, $v_1 = 2^{n-1}$. In other words, a random walk out-performs enumeration of the search space *only* when search is initiated from a solution that is adjacent to the optimal solution s^*; for $i \geq 2$, a random walk is actually more costly than enumeration due to re-sampling, although not by a significant margin. For example, with $n = 19$, the mean cost from $i = 9$ is 557,757, while the size of the search space is 524,288.

6.5 A Dynamic Cost Model of *RW*: A Failed First Attempt

We now attempt to extend the dynamic cost model of a random walk for our idealized problem instance to *RW* for the random JSP. Due to the presence of both infeasible solutions and multiple optimal solutions, the transition probabilities for all solutions at a given distance i from the nearest optimal solution are in general no longer constant. While we can compute the transition probabilities for any particular solution, the question is whether or not regularities exist in these probabilities, such that we can still approximate the behavior of a Markov chain with $\mathcal{O}(2^{m \cdot \binom{n}{2}})$ states using a Markov chain with $\mathcal{O}(m \cdot \binom{n}{2})$ states.

For each 6×4 and 6×6 instance, the transition probabilities q_i, r_i, and p_i (respectively the probability of moving closer to, remaining equidistant , and moving farther from the nearest optimal solution) are estimated using a simple, albeit expensive, iterative sampling procedure. At each iteration, a random semi-active solution s is generated (using the procedure discussed in Section 3.5.3) and the neighborhood of s under the $N1$ operator is computed. We then compute the distance i to the nearest optimal solution for both s and each neighbor $s' \in N1(s)$. From these distances, we obtain the exact transition probabilities $q'_{s,i}$, $r'_{s,i}$, and $p'_{s,i}$ for the solution s under *RW*. Let S_i denote the set of solutions s observed at distance i, and let $\sum_{s \in S_i} q'_i$, $\sum_{s \in S_i} r'_i$, and $\sum_{s \in S_i} p'_i$ denote the respective sums of the transition probabilities for solutions observed at distance i.

We execute the iterative procedure until $|S_i| \geq 50$ for $1 \leq i \leq \textit{rint}(\overline{d}_{\textit{rand-opt}})$, where the function $\textit{rint}(x)$ returns the integer closest to x, rounding is upward if the fractional component of x equals 0.5. This does *not* prevent $|S_i| \geq 50$ for some $i > \textit{rint}(\overline{d}_{\textit{rand-opt}})$. Rather, the condition ensures that solutions at low-to-moderate distances from the nearest optimal solution are sufficiently sampled. Let D denote an estimate of the maximal distance between feasible solutions and the nearest optimal solution. Because $|S_i| \geq 50$ can occur for $i > \textit{rint}(\overline{d}_{\textit{rand-opt}})$, we take $D = X - 1$ such that $|S_X|$ is the smallest

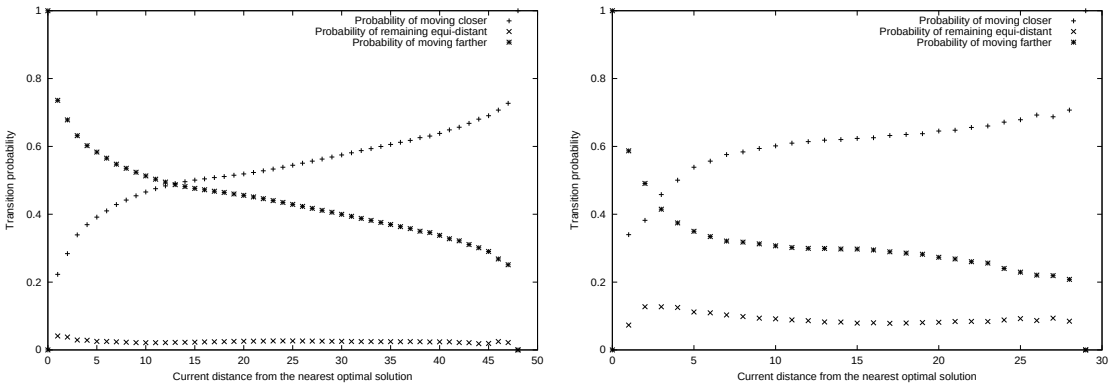

Figure 6.5: Transition probabilities for two 6×6 random JSPs under *RW* generated via sample-based estimation.

distance X at which less than 50 solutions were observed. In effect, we take $\overline{d}_{\textit{rand-opt}}$ as a lower bound on D, with the actual value of D depending on the frequency of solutions observed at distance $i > \textit{rint}(\overline{d}_{\textit{rand-opt}})$ from the nearest optimal solution. As we demonstrate below in Section 6.6, more accurate estimation of D is unnecessary. Upon termination, we compute estimates of the transition probabilities q_i, r_i, and p_i using the obvious formulas (e.g., $q_i = \sum_{s \in S_i} q'_i/|S_i|$). We observe that the estimated transition probabilities are largely insensitive to the particular set of random semi-active solutions upon which they are based. We refer to this iterative process as *sample-based* estimation.

In Figure 6.5, we show graphs of the transition probabilities resulting from sample-based estimation for two 6×6 JSPs. Clearly, there exist qualitative similarities between the transition probabilities for a random walk in the JSP and our idealized problem instance, in that there is a strong bias that guides search away from both optimal solutions and solutions that are maximally distant from optimal solutions. However, it is equally obvious that one or more of the following factors, or combination thereof, impacts the overall form of the transition probabilities: (1) the estimation process, (2) the presence of infeasible solutions, and (3) the presence of multiple optimal solutions. Specifically, the transition probabilities for many instances are highly asymmetric (e.g., see the right side of Figure 6.5) and typically exhibit slight-to-moderate curvatures in q_i and p_i for small and large i.

To quantify the accuracy of our model, we compute the predicted mean search cost $\overline{c}$ via repeated simulation of the Markov model defined by (1) the transition probabilities q_i, r_i, and p_i and (2) the set of states S_i, $0 \leq i \leq D$. We impose the boundary conditions $p_D = 0$ and $r_0 = 1$ to respectively create an reflecting barrier at distance D and an absorbing state S_0. We use simulation instead of Equations 6.1 - 6.4 in order to control for the impact of the distribution of the distance between random initial solutions and the nearest optimal solution. We initiate each simulation trial in a state S_z, where z is the distance between a random semi-active solution and the nearest optimal solution.

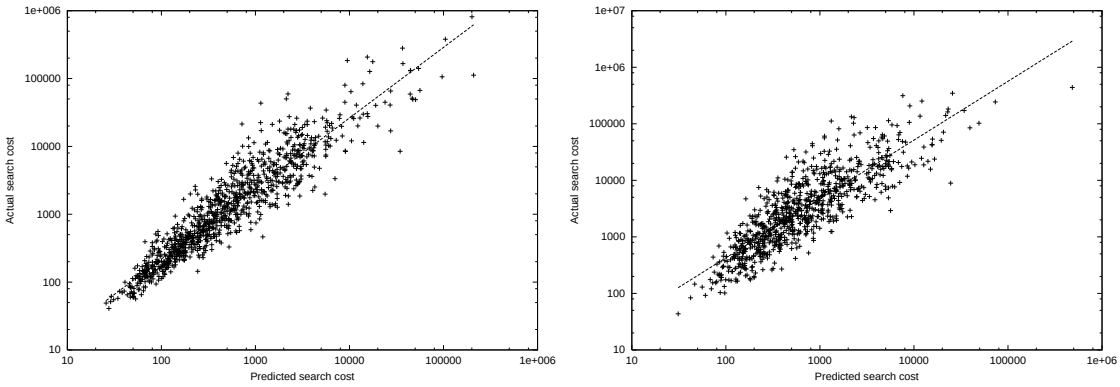

Figure 6.6: Scatter-plots of the observed versus predicted mean cost to locate an optimal solution under a random walk, for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

Simulation proceeds until state S_0 is encountered, and the total number of iterations is recorded; the predicted $\overline{c}$ is then taken as the mean number of iterations over 10,000 independent trials.

We show scatter-plots of the predicted versus actual search cost $\overline{c}$ for our 6×4 and 6×6 problem sets in Figure 6.6. The r^2 values for the corresponding regression models are 0.8701 and 0.7186, respectively. In either case, model accuracy is *worse* than that obtained by the quasi-dynamic $\overline{d}_{\textit{rw-opt}}$ model and only marginally better than that of the static $\overline{d}_{\textit{rand-opt}}$ model, despite the fact that our Markov model is based on much more comprehensive and detailed information. Further, the predicted $\overline{c}$ consistently underestimates the actual $\overline{c}$. Clearly, either the approach to aggregating solutions based on their distance to the nearest optimal solution or the process for estimating the transition probabilities fails is responsible for the relative poor accuracy of the dynamic cost model. Given the fact that random solutions are not necessarily representative of solutions visited by *RW* during search and the sample-based estimation process relies on randomly generated solutions, we now focus on the second alternative.

6.6 A Dynamic Cost Model of *RW*: Accounting for Sampling Bias

As demonstrated in Section 6.3, random semi-active solutions are not necessarily representative of the set of solutions actually visited by a random walk. This observation immediately suggests the hypothesis that transition probabilities estimated from random semi-active solutions might in fact be quite different from those estimated by solutions visited during a random walk. It is relatively straightforward to test this hypothesis, as shown below. However, the process at first appears to be somewhat circular: If we use those solutions visited during search to estimate the transition probabilities, wouldn't

we expect the corresponding Markov model to accurately predict search cost? This criticism is addressed as follows. First, the runs of *RW* used to compute the actual $\overline{c}$ are different from (i.e, independent of) the runs used to estimate the transition probabilities. Second, there is no *a priori* guarantee that there are sufficient regularities in the transition probabilities to enable a linear (one-dimensional) Markov model to accurately predict search cost. Ideally, the transition probabilities would be estimated via sampling procedure that generated solutions representative of those visited during search. However, given the current lack of understanding of this search space, and how it differs from the space of random semi-active solutions, *RW* remains the only viable alternative for generating such solutions.

We refer to the process of computing transition probabilities from the set of solutions visited by a search algorithm as *on-line* estimation. We observe *RW* over a number of independent trials, computing both (1) the set of solutions S_i observed at each distance i from the nearest optimal solution and (2) the aggregate transition probabilities $\sum_{s \in S_i} q'_i$, $\sum_{s \in S_i} r'_i$, and $\sum_{s \in S_i} p'_i$ for solutions distance i from the nearest optimal solution. The method for determining when to terminate the sampling process is identical to that described above in Section 6.5, as are the formulas for computing both D and the transition probabilities. As with sample-based estimation, the transition probabilities generated via on-line sampling are empirically insensitive to the nature of the random walks performed, i.e., the estimates are statistically isotropic.

In Figure 6.7, we show graphs of the transition probabilities resulting from on-line estimation for two typical 6×6 random JSPs. In contrast to Figure 6.5, we observe very strong similarities with the transition probabilities predicted by our idealized problem instance. If we set $r_i = 0$ and scale q_i and p_i accordingly, the transition probabilities shown in the left side of Figure 6.7 are nearly identical to those of our idealized problem instance. The transition probabilities shown in the right side of Figure 6.7 exhibit both a slight curvature in q_i and p_i for small i and negligible r_i for all i.

The differences between the transition probabilities resulting from the sample-based and on-line estimation procedures can be significant, as shown in Figure 6.8. For this particular 6×6 instance, the sample-based transition probabilities predict that search is biased toward solution roughly distance 10 from the nearest optimal solution, in contrast to a predicted distance of 25 under on-line estimation. Differences in this crossover point can be found throughout our 6×4 and 6×6 problem set, indicating the potential for divergence in the predicted search cost generated by the Markov models based on the two sets of probabilities.

Scatter-plots of the predicted versus observed $\overline{c}$ for 6×4 and 6×6 random JSPs are shown in Figure 6.9. The r^2 values of the corresponding regression models are 0.9954 and 0.9929, respectively. In either case, model accuracy is exceptionally high; for all problem instances, the predicted $\overline{c}$ is within a factor of 4 of the actual $\overline{c}$, and is typically within a factor of 2. These results confirm that the initial attempt to construct an accurate dynamic model failed due to the fact that the transition probabilities for random semi-active solutions are not representative of those for solutions visited by *RW*

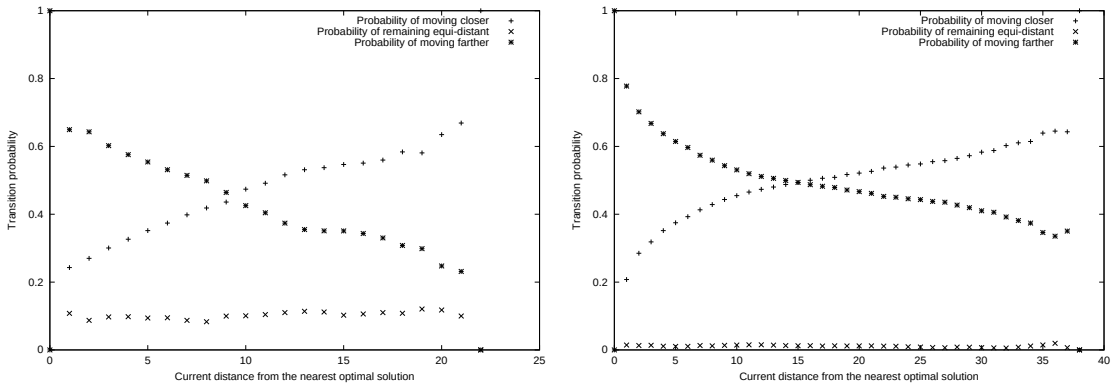

Figure 6.7: Transition probabilities for two 6×6 random JSPs under *RW* generated via online estimation.

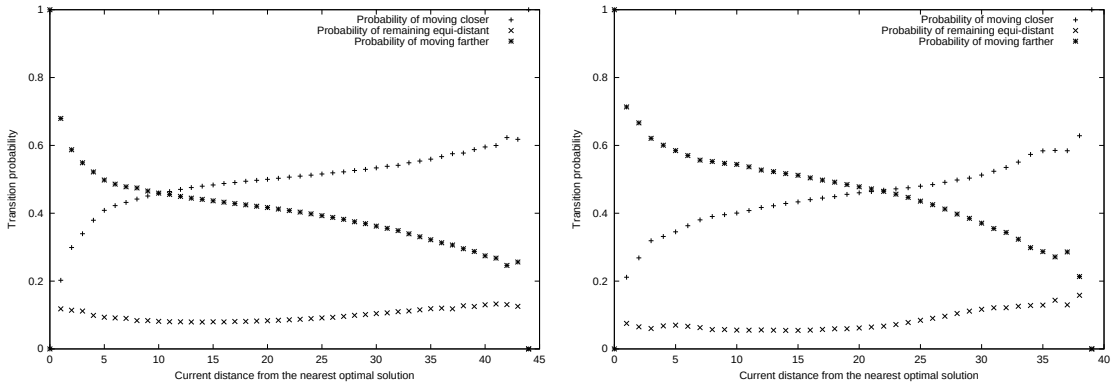

Figure 6.8: Transition probabilities generated via sample-based estimation (left figure) and on-line estimation (right figure) for an identical 6×6 random JSP.

during search. Finally, although we do not know the exact cause of the curvatures in the transition probabilities (e.g., that occur for the instance represented in the right side of Figure 6.7), we do know that they are strictly a function of the distribution of infeasible solutions: similar curvatures were observed in a set of 100 6×6 random JSPs containing a single optimal solution apiece.

Finally, we analyze the influence of the initial solution on the cost of locating an optimal solution under *RW*. In Figure 6.10, we show plots of the distance i from the initial solution to the nearest optimal solution versus the search cost v_i for the two 6×6 represented in Figure 6.5. As with our idealized problem, we observe minimal differences in search cost for nearly all initial solutions, with the exception of those with small i. Further, the effect is more pronounced in the more difficult instances, e.g., those with large D.

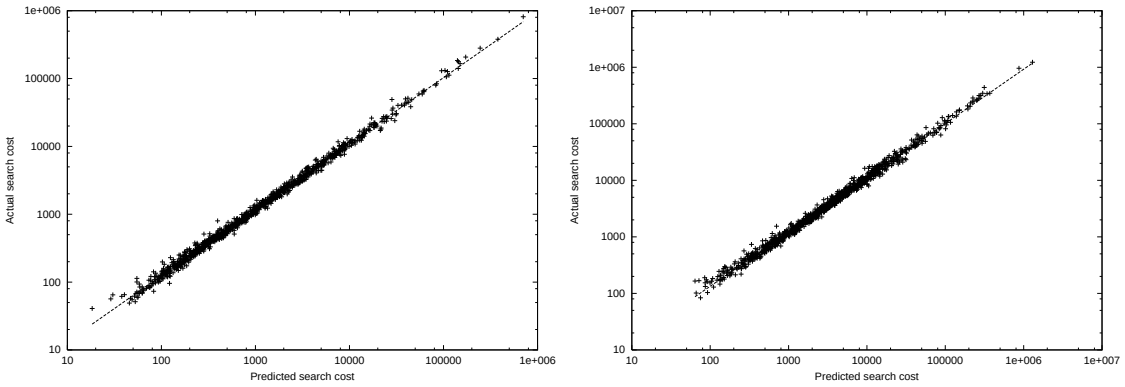

Figure 6.9: Scatter-plots of the observed versus predicted mean cost to locate an optimal solution under a random walk, for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

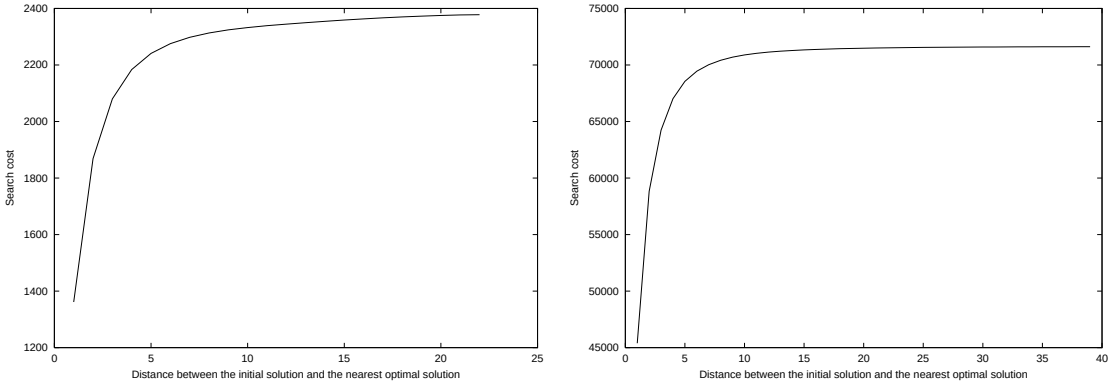

Figure 6.10: Mean cost for a random walk to locate an optimal solution, given an initial solution that is distance i from the nearest optimal solution, for two 6×6 random JSPs.

6.7 Relationship Between the Cost Models of *RW*

In hindsight, the success of the $\overline{d}_{\textit{rand-opt}}$ cost model was due to the fact that $\overline{d}_{\textit{rand-opt}}$ and $\overline{d}_{\textit{rw-opt}}$ are highly correlated for small problem instances. What remains is to establish a link between the $\overline{d}_{\textit{rw-opt}}$ model and our dynamic Markov model. As previously noted, the qualitative forms of the estimated transition probabilities (e.g., see Figure 6.7) are identical for all of the problem instances we examined. Any major differences are due to variability in D, which (like $\overline{d}_{\textit{rw-opt}}$) can be viewed as a measure of the size of the search space. We also observe that these transition probabilities are roughly symmetric around $D/2$, and that search in *RW* is biased toward solutions that are approximately distance $D/2$ from the nearest optimal solution. But $D/2$ is roughly equivalent to $\overline{d}_{\textit{rw-opt}}$, and consequently $\overline{d}_{\textit{rw-opt}} \approx D/2$. Thus, we believe the success of the $\overline{d}_{\textit{rw-opt}}$ model is due to the fact that it estimates a key parameter (D) of the dynamic cost model.

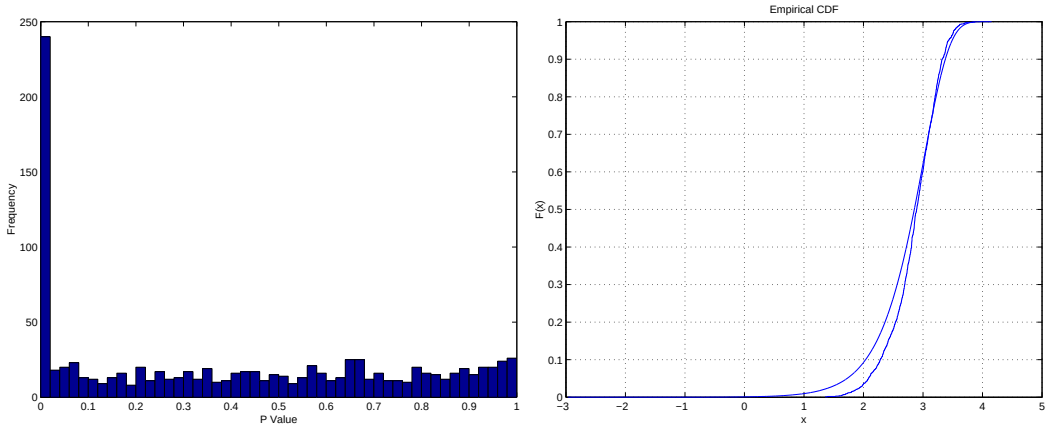

Figure 6.11: Left Figure: p-values for 1,000 6×6 instances for rejecting the null hypothesis that the actual run-length distributions are exponentially distributed. Right Figure: The actual and exponential run-length distributions for the 6×6 instance with the smallest p-value (p=3.6e-16).

6.8 An Analysis of Run-Length Distributions of *RW*

We now shift our focus from modeling the average cost required to locate optimal solutions to modeling the full run-length distribution (RLD) of search cost. As indicated in Section 6.1, pilot experiments indicated that search cost under *RW* is exponentially distributed. We use a two-sample Kolmogorov-Smirnov (KS) goodness-of-fit test [SM90] to test this hypothesis for each of our 6×4 and 6×6 problem instances. We take the first sample as the distribution of actual search costs c observed for *RW* over 1,000 independent trials. The second sample consists of 1,000,000 random samples from an exponential distribution with a mean equal to the $\overline{c}$ computed from the first sample. We only report results for the 6×6 instances; results for the 6×4 problem set are qualitatively similar.

We show a histogram of the resulting p-values for our 6×6 instances in the left side of Figure 6.11; the null hypothesis is that the two samples originated from the same underlying distribution. At $p \leq 0.01$, we reject the null hypothesis for 218 of the 1,000 instances: i.e., for roughly 20% of the 6×6 instances, search cost under *RW* is not exponentially distributed. We show the cumulative density function (CDF) of both the actual and associated exponential distributions for the 6×6 instances with the smallest p-value in the right side of Figure 6.11. For this instance, and all other instances with $p \leq 0.01$, the two distributions differ largely in their left tails. As shown in Figure 6.12, the value of the KS test statistic is inversely proportional to instance difficulty (the p-value is proportional to instance difficulty). In other words, the run-length distribution of *RW* is exponential for moderate-to-difficult instances, but not for easy instances. Although statistically significant, our results also indicate that the maximal deviation from the exponential distribution is not substantial; the run-length distribution under *RW* can still

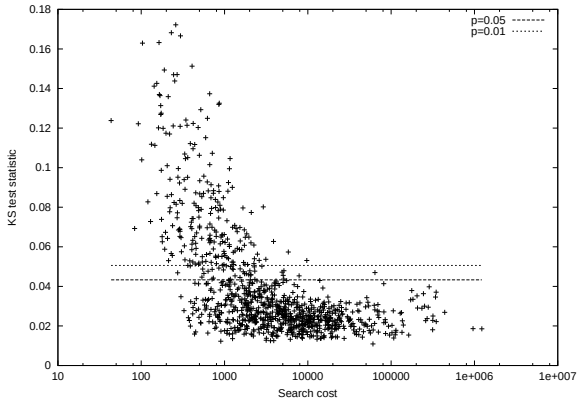

Figure 6.12: Scatter-plot of search cost versus the value of the Kolmogorov-Smirnov test statistic for comparing the actual search cost distribution with that of an exponential distribution. Large values of the test statistic indicate more significant differences. The horizontal lines indicate null hypothesis rejection thresholds at significance $p = 0.01$ and $p = 0.05$.

be approximated by an exponential.

Surprisingly, the results presented above (and in particular those shown in Figure 6.12), mirror those reported by Hoos [Hoo98] for a well-known local search algorithm, Walk-SAT, for MAX-SAT. Search in Walk-SAT operates in two phases: a hill-climbing phase and a plateau phase [SGS00]. During the hill-climbing phase, search is quickly driven from poor initial solutions to near-optimal solutions. Subsequent plateau search is restricted to the sub-space containing optimal and near-optimal solutions. In moderate-to-difficult MAX-SAT instances, the plateau phase dominates search cost; in easy instances, the cost of the two phases is similar. Hoos hypothesized that a costly hill-climbing phase, relative to the plateau phase, caused Walk-SAT RLDs to deviate from the exponential ideal. He then introduced a generalized distribution to model the hill-climbing phase, and showed the distribution more closely approximates the Walk-SAT RLDs observed for easy problem instances. While our results are consistent with those reported by Hoos, they also indicate that the deviation from the exponential ideal may be caused by factors other than initial hill-climbing during local search, which is clearly absent in *RW*.

Next, we compare the actual RLDs under *RW* to those predicted by our dynamic cost model. We again use a two-sample Kolmogorov-Smirnov goodness-of-fit test to the hypothesis that the actual and predicted RLDs for a given problem instance originate from the same underlying distribution. We only report results for our 6×6 problem set; the results for 6×4 instances were qualitatively identical. For 647 of the 1,000 instances, the significance (i.e., p-value) of the resulting KS test statistic was ≤ 0.01. In contrast to Figure 6.12, we observed no significant correlation between the value of the KS test statistic and problem difficulty. We show CDFs of the predicted and actual RLDs for two problem instances in Figure 6.13. With very few exceptions, the actual

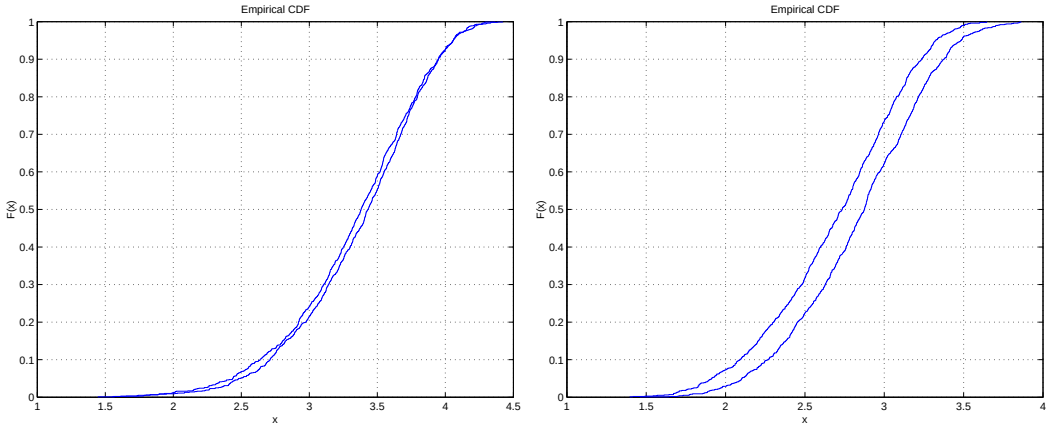

Figure 6.13: CDFs of the predicted and actual RLDs for two 6×6 instances. The p-values for the KS test statistic are respectively 0.14408 and $7.57e-8$.

and predicted RLDs are qualitatively identical; for instances with $p \leq 0.01$, the actual RLDs are closely approximated by shifting the predicted RLDs along the x-axis by a constant value. Consequently, we believe such discrepancies are due to inaccuracies in the process of estimating the transition probabilities, and not to some inherent structural flaw in our dynamic model. As expected (see Section 4.5), our results indicate that near-perfect dynamic cost models are necessary to achieve statistically accurate prediction of the full RLD.

Chapter 7

Tabu Search

The job-shop scheduling problem is widely acknowledged as one of the most difficult NP-hard combinatorial optimization problems encountered in practice. Nearly all well-known optimization and approximation techniques have been applied to the JSP, including linear programming, Lagrangian relaxation, branch-and-bound, constraint satisfaction, local search, and even neural networks and expert systems [JM99]. Recent comparative studies of techniques for the JSP conclude that local search algorithms provide the best overall performance on the set of widely-available benchmark problems, e.g., see the recent surveys by Blażewicz et al. [BDP96] or Jain and Meeran [JM99]. Within the class of local search algorithms, the strongest performers are typically derivatives of tabu search, an exception being the guided local search algorithm of Balas and Vazacopoulos [BV98]. The current state-of-the-art algorithm for locating high-quality solutions to the JSP, outperforming competitors by a significant margin, is a tabu search algorithm developed by Nowicki and Smutnicki [NS03].

Despite the relative simplicity and excellent performance of tabu search algorithms for the JSP, very little is known about *why* these algorithms work so well, and under what conditions. For example, we currently have no answers to fundamental, related questions such as "Why is one problem instance more difficult than another?" and "What features of the fitness landscape influence search cost?" No published research has analyzed models of problem difficulty for tabu search in the JSP, and only one group of researchers, Mattfeld et al. [MBK99], has analyzed the link between problem difficulty and local search for the JSP in general.

In this chapter, we develop a variety of cost models of a "representative" tabu search algorithm for the JSP, introduced by Taillard in 1989. Although no longer state-of-the-art, Taillard's algorithm is closely related to modern tabu search algorithms, but for a variety of reasons is significantly more amenable to analysis (although we do analyze a more powerful variant of Taillard's algorithm in Chapter 11). Additionally, we show that the resulting cost models account for many observations relating to problem difficulty in the JSP, and use the models to clarify some "well-known" facts regarding the behavior of tabu search algorithms. This rest of this chapter is organized as follows.

We begin in Section 7.1 with a general introduction to the tabu search meta-heuristic. The history of tabu search and the JSP is documented in Section 7.2, and we discuss the details of Taillard's tabu search algorithm in Section 7.3. Our hypothesis that tabu search algorithms for the JSP are effectively performing a random walk over a restricted sub-space of the fitness landscape is based in part on an analysis of run-time behavior that we present in Section 7.4. We analyze the accuracy of numerous static cost models for Taillard's algorithm in Section 7.5, where we demonstrate that the mean distance between random local optima and the nearest optimal solution ($\overline{d}_{lopt\text{-}opt}$) is highly correlated with problem difficulty. We explore applications and limitations of the $\overline{d}_{lopt\text{-}opt}$ model in Sections 7.6 and 7.7. We turn from static to quasi-dynamic cost models in Sections 7.8 and 7.9, where we show that random local optima are not necessarily representative of the set of solutions visited during search, and use this information to construct a more accurate, scalable quasi-dynamic cost model. We develop our dynamic cost model in Section 7.9, and discuss several applications of the resulting model in Section 7.10. The relationship between the static, quasi-dynamic, and dynamic cost models is explored in Section 7.11. We discussion of the impact of neighbor makespan estimation on our dynamic cost model (Section 7.12) and the ability of the dynamic cost model to account for the full run-length distribution (Section 7.13).

7.1 An Overview of Tabu Search

Basic forms of most meta-heuristics for local search, including iterated local search, simulated annealing, and Markov Chain Monte Carlo, are *memoryless*: the neighbor s' of the current solution s selected at each iteration is independent of the set of solutions encountered in previous iterations. Yet, intuition suggests that memory might be leveraged in a fundamental way to improve meta-heuristic performance. For example, long-term memory may be used to prevent search from re-visiting previously explored regions of the fitness landscape, while short-term memory may be used to escape attractor basins by preventing search from re-visiting recently generated solutions. These and other memory mechanisms are embodied in the *tabu search* meta-heuristic, first introduced by Glover in 1986 [Glo86]. Meta-heuristics such as iterated local search and simulated annealing exhibit relatively little variance across implementations. In contrast, tabu search is really a *framework* for expressing memory-based meta-heuristics: numerous design decisions are required to produce a realizable implementation of tabu search for a given combinatorial optimization problem, each with the potential to have a major impact on performance. Finally, some variants of iterated local search and simulated annealing *do* make limited use of memory, e.g., see Yamada et al. [YRN94] or Ingber and Rosen [IR92]; however, memory is not an integral component of the core meta-heuristic.

Like all single-solution meta-heuristics for local search, tabu search proceeds via a sequence of incremental modifications to some initial solution s_{init}. We denote the

```
function Tabu Search(MaxIters )
        s = s_init
        s_best = s_init
        clear(M)
        i = 0
        repeat
                update(M,s)
                n_all = N(s)
                n_tabu=tabu(n_all,M)
                s = argmin_{x∈n_all\n_tabu}(F(x))
                if F(s) < F(s_best) then
                        s_best = s
                end
                i = i + 1
        until i eq MaxIters
        return s_best
end
```

Figure 7.1: Pseudo-code for the tabu search meta-heuristic. See text for details.

solution at iteration i by X_i, $0 \leq i \leq n$, where $X_0 = s_{init}$ and n is the total number of iterations performed; the entire sequence is denoted simply by X. Upon termination, the best solution in the X is returned. All tabu search algorithms use steepest-descent as the core meta-heuristic. A short-term memory $M(X, k)$ stores some subset of the features of solutions in X encountered during the last k iterations. During each iteration X_i, $i < N$, the neighbors $N(s)$ of the current solution s are generated in the normal fashion. However, before selecting a particular neighbor s' for the next iteration, the short-term memory $M(X, k)$ is used to identify a subset of *tabu* solutions $T \subseteq N(s)$ that share similarities with recently encountered solutions. Tabu search then proceeds by selecting the *best* non-tabu solution from $N(s) \backslash T$; ties are typically broken at random. It should be clear that tabu search behaves identically to steepest-descent local search, with two key exceptions: only non-tabu moves are considered at each iteration, and a move is *always* taken, i.e., search does not terminate at a local optimum. While search is descending toward a local optimum, the short-term memory is largely passive, simply tracking attributes of the solutions encountered. However, once a local optimum is encountered and search begins to accept dis-improving moves, the short-term memory prevents search from "slipping" back down into the attractor basin.

We provide pseudo-code for the basic form of tabu search in Figure 7.1, which assumes the existence of a move operator N and fitness function F. Beyond this, an algorithm designer must supply definitions for only three functions to yield a complete implementation of tabu search. The first function, *clear*, empties the contents of short-

term memory and initializes any associated auxiliary data structures. The second function, *update*, updates the contents of the short-term memory M to reflect the generation of a new solution s. The third function, *tabu*, uses the short-term memory to determine those solutions in the input set that are forbidden.

Thus far, we have been intentionally vague regarding any details of short-term memory. In practice, the contents of short-term memory, and how those contents are used to identify tabu neighbors, is problem-specific and, as a consequence, highly variable. As indicated above, the purpose of short-term memory is to keep search from re-visiting solutions encountered in recent iterations by labeling some solutions as tabu, with the ultimate goal of escaping local optima. For example, the short-term memory of a tabu search algorithm for MAXSAT might track the set of variables that were inverted over the last k iterations, and label as tabu any solution obtained from the current solution by inverting the value of any such variable. This is an example of *attribute-based* short-term memory. Alternatively, the short-term memory may simply store the last k solutions, and prevent search from re-visiting one of these solutions. Such a scheme is an instance of *solution-based* short-term memory. In practice, attribute-based memory is more common, although for any given problem there are a wide variety of possible solutions attributes that can be extracted and stored in short-term memory, each with its own potential impact on performance.

The use of long-term memory is even less standardized. Relatively few long-term memory mechanisms are found in even a majority of tabu search implementations. Further, long-term memory is frequently absent, even in high-performance implementations. The sole exception is the widespread use of an *aspiration level* criterion, which over-rides the tabu status of any neighboring solution with a better fitness that the best solution encountered during the current run. Numerous tabu search algorithms re-intensify search around previously encountered high-quality solutions, while frequency-based memory is often used to penalize solutions in proportion to the frequency with which their attributes appear in the solutions encountered during prior iterations. Tabu search can be prone to cyclic behavior that cannot be detected via short-term memory. Consequently, cycle detection mechanisms can also be viewed as long-term memory, and are found in many tabu search implementations. Detailed descriptions of these and other long-term memory mechanisms can be found in [GL97].

Many of the ideas presented above first appeared in a pair of introductory articles by Glover [Glo89] [Glo90]. Tabu search algorithms have been successfully applied to a truly impressive range of combinatorial optimization problems, e.g., see the book by Glover and Laguna [GL97] for a slightly dated survey. However, the popularity of tabu search is largely concentrated in the operations research community, and only recently has drawn the attention of researchers in artificial intelligence.

7.2 Tabu Search and the JSP: An Historical Perspective

Taillard introduced the first tabu search algorithm for the JSP in 1989 [Tai89], a relatively straightforward implementation of tabu search based on the $N1$ move operator. Despite its simplicity, the algorithm out-performed, in terms of both speed and accuracy, contemporary state-of-the-art algorithms such as van Laarhoven et al.s simulated annealing algorithm [vLAL88] and Applegate and Cook's shifting bottleneck procedure [AC91]. The strong relative performance of Taillard's algorithm fueled a subsequent explosion of interest in tabu search algorithms for the JSP. The results of the "first wave" of follow-on research yielded the algorithms by Dell'Amico and Trubian [DT93], Barnes and Chambers [BC95, CB96], Nowicki and Smutnicki [NS96], in addition an enhanced version of Taillard's original algorithm [Tai94]. Of these, Nowicki and Smutnicki's algorithm provides the strongest overall performance, which is capable of locating optimal solutions to Fisher and Thompson's infamous `ft10` instance in less than 30 seconds on now-dated hardware based on an Intel 386 processor. At the time, Nowicki and Smutnicki's algorithm outperformed all other algorithms for the JSP, including other local search algorithms and the best available branch-and-bound algorithms. Although rarely noted in the literature, the algorithms of Barnes and Chambers are nearly competitive with Nowicki and Smutnicki's algorithm.

Taillard's original algorithm and the aforementioned follow-on algorithms differ primarily in the following three respects: the method used to generate the initial solution, the move operator, and the use of long-term memory. There is some evidence to suggest that the quality of the initial solution can have an impact on the performance of tabu search. Consequently, researchers have hybridized tabu search algorithms with methods to generate high-quality initial solutions. All of the first-wave tabu search algorithms for the JSP employ different methods for generating the initial solution; the greedy constructive procedure [WW95] used in Nowicki and Smutnicki's algorithm yields the best initial solutions. The algorithms of both Barnes/Chambers and Taillard are based on the $N1$ move operator, although the latter only estimates the makespan of neighboring solutions (using the procedure documented in Section 3.5.2). Both Dell'Amico/Trubian's and Nowicki/Smutnicki's algorithms are based on novel move operators; the former simultaneously inverts pairs of critical operations, while the latter uses a restricted variant of the $N1$ operator, $N5$. Finally, both Barnes/Chambers and Nowicki/Smutnicki track the best solutions located during search, periodically re-intensifying search around these solutions.

Non-systematic variation among these algorithms makes it very difficult to determine which components, or combinations of components, dictate the success of a given algorithm, and to what degree. All of the follow-ons to Taillard's original algorithm differ in at least two out of the three primary features mentioned earlier. For example, Nowicki and Smutnicki simultaneously introduce a new initialization method, a new move operator, and a new re-intensification method. Although high-quality solutions can improve the effectiveness of search, both Taillard [Tai94] and Nowicki/Smutnicki

[NS96] indicate that tabu search can eventually over-come the impact of a poor choice of initial solution, i.e., the quality of the initial solution appears to only impact short-term performance. Therefore it seems likely that the variability in performance of different tabu search algorithms for the JSP is dictated by the choice of move operator and whether re-intensification is employed. The two most effective algorithms, those of Barnes/Chambers and Nowicki/Smutnicki, use similar re-intensification schemes, but different move operators. Consequently, we hypothesize that re-intensification is largely responsible for the superior performance of these algorithms, which is impacted to a lesser extent by the move operator (we test this hypothesis in Chapter 11).

Since the introduction of Nowicki and Smutnicki's algorithm, researchers have concentrated on enhancing the use of memory in tabu search algorithms for the JSP. Jain [Jai98] extends Nowicki and Smutnicki's original algorithm via a core/shell strategy. The "core" is a simple variant of Nowicki and Smutnicki's algorithm that does *not* re-intensify search around previously encountered high-quality solutions. Re-intensification is controlled by a series of *shell* strategies, which apply the core procedure to both previously visited high-quality solutions and new high-quality solutions generated by various external procedures, specifically path relinking and scatter search. Pezzella and Merelli [PM00] describe a tabu search algorithm that uses the shifting bottleneck procedure [ABZ88] to re-optimize the machine sequences whenever a new best-so-far solution is located, which in effect intensifies search around high-quality solutions. Most recently, Nowicki and Smutnicki [NS01, NS02, NS03] have extended their original algorithm by using path relinking between previously encountered high-quality solutions to generate new solutions to which their core procedure is then applied. The latter approach represents, by a significant margin, the current state-of-the-art in approximation algorithms for the JSP.

7.3 Algorithm and Methodological Considerations

In developing cost models of tabu search for the JSP, we necessarily proceed in the context of a specific algorithm. For our analysis, we selected an algorithm introduced by Taillard in 1994 [Tai94] (as opposed to an algorithm documented by Taillard in an earlier technical report [Tai89]). We implemented our own version of the algorithm, which we denote $TS_{Taillard}$, and were easily able to replicate the performance results documented in [Tai94]. We acknowledge up-front that $TS_{Taillard}$ is *not* a state-of-the-art tabu search algorithm for the JSP; the algorithms of Nowicki/Smutnicki, Pezzella/Merelli and Barnes/Chambers provide stronger overall performance. However, state-of-the-art algorithms still bear strong resemblance to $TS_{Taillard}$, differing primarily in the choice of move operator, the method used to generate the initial solution, and the presence of long-term memory mechanisms such as reintensification. Instead of tackling the most complex algorithms first, our goal is to develop cost models for a simple "representative" tabu search implementation for the JSP, and then to *systematically* assess the

influence of more complex algorithmic features on the cost models of the basic algorithm. In particular, we examine the impact of the initial solution in Section 7.10.3 and re-intensification in Chapter 11; for reasons discussed below, we are currently unable to assess the impact of move operators found in state-of-the-art algorithms.

TS$_{Taillard}$ is based on the $N1$ move operator. In contrast to many other algorithms based on $N1$, *all* pairs of adjacent critical operations are considered – not just those on a single critical path. At each iteration, the neighborhood of the current solution s is evaluated, and the non-tabu neighbor $s' \in N1(s)$ with the smallest makespan is selected for the next iteration; ties are broken randomly. Let (o_i, o_j) denote the pair of adjacent critical operations that are swapped in s to generate s', such that $ect_i = est_j$, i.e., o_i appears before o_j in the machine order of s. The short-term memory *STM* in *TS$_{Taillard}$* is a queue of maximal size L, containing pairs of critical operations (o_i, o_j). The current size of the queue is denoted $|STM|$. At the end of each iteration, the *inverse* pair (o_j, o_i) is added to $|STM|$; if $|STM| > L$, then the operation pair at the front of the queue is removed. The idea, a variant of frequency-based memory, is to prevent recently swapped pairs of critical operations from being re-established. Clearly, the oldest and youngest tabu moves reside near the front and back of the queue, respectively. Consider a neighboring state $s' \in N1$ of a current solution s, obtained by swapping the order of a pair of critical operations (o_i, o_j). The state s' is tabu with respect to *STM* if and only if $(o_j, o_i) \in STM$. An exception occurs when $C_{max}(s') < C_{max}(s_{best})$, i.e., *TS$_{Taillard}$* employs a simple aspiration level criterion. In rare cases, the minimal makespan may be achieved by both non-tabu and tabu-but-aspired moves, in which case a non-tabu move is always accepted. Similarly, situations occur when all moves in $N1(s)$ are tabu, in which case the oldest tabu moves are removed from the front of *STM* until at least one neighbor in $N1(s)$ in non-tabu or tabu-but-aspired. Taillard did not specify methods for handling either of the latter two exceptions in his original papers [Tai89, Tai94].

It is well-known that tabu search implementations with fixed-size short-term memories are prone to cycling behavior [GL97]. To avoid cycling, *TS$_{Taillard}$* randomly and uniformly samples the tabu tenure L (i.e., the maximal size $|STM|$ of the short-term memory *STM*) from a fixed-width interval $[L_{min}, L_{max}]$ every $1.2L_{max}$ iterations. Tabu tenure can have a major impact on performance. Based on empirical tests, Taillard defines $L_{min} = 0.8X$ and $L_{max} = 1.2X$, where $X = (n + m/2) \cdot e^{-n/5m} + N/2 \cdot e^{-5m/n}$; n and m are respectively the number of jobs and machines in the problem instance under consideration, and $N = nm$.

Our implementation of *TS$_{Taillard}$* deviates from Taillard's original algorithm in several respects; in all cases, our decision was based on the need to control for the impact of a particular feature on the form and/or accuracy of our cost models. Taillard's original algorithm [Tai94] used the scheme described in Section 3.5.2 to *estimate* the makespan of neighboring solutions. There is some evidence (e.g., see Nowicki and Smutnicki [NS02]) that estimation can impact the performance of tabu search algorithms. We instead compute the makespan of neighboring solutions exactly. We defer analysis of the impact of estimation on our cost models to Section 7.12. We do not use the frequency-

based memory introduced by Taillard in our implementation of $TS_{Taillard}$. As Taillard indicates ([Tai94], p. 100), long-term memory is only necessary for problems that require a very large (> 1 million) number of iterations, which is generally not the case for the test problems we consider. Finally, in contrast to Taillard, who used a deterministic scheme to generate initial solutions, we initiate all runs of $TS_{Taillard}$ from random local optima. We use random local optima in contrast to random solutions to control for the length of the initial descent in our analysis. The impact of the quality of initial solution on performance is analyzed in Section 7.10.3.

An important property of $N1$ is that it induces search spaces that are provably connected, in that it is always possible to move from an arbitrary solution to a global optimum. Consequently, it is *possible* to construct a local search algorithm based on $N1$ that is asymptotically complete, such that it will eventually locate an optimal solution given sufficiently large run-times. Empirically, $TS_{Taillard}$ is asymptotically complete given the aforementioned rules for specifying the tabu tenure and update frequency; all trials of $TS_{Taillard}$ have eventually located an optimal solution to all of our 6×4, 6×6, and 10×10 problem instances; the convergence on larger instances was not tested due to the computational costs. The property of asymptotic completeness allows us to naturally define the cost required to solve a given problem instance using a *single* trial of $TS_{Taillard}$ as the number of iterations required to locate a globally optimal solution. In general, search cost is a random variable with an approximately exponential distribution - see Taillard [Tai94] or Section 7.13. Consequently, we define the search cost for a given problem instance as either the median or mean number of iterations required to locate an optimal solution; the respective quantities are denoted by c_{Q2} and $\overline{c}$. We estimate both c_{Q2} and $\overline{c}$ using 5,000 independent trials of $TS_{Taillard}$; due to the exponential nature of the distribution, a large number of samples is required to achieve stable estimates. We use the more stable c_{Q2} measure when possible. Other tabu search algorithms for the JSP, including Nowicki and Smutnicki's algorithms, are based on move operators that induce disconnected search spaces and are therefore *not* asymptotically complete. A major impediment to the analysis of these algorithms is the definition of search cost, as they are not guaranteed to locate optimal solutions. Our selection of $TS_{Taillard}$ as a "representative" tabu search algorithm for the JSP is driven in part by the desire to avoid this issue.

In preliminary experimentation, we observed that the tabu tenure heuristic defined by Taillard was insufficient to prevent cycling or stagnation behavior in instances from our 6×4 and 6×6 problem sets; the respective $[L_{min}, L_{max}]$ tenure intervals are computed as $[3, 5]$ and $[4, 6]$. To avoid such behavior, we set $[L_{min}, L_{max}]$ equal to $[6, 14]$ for all trials involving these instances, and re-sample the tabu tenure every 15 iterations. We observed similar, albeit less extreme, behavior on 10×10 instances under Taillard's heuristic. Here, we set $[L_{min}, L_{max}]$ equal to $[8, 14]$ for all trials involving these instances, and re-sample the tabu tenure every 15 iterations. These values were taken from Taillard, who also ignored his heuristic for trials involving 10×10 instances. For all other problem sizes, we compute the tabu tenure intervals and update frequency using Taillard's

		Depth under $TS_{Taillard}$				Mean Depth of
Size	Instance	Median	Mean	Std. Dev	Max.	Random Solutions
10×10	la16	1	1.84	1.78	20	12.43
10×10	la17	1	1.96	1.86	16	13.03
10×10	la18	2	2.06	1.94	18	13.42
10×10	la19	2	2.15	1.94	18	13.99
10×10	la20	2	2.22	1.94	20	13.00
10×10	abz5	2	2.16	1.95	16	13.96
10×10	abz6	2	2.12	1.89	16	12.95
15×15	ta01	1	1.95	1.97	18	24.38
20×15	ta11	1	1.51	1.83	21	30.07
20×20	ta21	2	2.28	2.24	24	34.10
30×15	ta31	1	2.00	2.20	20	39.27
30×20	ta41	1	1.68	2.02	36	46.19
50×15	ta51	1	1.86	2.35	28	45.11
50×20	ta61	2	2.25	2.56	60	55.50
100×20	ta71	1	2.50	3.16	40	71.26

Table 7.1: Depth statistics for $TS_{Taillard}$ on select random JSPs from the OR Library. Statistics are taken over a single run of length 1,000,000 iterations.

heuristic.

7.4 Run-Time Behavior: Some Qualitative Observations

Given a core strategy of steepest-descent local search, tabu search algorithms exhibit a strong bias search toward local optima. Coupling this bias with the relative weakness of attractor basins in the JSP, demonstrated in Section 5.2.4, we hypothesize that tabu search in general, and $TS_{Taillard}$ in particular, is restricted to the sub-space of the fitness landscape containing both local optima and solutions that are "nearly" local optima – in terms of the number of moves, and *not* the difference in fitness.

To test this hypothesis, we observe the *depth* of solutions visited by $TS_{Taillard}$ during search on a range of problem instances. We define the search depth with respect to the current solution s as the disjunctive graph distance $\overline{D}(s, s')$ between the current solution s and a local optimum s' generated by applying steepest-descent local search to s. In reality, the definition of search depth is more complex, due to randomization in the steepest-descent algorithm. We avoid this issue by computing the mean search depth over a very large number of samples. For each problem instance, we execute $TS_{Taillard}$ for 1,000,000 iterations. We compute search depth at each iteration, and record the resulting

time-series.

In Table 7.1, we present summary statistics for the observed search depth under $TS_{Taillard}$ for various well-known OR Library benchmark instances. In all cases, search remains on average only 1–2 moves away from local optima. For smaller problem instances, this is consistent with the fact that local optima can be escaped under steepest-descent by accepting a short sequence of dis-improving moves. For larger problem instances, we expected search depth to decrease, given the corresponding decrease in attractor basin strength. However, as shown in Table 7.1, search depth remains relatively constant over different problem sizes. The lack of a decrease in mean search depth at larger problem sizes is due to the increase in the tabu tenure under the rules specified in Section 7.3; if the tabu tenure is decreased, the search depth is inversely correlated with problem size. The low standard deviation indicates that $TS_{Taillard}$ *consistently* maintains a low search depth, although the maximal depth indicates that search is occasionally driven far from local optima. It is unclear whether such large distances are actually required to escape certain local optima, or are an artifact of the short-term memory. Additionally, we show the mean search depth for a sample of 1,000,000 *random* semi-active solutions in Table 7.1. Comparing the search depths of random solutions and solutions visited by $TS_{Taillard}$, we again observe that search is, on average, able to remain very deep in the attractor basins of local optima.

In summary, search under $TS_{Taillard}$ is largely restricted to the sub-space of the fitness landscape containing local optima and solutions that are close, in terms of distance, to local optima. Two factors enable this behavior: the strong bias toward local optima that is induced by the core steepest-descent heuristic and relative weakness of attractor basins in the JSP. Together, these factors allow $TS_{Taillard}$ to ignore a substantial proportion of the search space, which may in part account for the superior performance of $TS_{Taillard}$ and other tabu search algorithms on the JSP.

7.5 A Static Cost Model

Let $S_{lopt+} \subseteq S_{\Omega}$ denote the subset of feasible solutions to a problem instance Ω that contains both local optima and solutions that are nearby, in terms of distance, to local optima. Due to the lack of long-term memory, and assuming that the short-term memory acts primarily to escape local optima, we would expect *a priori* the high-level dynamics of $TS_{Taillard}$ to be nothing more than a random walk over the S_{lopt+} sub-space of the fitness landscape. Although undoubtedly controversial to promoters of the tabu search meta-heuristic, the random walk hypothesis is the most reasonable hypothesis, as alternative hypotheses depend on currently non-existent evidence for some emergent high-level search dynamic that is more effective than a random walk. Of course, our null hypothesis is only applicable to $TS_{Taillard}$ and other tabu search algorithms that lack long-term memory mechanisms; we investigate the impact of such mechanisms on tabu search dynamics in Chapter 11.

If search under $TS_{Taillard}$ is really a random walk over S_{lopt+}, then we would expect problem difficulty to be a function of both the size of S_{lopt+} and the number and/or distribution of optimal solutions (i.e., targets) within S_{lopt+}. To test this hypothesis, we analyze the accuracy of static cost models based on the following three measures, each previously described in Section 5.3: the number of optimal solutions ($|optsols|$), the mean distance between random local optima ($\overline{d}_{lopt\text{-}lopt}$), and the mean distance between random local optima and the nearest optimal solution ($\overline{d}_{lopt\text{-}opt}$). The mean distance between random local optima indirectly estimates $|S_{lopt+}|$, while the mean distance between random local optima and the nearest optimal solution simultaneously accounts for the number/distribution of optimal solutions and $|S_{lopt+}|$. Taken together, these allow us to analyze both the independent and aggregate impact of the set of optimal solutions and the size of $|S_{lopt+}|$ on search cost in $TS_{Taillard}$.

As indicated in Section 5.3, most research on cost models of local search, static or otherwise, has proceeded in the context of either MAXSAT or the more general MAXCSP. Indeed, three of the four measures we consider are either drawn from, or inspired by, prior research on MAXSAT. Because our research represents the first attempt to develop static cost models for NP-hard problems other than MAXSAT/MAXCSP, we also take the opportunity to investigate the potential for "universals" in static cost models for local search algorithms. Given the potential for cross-comparison, we also analyze the accuracy of a static cost model based on the backbone size (Section 5.3.5), a factor widely thought to influence local search cost in MAXSAT.

In this subsection, we analyze the ability of static cost models to account for the observed variability in the cost required by $TS_{Taillard}$ to locate optimal solutions to 6×4 and 6×6 random JSPs. In doing so, we make the following contributions:

1. We show that analogs of fitness landscape features known to influence local search cost in MAXSAT, specifically the number of optimal solutions ($|optsols|$) and the mean distance between random local optima and the nearest optimal solution ($\overline{d}_{lopt\text{-}opt}$), also influence the cost of locating optimal solutions under $TS_{Taillard}$. Further, the *strength* of the influence of these two features is nearly identical in both problems. As in MAXSAT, we find that $\overline{d}_{lopt\text{-}opt}$ has a much stronger influence on search cost than $|optsols|$, and ultimately accounts for a significant proportion of the variability in the cost of finding optimal solutions to random JSPs.

2. Our experiments indicate that for random JSPs with moderate to large backbones, the correlation between backbone size and the number of optimal solutions is extremely high. As a direct consequence, for these problems backbone size provides no more information than the number of optimal solutions, and vice versa: one of the two features is necessarily redundant. Given the recent surge of interest in the link between backbone size and problem difficulty, the nearly one-to-one correspondence between these two features was completely unanticipated.

3. In contrast to Singer et al. [SGS00], we find *no* interaction effect between the backbone size and $\overline{d}_{lopt\text{-}opt}$. Further, we find that more complex static cost models

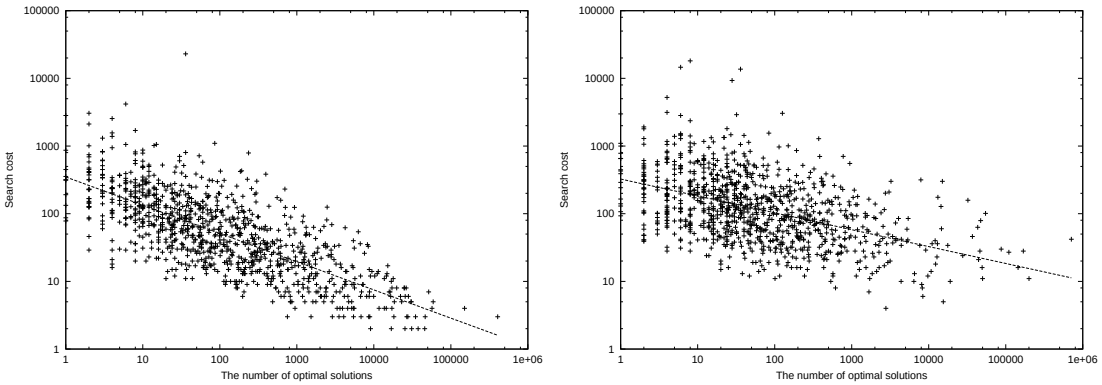

Figure 7.2: Scatter-plots of the number of optimal solutions ($|optsols|$) versus search cost (c_{Q2}) for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

based on multiple search space features, or those that consider interaction effects between search space features, are no more accurate than the simple model based solely on $\overline{d}_{lopt\text{-}opt}$.

Although tabu search algorithms have been successfully applied to a number of NP-hard optimization problems, very little is known in general about which search space features influence problem difficulty, and to what degree. Our research provides a preliminary answer to this question for one particular problem, the JSP, and only for a relatively simple form of tabu search. Consequently, our results may be useful to researchers developing problem difficulty models of tabu search in NP-hard problems other than the JSP, or for models of more advanced tabu search algorithms for the JSP.

7.5.1 The Number of Optimal Solutions

The first static cost model we consider is based only on the number of optimal solutions to a problem instance, which we denote $|optsols|$. As discussed in Section 5.3.1, we would expect the number of optimal solutions to be inversely correlated with search cost; fewer "targets" should decrease the likelihood of a local search algorithm locating one. This observation formed the basis of the first static cost model of local search in both MAXSAT and MAX CSP, introduced by Clark et al. [CFG+96] and later refined by Singer et al. [SGS00]. Clark et al. demonstrated a relatively strong negative log_{10}-log_{10} correlation between the number of optimal solutions and search cost for three local search algorithms, with r-values ranging anywhere from -0.77 to -0.91. However, the model failed to account for the large cost variance observed for problems with small numbers of optimal solutions, where model residuals varied over three or more orders of magnitude.

We show scatter-plots of $|optsols|$ versus c_{Q2} for our 6×4 and 6×6 problem sets in Figure 7.2. The r^2 values of the corresponding $log_{10} - log_{10}$ regression models are

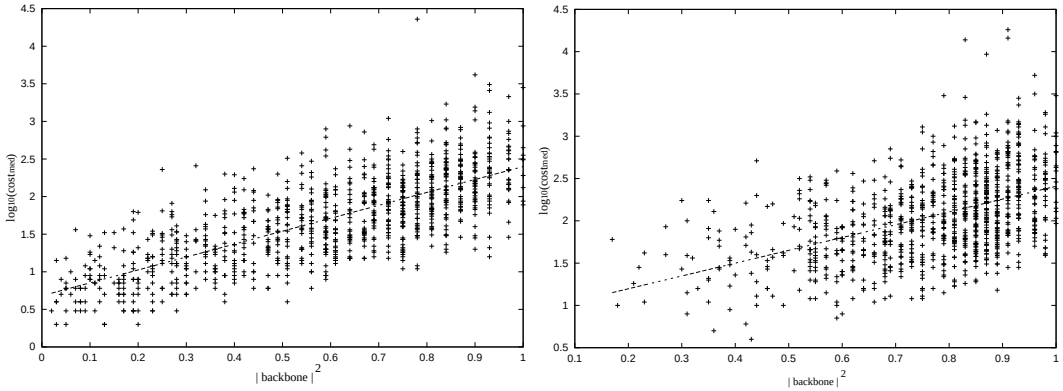

Figure 7.3: Scatter-plots of $|\textit{backbone}|^2$ versus c_{Q2} for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

0.5365 ($r = 0.7325$) and 0.2223 ($r = 0.4715$), respectively. The strength of the correlation is corroborated by the computed non-parametric rank correlation coefficients (-0.7277 and -0.4661, respectively). In comparing the results for the 6×4 and 6×6 problem sets, it is important to note the large difference in the size of the search spaces: 2^{60} versus 2^{90}, respectively. Thus, although the range of $|\textit{optsols}|$ is nearly identical in the two problem sets, the *relative* number of optimal solutions is, on average, much smaller for 6×6 instances. In both cases, search cost varies over 3 orders of magnitude for instances with small (i.e., ≤ 100) numbers of optimal solutions and model accuracy appears to increase slightly as $|\textit{optsols}| \rightarrow \infty$, e.g., observe the decreased variability for the 6×4 problem set for $|\textit{optsols}| \geq$ 1,000. In Watson et al. [WBHW01], we further demonstrate the relationship between model accuracy and $|\textit{optsols}|$ by controlling for the number of optimal solutions in sets of problem instances. The discrepancy between the r^2 values observed for the 6×4 and 6×6 sets can be explained by the fact that the frequency of instances with relatively small numbers of optimal solutions is larger in square random JSPs.

Our results indicate that for typical random JSPs, a static cost model based on $|\textit{optsols}|$ is relatively inaccurate, accounting for roughly 50% of the variance in search cost in the *best* case. As indicated in Section 5.3.1, as $n/m \rightarrow \infty$ the frequency of random JSPs with a large number of optimal solutions, relative to the size of the search space, appears to increase. By extrapolation, we would then expect the accuracy of the $|\textit{optsols}|$ model to increase as $n/m \rightarrow \infty$. In contrast, the accuracy of the model appears worst for the most difficult class of random JSP (i.e., those with $n/m \approx 1.0$), with model residuals varying over 2 to 3 orders of magnitude. Our results also indicate that, both qualitatively and quantitatively, the accuracy of the static cost models based on $|\textit{optsols}|$ is similar in both MAXSAT, MAXCSP, and the JSP.

7.5.2 Backbone Size

Recently, researchers have introduced several problem difficulty models based on the concept of a backbone. Informally, the backbone of a problem instance is the set of solution attributes that have identical values in *all* optimal solutions to the instance. For example, in MAXSAT the backbone is the set of Boolean variables whose value is identical in all satisfying assignments; in the TSP, the backbone consists of the set of edges common to all optimal tours. The recent interest in backbones stems largely from the discovery that backbone size (as measured by the fraction of solution attributes appearing in the backbone) is correlated with search cost in SAT (e.g., see Monasson et al. [MZK$^+$98]). Specifically, Parkes [Par97] showed that large-backboned SAT instances begin to appear in large quantities in the critical region of the phase transition (for a more detailed investigation into the relationship between backbone size and the MAXSAT phase transition, see Singer et al. [SGS00] or Singer [Sin00]). Similarly, Achlioptas et al. [AGKS00] demonstrated a rapid transition from small to large-backboned instances in the phase transition region. While researchers have demonstrated a correlation between backbone size and problem difficulty in SAT, the *degree* to which backbone size accounts for the variability in problem difficulty remains largely unknown.

Only Slaney and Walsh [SW01] have studied the influence of backbone size on search cost in problems other than MAXSAT. Focusing on constructive search algorithms, they analyzed the cost of both finding optimal solutions and proving optimality for a number of NP-complete problems, including the TSP and the number partitioning problem. For these two problems, Slaney and Walsh report a weak-to-moderate correlation between backbone size and the cost of finding an optimal solution (0.138 to 0.388). No studies to date have directly quantified the correlation between backbone size and problem difficulty for local search algorithms.

We now consider a static cost model based on backbone size, which we denote $|\textit{backbone}|$. Our motivation is pragmatic, given the apparent success of the measure to account for problem difficulty in other NP-hard problem; we see no obvious link between the backbone size and either $|S_{\textit{lopt+}}|$ or the number of optimal solutions. In preliminary analysis, we observed a significant non-linear components in the relationship between $|\textit{backbone}|$ and $\textit{log}_{10}(c_{Q2}$. We show scatter-plots of $|\textit{backbone}|^x$ versus c_{Q2} for 6×4 ($x = 3$) and 6×6 ($x = 5$) random JSPs in Figure 7.3. The r^2 values of the corresponding regression models are 0.5307 and 0.2331, respectively. As with the $|\textit{optsols}|$, the model errors are heterogeneous, although the results are consistent with the computed rank correlation coefficients (0.7275 and 0.4701, respectively). In both instances, the r-values (0.7285 and 0.4828, respectively) are significantly larger than that reported by Slaney and Walsh for constructive search algorithms. Further, we found absolutely no evidence that the most difficult instances possess medium-sized backbones, as conjectured by Achlioptas et al. [AGKS00] for SAT.

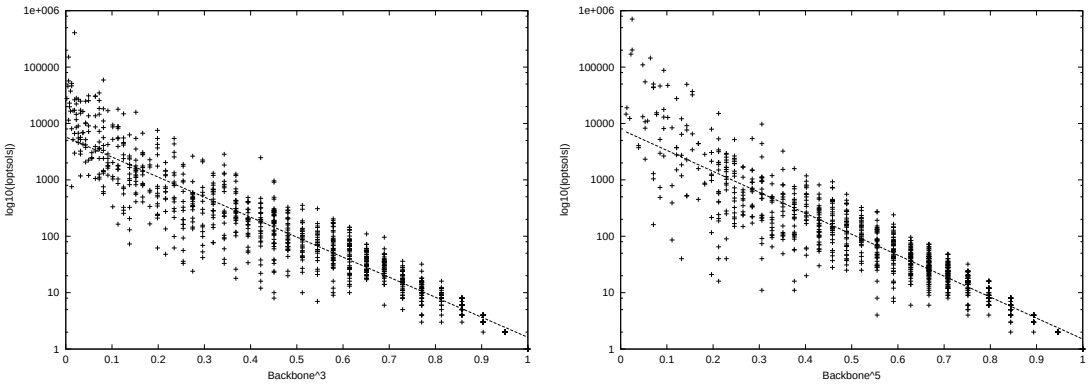

Figure 7.4: Scatter-plots of $|\textit{backbone}|^x$ (where x varies depending on the problem dimensions - see text) versus $|\textit{optsols}|$ for 6×4 and 6×6 random JSPs; the least-squares fit lines are super-imposed.

7.5.3 The Relationship Between Backbone Size and the Number of Optimal Solutions

In contrast to $|\textit{optsols}|$, the backbone size captures some information regarding the distribution of the optimal solutions in the fitness landscape. Consequently, we would expect *a priori* the accuracy of a static cost model based on $|\textit{backbone}|$ to at least be different than that of the $|\textit{optsols}|$ model. However, we observe almost *identical* r^2 values; the absolute difference in r^2 for the two models is respectively 0.0058 and 0.0008 for the 6×4 and 6×6 problem sets. Upon closer examination, the minimal discrepancy is due to the fact that $|\textit{backbone}|$ and $|\textit{optsols}|$ are highly correlated. To illustrate this point, we show scatter-plots of $|\textit{backbone}|^x$ versus $|\textit{optsols}|$ for our 6×4 ($x = 3$) and 6×6 ($x = 5$) problem sets; the r^2 values of the corresponding regression models are 0.8654 and 0.8403, respectively. Within each problem set, the correlation is significantly higher for instances with large backbones, and gradually decays as $|\textit{backbone}| \rightarrow 0.0$.

Our results indicate that for problem instances with large-to-moderate backbones, the backbone size is essentially a proxy for the number of optimal solutions, and vice-versa. From the standpoint of static cost models for reasonably difficult random JSPs (i.e., those with moderate-to-large backbones), the two features are largely redundant. In retrospect, this observation is not surprising given what is implied by a large backbone – as more order variables are fixed, fewer solutions can satisfy the constraints of the backbone. It is unclear to what degree, if any, this correlation is due to randomness in the operation durations, job routing orders, or both. However, it *is* surprising that the link between the number of optimal solutions and backbone size has not previously been explored for MAXSAT and related problems, where most research on the role of backbones in problem difficulty originates. We note that in preliminary experiments, we have observed similar results for MAXSAT in a range of problem instances near the phase transition region of Random 3-SAT, indicating the link between backbone size

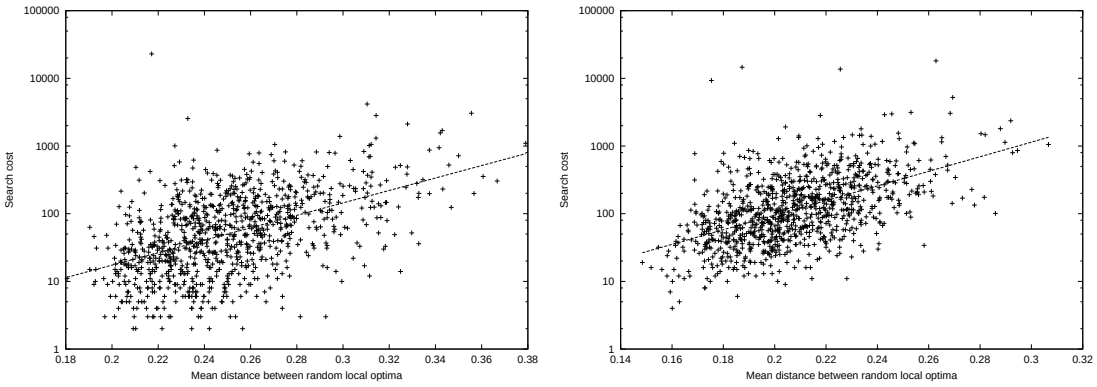

Figure 7.5: Scatter-plots of the mean distance between random local optima ($\overline{d}_{lopt\text{-}lopt}$) versus search cost ($c_{Q2}$) for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

and the number of optimal solutions may in fact be a more general phenomenon.

7.5.4 The Mean Distance Between Random Local Optima

If search under *TS/Taillard* behaves as a random walk in the sub-space S_{lopt+}, we would expect search cost to be correlated with the size of this sub-space. Using local optima as representative members of S_{lopt+}, we can indirectly estimate the size of S_{lopt+} by computing the mean distance between random local optima, a landscape measure introduced in Section 5.3.2. Although this measure, denoted $\overline{d}_{lopt\text{-}lopt}$, ignores the number and distribution of optimal solutions within S_{lopt+}, it allows us to independently assess the impact of $|S_{lopt+}|$ on problem difficulty. The $\overline{d}_{lopt\text{-}lopt}$ was originally introduced by Mattfeld et al. [MBK99] to account for differences in the relative difficulty for random versus workflow JSPs. Although Mattfeld et al. did demonstrate significant mean differences in $\overline{d}_{lopt\text{-}lopt}$ between random and workflow JSPs, they did not analyze the ability of $\overline{d}_{lopt\text{-}lopt}$ to account for the variability in problem difficulty observed *within* a given set of random or workflow JSPs.

We show scatter-plots of $\overline{d}_{lopt\text{-}lopt}$ versus c_{Q2} for 6×4 and 6×6 random JSPs in Figure 7.5. The r^2 values of the corresponding regression models are 0.2415 and 0.2744, respectively. These results confirm the intuition that the size of the S_{lopt+} sub-space is correlated with problem difficulty under *TS/Taillard*, albeit more weakly than either $|$*optsols*$|$ or $|$*backbone*$|$ in 6×4 random JSPs (r^2 values of 0.2415 versus 0.5365 and 0.5307, respectively). In 6×6 JSPs, the correlation is roughly identical to that observed for $|$*optsols*$|$ and $|$*backbone*$|$ (r^2 values of 0.2744 versus 0.2223 and 0.2331, respectively). In contrast to both $|$*optsols*$|$ and $|$*backbone*$|$, the strength of the $\overline{d}_{lopt\text{-}lopt}$ model appears largely insensitive to relatively small changes in problem size.

Our results indicate that either $\overline{d}_{lopt\text{-}lopt}$ is not an accurate measure of the size of the S_{lopt+} sub-space, or the size of S_{lopt+} is only weakly correlated with problem difficulty

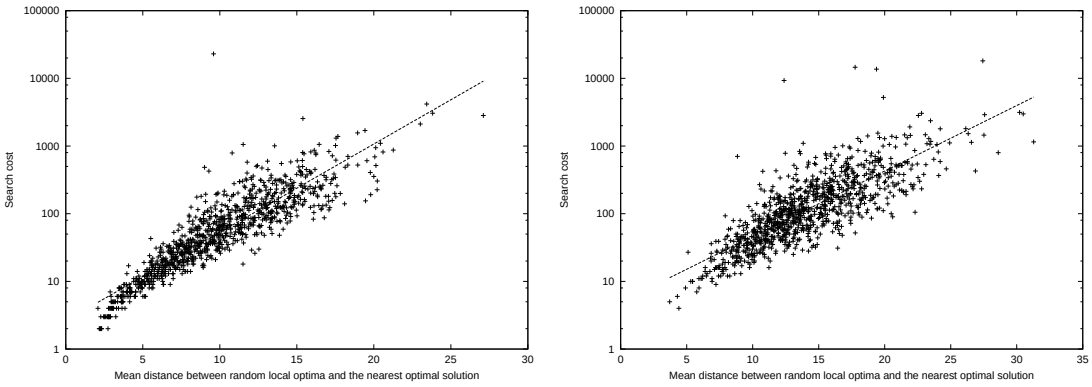

Figure 7.6: Scatter-plots of $\overline{d}_{lopt\text{-}opt}$ versus versus search cost ($log_{10}(c_{Q2})$ for 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

under $TS_{Taillard}$. Further, the $|optsols|$ and $|backbone|$ models are *at least* as accurate as the $\overline{d}_{lopt\text{-}lopt}$ model. Using analogous observations for workflow JSPs, we re-visit and ultimately refute Mattfeld et al.'s original claim regarding the ability of differences in $\overline{d}_{lopt\text{-}lopt}$ to account for the relative difficulty of random and workflow JSPs.

7.5.5 Mean Distance Between Random Local Optima and the Nearest Optimal Solution

Our results indicate that the number of optimal solutions has a stronger influence on problem difficulty than the size of the S_{lopt+} sub-space in 6×4 random JSPs (with respective r^2 values of 0.5365 and 0.2415), while in 6×6 random JSPs, their influence is roughly equal (with respective r^2 values of 0.2223 and 0.2744). In any case, when taken independently, $|optsols|$ and $\overline{d}_{lopt\text{-}lopt}$ only account for a small-to-moderate proportion of the observed variability in problem difficulty. This result was expected – and we performed the prior analyses to assess the impact of each factor independently.

We now simultaneously consider the influence of the number of optimal solutions and the size of the S_{lopt+} sub-space on problem difficulty. In particular, we consider a landscape measure that accounts for both features: the mean distance between random local optima and the nearest optimal solution, denoted $\overline{d}_{lopt\text{-}opt}$, which we introduced in Section 5.3.4. The intuition behind this measure is that search cost is proportional to the distance that must be traversed, i.e., the distance between the initial solution and the nearest optimal solution. Alternatively, we view $\overline{d}_{lopt\text{-}opt}$ as a measure of the *effective* size of S_{lopt+}, accounting for the distribution of optimal solutions within S_{lopt+}. An alternative to introducing a new measure is to develop static cost models based on the two existing measures, via multiple regression methods; we discuss such an approach in Section 7.5.6.

We show scatter-plots of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} for our 6×4 and 6×6 random JSPs in Figure 7.6. The r^2 values of the corresponding regression models are 0.8044 and

0.6424. As we discuss in detail below and in Section 7.7.1, the model errors are heterogeneous; however, the results are consistent with the computed rank correlation coefficients (0.9162 and 0.8072, respectively). In both figures, we observe a slight curvature for instances with small $\overline{d}_{\textit{lopt-opt}}$, the presence of which we cannot currently explain. The curvature can be compensated for by applying a square-root transformation to the independent variable, although this only slightly inflates the resulting r^2 values, respectively to 0.826 and 0.6541.

The $\overline{d}_{\textit{lopt-opt}}$ model is significantly more accurate than any of the $|\textit{optsols}|$, $|\textit{backbone}|$, or $\overline{d}_{\textit{lopt-lopt}}$ models, and roughly accounts for a remarkable 2/3 to 3/4 of the variability in problem difficulty. With few exceptions, the model residuals vary over roughly 1 to 1.5 orders of magnitude in the 6×4 and 6×6 problems, respectively; the improvement is substantial in comparison to the residuals for the models based on either $|\textit{optsols}|$, $|\textit{backbone}|$, or $\overline{d}_{\textit{lopt-lopt}}$. Interestingly, our r^2 values are very similar to those reported by Singer et al. [SGS00] for an analogous cost model in MAXSAT, where the r^2 values ranged between roughly 0.6 and 0.9.

In both problem sets, there is strong evidence that model residuals are heterogeneous, growing larger with increases in $\overline{d}_{\textit{lopt-opt}}$. Consequently, the $\overline{d}_{\textit{lopt-opt}}$ model is on average less accurate for problem instances with large $\overline{d}_{\textit{lopt-opt}}$, or equivalently, with large $c_{\textit{Q2}}$. Singer et al. report a similar phenomenon for the analogous MAXSAT cost model. The discrepancy between the r^2 values of the 6×4 and 6×6 problem sets is due to two factors. First, there are more very high-cost 6×6 instances (i.e., those with $c_{\textit{Q2}} \geq$ 5,000), and large model residuals are typically associated with these instances. Second, although the range of $\overline{d}_{\textit{lopt-opt}}$ is nearly identical in the two problem sets, the number of instances for which $\overline{d}_{\textit{lopt-opt}} \leq 0.1x$, where x is the maximum possible distance, is much larger in 6×4 random JSPs (161 versus 67). We further analyze the relationship between the $\overline{d}_{\textit{lopt-opt}}$ model and very high-cost random JSPs in Section 7.7.1 and consider the influence of the ratio of jobs to machines (n/m) on the accuracy of the $\overline{d}_{\textit{lopt-opt}}$ model in Section 7.6.2.

In conclusion, we have shown that a simple function of both the number/distribution of optimal solutions and the size of the $S_{\textit{lopt+}}$ sub-space accounts for a significant proportion of the variability in problem difficulty under $\textit{TS}_{\textit{Taillard}}$. The high correlation also provides strong evidence to support our hypothesis that search under $\textit{TS}_{\textit{Taillard}}$ behaves largely as a random walk over the effective size of the $S_{\textit{lopt+}}$ sub-space. Finally, we note that the $\overline{d}_{\textit{lopt-opt}}$ model is consistent with the observation that hard (easy) problem instances tend to be hard (easy) for all local search algorithms, as discussed in Section 2.3. Intuitively, if the distance between random local optima and the nearest optimal solution for a particular problem instance is very large, we would expect the instance to be difficult for *any* algorithm based on local search, as search in such algorithms clearly progresses in small increments.

	$log_{10}(\|optsols\|)$	$\|backbone\|$	$\overline{d}_{lopt\text{-}lopt}$	$\overline{d}_{lopt\text{-}opt}$
$log_{10}(\|optsols\|)$	1.0	-0.921	-0.039	-0.751
$\|backbone\|$	-0.921	1.0	0.006	0.722
$\overline{d}_{lopt\text{-}lopt}$	-0.039	0.006	1.0	0.571
$\overline{d}_{lopt\text{-}opt}$	-0.751	0.722	0.571	1.0

Table 7.2: The correlation (Pearson's r) between fitness landscape features for 6×4 random JSPs.

	$log_{10}(\|optsols\|)$	$\|backbone\|$	$\overline{d}_{lopt\text{-}lopt}$	$\overline{d}_{lopt\text{-}opt}$
$log_{10}(\|optsols\|)$	1.0	-0.8967	0.0905	-0.514
$\|backbone\|$	-0.8967	1.0	-0.0797	0.4987
$\overline{d}_{lopt\text{-}lopt}$	0.0905	-0.0797	1.0	0.6507
$\overline{d}_{lopt\text{-}opt}$	-0.514	0.4987	0.6507	1.0

Table 7.3: The correlation (Pearson's r) between fitness landscape features for 6×6 random JSPs.

7.5.6 Models Based on Multiple Landscape Features

Thus far, we have only analyzed static cost models based on individual fitness landscape features. We now consider static models based on multiple features, with two primary goals. First, we test whether we can improve the accuracy of the $\overline{d}_{lopt\text{-}opt}$ model by simultaneously considering $\overline{d}_{lopt\text{-}opt}$ in conjunction with $|optsols|$, $|backbone|$, and $\overline{d}_{lopt\text{-}lopt}$. Second, we investigate whether the $\overline{d}_{lopt\text{-}opt}$ can be approximated via some combination of other features, specifically $|optsols|$ and $\overline{d}_{lopt\text{-}lopt}$.

We proceed via well-known multiple regression methods. Ideally, the independent variables in a multiple regression model are highly correlated with the dependent variable, but not with each other; if the independent variables are highly correlated, they are said to be collinear. Collinearity is known to cause difficulties for multiple regression model selection techniques (which identify the relevant subset of features and/or combination of features), in part because the regression coefficients are not unique, making interpretation very difficult [Ott93]. In Tables 7.2 and 7.3, we show the pair-wise correlation between the various landscape features for 6×4 and 6×6 random JSPs, respectively. In both problem sets, there exists a high degree of collinearity between many of the four features that serve as the independent variables in our multiple regression models, suggesting that we may encounter problems in developing multiple regression models. Further, we note that when the sample size is large (recall $n = 1000$ for these two problem sets), model terms may be statistically significant due to high power (β), but in reality have little effect on model accuracy: i.e., dropping such terms yields a negligible reduction in the model r^2.

To improve the accuracy of the basic $\overline{d}_{lopt\text{-}opt}$ model, we first consider multiple regression models with additive terms, i.e., we do not account for interaction effects among the independent variables. In both the 6×4 and 6×6 problem sets, the multiple regression models resulting from forward selection, backward elimination, and step-wise model selection methods [Ott93] [Coh95] are very different, as expected given collinear independent variables and a large sample size. However, the $\overline{d}_{lopt\text{-}opt}$ term was present in

all of the resulting models, and was consistently the most statistically significant term. For 6×4 and 6×6 random JSPs, the *best* additive multiple regression models we obtained yielded r^2 values of 0.8296 and 0.6589, respectively. Although these models contained terms other than $\overline{d}_{lopt\text{-}opt}$ (due to the artificial significance caused by the large sample size), these terms have minimal impact on overall model accuracy (compare the respective r^2 values of 0.8044 and 0.6424 for the basic $\overline{d}_{lopt\text{-}opt}$ model). Further, when we considered both additive and interaction effects, we found *no* statistically significant interaction terms.

Interestingly, although Singer et al. control for backbone size in their experiments, they do not explicitly indicate whether an interaction effect between the backbone size and the mean distance between random quasi-solutions and the nearest optimal solution ($\overline{d}_{quasi\text{-}opt}$) was observed. However, their results do suggest a lack of interaction effect, in that the regression slopes observed for their $\overline{d}_{quasi\text{-}opt}$ model are largely homogeneous across a wide range of backbone sizes and clause-to-variable ratios (e.g., see Singer et al. (2000), Table 2, p. 249); the intercepts are slightly more variable, which is likely due in part to the presence of high-residual problem instances.

7.5.7 A note on Backbone Robustness

In addition to introducing the $\overline{d}_{quasi\text{-}opt}$ static cost model for MAXSAT (see Section 5.3.4), Singer et al. also posited a causal model to account for the variability in $\overline{d}_{quasi\text{-}opt}$ observed for different problem instances. Their model is based on the notion of *backbone robustness*. A MAXSAT instance is said to have a *robust* backbone if a substantial number of clauses can be deleted before the backbone size is reduced by at least half. Conversely, an instance is said to have a *fragile* backbone if the deletion of just a few clauses reduces the backbone size by half or more. Singer et al. argue that "backbone fragility approximately corresponds to how extensive the quasi-solution area is" ([SGS00], p. 251), by noting that a fragile backbone allows for large $\overline{d}_{quasi\text{-}opt}$ because of the sudden drop in backbone size, while $\overline{d}_{quasi\text{-}opt}$ is necessarily small in problem instances with robust backbones.

As evidence of this hypothesis, Singer et al. measured a moderate (≈ -0.5) negative correlation between backbone robustness and the log of local search cost for large-backboned MAXSAT instances. Surprisingly, this correlation degraded as the backbone size was decreased, leading to the conjecture that "finding the backbone is less of an issue and so backbone fragility, which hinders this, has less of an effect" ([SGS00], p. 254); this conjecture was never explicitly tested. We have previously reported very similar results for random JSPs [WBHW01]. As indicated in Section 7.5.5 and more fully in Section 7.7, we have since discovered relatively serious deficiencies in the $\overline{d}_{lopt\text{-}opt}$ model (and by analogy, likely deficiencies in the $\overline{d}_{quasi\text{-}opt}$), and feel it is somewhat premature to posit causal hypotheses before the source of these deficiencies is completely understood. As a consequence, we have not pursued further analyses of backbone robustness in the JSP.

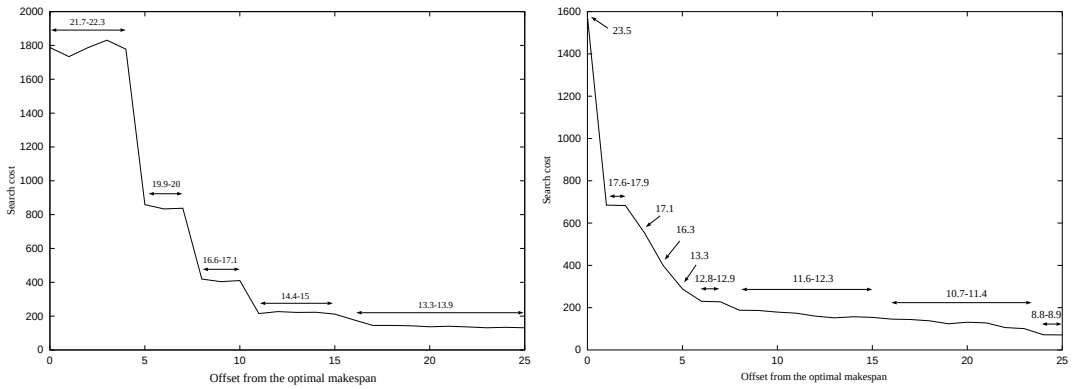

Figure 7.7: The offset x from the optimal makespan C^*_{max}, $0 \leq x \leq 25$, versus the cost $c_{Q2}(x)$ required to locate a solution with $C_{max} \leq C^*_{max} + x$ for two 6×6 random JSPs. The numeric annotations indicate either $d_{lopt-T(x)}$ for a specific x, or the range of $d_{lopt-T(x)}$ over a contiguous sub-interval of x.

7.6 Applications of the $\overline{d}_{lopt\text{-}opt}$ Static Cost Model

We now move from the development of static cost models of $TS_{Taillard}$ to their *application*, i.e., their ability to account for observations regarding problem difficulty and local search for random JSPs. We begin in Section 7.6.1 by demonstrating that a simple extension of the $\overline{d}_{lopt\text{-}opt}$ static cost model accounts for most of the variability in the cost of locating *sub-optimal* solutions to our 6×4 and 6×6 random JSPs. This extension represents the first quantitative model of the cost of locating sub-optimal solutions to any NP-hard optimization problem, and explains the frequently large differences in the cost of locating sub-optimal solutions of similar quality (e.g., see [Stu99]). It is well-known that square JSPs are generally more difficult than rectangular JSPs. In Section 7.6.2, we provide evidence that this phenomenon is likely due to differences in the distribution of $\overline{d}_{lopt\text{-}opt}$ for the two problem types. For square JSPs, the proportion of problem instances with large values of $\overline{d}_{lopt\text{-}opt}$ is substantial, while most rectangular JSPs have very small values of $\overline{d}_{lopt\text{-}opt}$.

7.6.1 Modeling the Cost of Locating Sub-Optimal Solutions

With the exception of some widely used benchmarks, the optimal makespan of any given problem instance is unknown. Local search algorithms are *at best* guaranteed to locate an optimal solution with probability approaching 1 as the run-time approaches ∞, i.e., if the algorithm is asymptotically complete. Given these circumstances, practitioners use a pragmatic termination criterion: allocate as much CPU time as possible for a given run, and return the best solution found.

Although larger run-times generally yield higher-quality solutions, the relationship is typically discontinuous, non-linear, or both. Often, small or moderate increases in run-

time fail, on average, to improve solution quality. For example, Stützle ([Stu99], p. 47) notes that in the Traveling Salesman Problem "...instances appear to have 'hard cliffs' for the local search algorithm, corresponding to deep local minima, which are difficult to pass." Similar observations have been reported for a variety of NP-hard problems, including the JSP. Another manifestation of this phenomenon has been observed by several researchers, including ourselves. Here, multiple independent trials of a particular local search algorithm typically yield equally fit sub-optimal solutions, or small sets of equally fit sub-optimal solutions.

A simple way to visualize this phenomenon is to plot the search cost required to achieve a solution with a fitness of *at least* $C^*_{max} + x$ over a wide range of $x \geq 0$. In Figure 7.7, we provide examples of such plots for two moderately difficult 6×6 random JSPs. In both plots, the offset from the optimal makespan x is varied from 0 to 25. For each offset x, we show the median search cost c_{Q2} required by $TS_{Taillard}$ to locate a solution s with $C_{max}(s) \leq C^*_{max} + x$; we denote this quantity by $c_{Q2}(x)$. In the left side of Figure 7.7, we show a typical example of a problem instance with discrete jumps in search cost at specific sub-optimal makespans, with plateaus in search cost in between the jump points. In the right side of Figure 7.7, we show a problem instance for which the decay in search cost as $x \rightarrow 25$ is more gradual; a large, discontinuous jump in search cost occurs only between $x = 0$ and $x = 1$.

As shown in Section 7.5, the cost required by $TS_{Taillard}$ to locate *optimal* solutions to random JSPs is strongly correlated with $\overline{d}_{lopt\text{-}opt}$. Intuitively, a problem is difficult if $TS_{Taillard}$ is, on average, initiated from solutions that are very distant from the nearest optimal solution. We now consider a generalization of this intuition: problem difficulty is correlated with the distance between the initial solution and the nearest solution in the set of *target* solutions, i.e., optimal *or* sub-optimal solutions. As evidence of this hypothesis, we consider a set $T(x)$ containing all solutions with a makespan between C^*_{max} and $C^*_{max} + x$, $x \geq 0$, and denote the mean distance between random local optima and the nearest solution in the set $T(x)$ by $\overline{d}_{lopt-T(x)}$. As with the computation of $\overline{d}_{lopt\text{-}opt}$, statistics are taken over 5,000 independent samples. We have annotated the plots in Figure 7.7 with the computed $\overline{d}_{lopt-T(x)}$ for $0 \leq x \leq 25$. In both figures, we observe that (1) large jumps in search cost coincide with large jumps in $\overline{d}_{lopt-T(x)}$, (2) intervals of roughly constant search cost correspond to contiguous sub-intervals of x with nearly identical values of $\overline{d}_{lopt-T(x)}$, and (3) gradual drops in search cost coincide with gradual drops in $\overline{d}_{lopt-T(x)}$. Consequently, it seems reasonable to conclude that $\overline{d}_{lopt-T(x)}$ may account for much of the variability in the cost of locating *both* optimal and sub-optimal solutions to random JSPs.

To test this hypothesis, we computed $c_{Q2}(x)$ and $\overline{d}_{lopt-T(x)}$ for our 6×4 and 6×6 random JSPs, varying x from 1 to 25. For random 6×4 and 6×6 JSPs, $TS_{Taillard}$ can easily locate solutions with $C_{max} > C^*_{max} + 25$, such that $c_{Q2}(25) \leq 100$ in all but a few instances. In effect, we are creating 25 derivatives of each problem instance (one for each value of x), resulting in new "sub-optimal" 6×4 and 6×6 problem sets, each with 25,000 instances. For many of the derivative instances, especially those corresponding

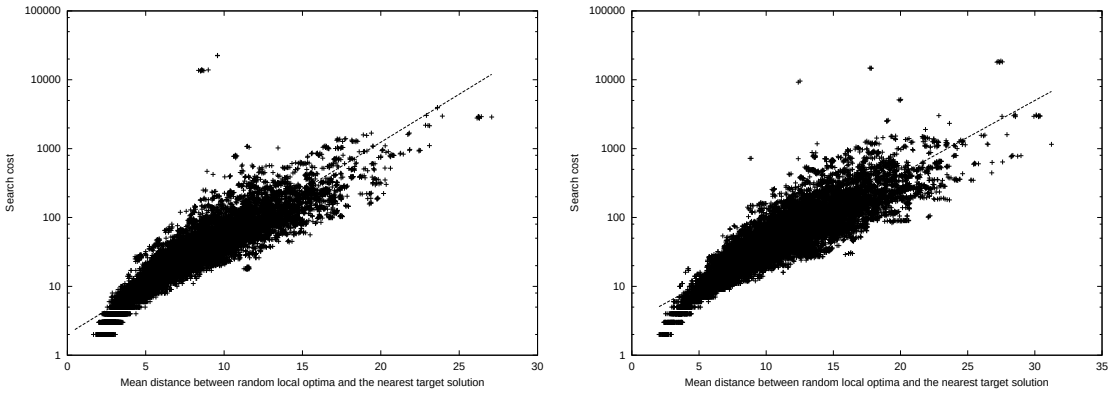

Figure 7.8: Scatter-plots of the mean distance between random local optima and the nearest target solution ($\overline{d}_{\textit{lopt-T(x)}}$) versus search cost ($c_{Q2}(x)$) for sub-optimal 6×4 (left figure) and 6×6 (right figure) random JSPs; the regression lines are super-imposed.

to large x, $c_{Q2}(x) = 0$: equivalently, $\overline{d}_{lopt-T(x)} \approx 0.0$. We observed 1,293 6×4 zero-cost instances, and 60 6×6 zero-cost instances; such instances are excluded from the following analysis.

In Figure 7.8, we show scatter-plots of $\overline{d}_{\textit{lopt-T(x)}}$ versus $c_{Q2}(x)$ for our sub-optimal 6×4 and 6×6 problem sets; the r^2 values for the corresponding regression models are 0.8866 and 0.8252, respectively. Clearly, $\overline{d}_{\textit{lopt-T(x)}}$ accounts for most of variability in the cost required by $TS_{Taillard}$ to locate sub-optimal solutions to typical random JSPs. Accuracy is slightly larger in both the 6×4 and 6×6 suboptimal problem sets, relative to the cost of locating optimal solutions; 0.8594 versus 0.8073 for the 6×4 problems and 0.8252 versus 0.6424 for the 6×6 problems. We explain the increase in accuracy by noting that the proportion of instances with small values of $\overline{d}_{\textit{lopt-opt}}$ is larger in the sub-optimal problem groups, corresponding to the types of problem instance for which the $\overline{d}_{\textit{lopt-opt}}$ cost model is most accurate.

The $\overline{d}_{\textit{lopt-opt}}$ cost model provides the first quantitative explanation for the "cliffs" in local search cost observed at particular sub-optimal fitness': abrupt changes in local search cost occur where there are abrupt changes in $\overline{d}_{\textit{lopt-opt}}$. Similarly, the plateaus observed in Figure 7.7 occur because solutions on the plateau are equi-distant from random local optima; $TS_{Taillard}$ is equally likely to encounter any of the solutions on the plateau, given a fixed run-time. Similarly, gradual increases in search cost occur when slightly better solutions are only marginally farther from random local optima.

7.6.2 Accounting for the Relative Difficulty of Square Versus Rectangular JSPs

Given the observed accuracy of the $\overline{d}_{\textit{lopt-opt}}$ static cost model, it is natural to consider whether differences in the distribution of $\overline{d}_{\textit{lopt-opt}}$ for problems with different ratios of n/m might account for the empirical observation that square JSPs are generally more

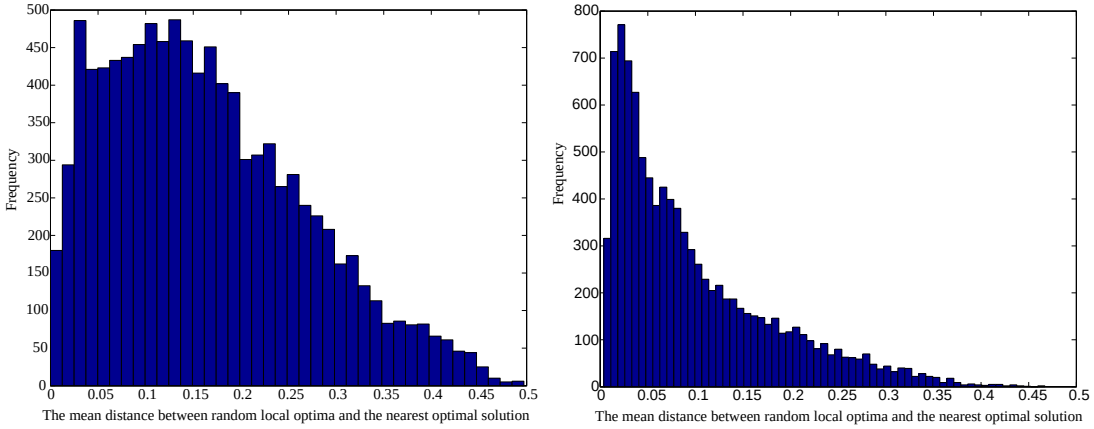

Figure 7.9: Histograms of $\overline{d}_{\textit{lopt-opt}}$ for 10,000 4×3 (left figure) and 7×3 (right figure) random JSPs.

difficult than rectangular JSPs. Unfortunately, we cannot *directly* test this hypothesis; as we discussed in Section 5.3.4, the astronomical number of optimal solutions to square and rectangular JSPs larger than 6×4 and 6×6, respectively, makes computation of $\overline{d}_{\textit{lopt-opt}}$ intractable for these instances. However, we can obtain indirect evidence for this hypothesis by analyzing changes in the distribution of $\overline{d}_{\textit{lopt-opt}}$ for smaller problems.

Fixing $m = 3$, we generated 10,000 random JSPs for each value of n between 3 and 7. Although small by any standard (the JSP is NP-hard only for $m \geq 2$ and $n \geq 3$), these are the largest problem instances for which we can currently compute the distribution of $\overline{d}_{\textit{lopt-opt}}$ for $m/n \approx 2$, e.g., many 8×4 instances possess in excess of one billion optimal solutions. We show histograms of $\overline{d}_{\textit{lopt-opt}}$ for 4×3 and 7×3 random JSPs in Figure 7.9. For the 4×3 instances, the right-tail mass of the distribution is substantial, e.g., for $\overline{d}_{\textit{lopt-opt}} \geq 0.3$). In contrast, we observe minimal right-tail density in the 7×3 problem set, such that instances with $\overline{d}_{\textit{lopt-opt}} \geq 0.3$ are relatively rare. We have also analyzed the distribution of random JSPs with $n/3 < 1$, and observe a continued shift of the distribution mass toward 0.5.

Our results provide relatively strong evidence that the right-tail mass of the $\overline{d}_{\textit{lopt-opt}}$ distribution vanishes as $n/m \rightarrow \infty$, suggesting a cause for the observation that square JSPs are generally more difficult than rectangular JSPs. Further, we conjecture that the shift from exponential to polynomial growth in search cost at $n/m \approx 6$ [Tai94] is due to the disappearance of any significant mass in the right tail of the $\overline{d}_{\textit{lopt-opt}}$ distribution. However, we are currently unable to test this hypothesis. Finally, we expect the accuracy of the $\overline{d}_{\textit{lopt-opt}}$ static cost model to further improve as $n/m \rightarrow \infty$, due to the increasing frequency of instances with small values of $\overline{d}_{\textit{lopt-opt}}$. Consequently, from the standpoint of static cost models, random JSPs with $n/m \approx 1.0$ warrant the most attention in the future.

In a previous paper [WBHW01], we argued that a shift in the distribution of $|\textit{backbone}|$, and not $\overline{d}_{\textit{lopt-opt}}$, was responsible for differences in the relative difficulty of

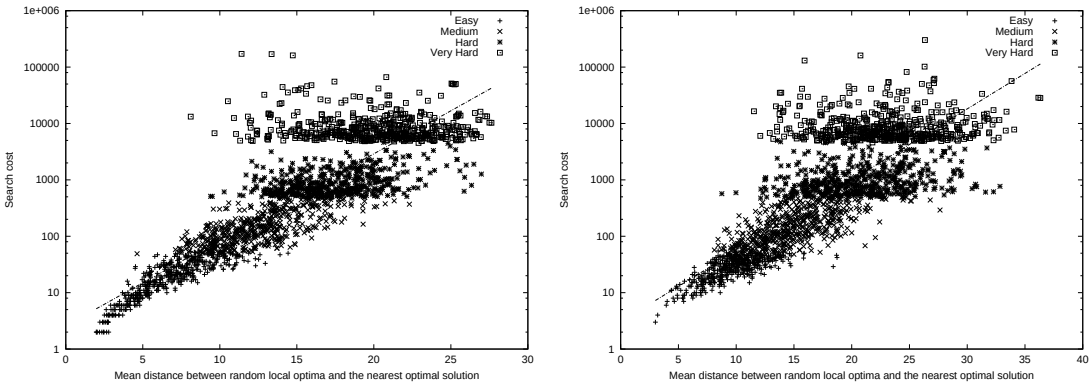

Figure 7.10: Scatter-plots of $\overline{d}_{\textit{lopt-opt}}$ versus search cost (c_{Q2}) for easy ($c_{Q2} \in [1, 49]$), medium ($c_{Q2} \in [50, 499]$), hard ($c_{Q2} \in [500, 4999]$), and very hard ($c_{Q2} \in [5000, \infty]$) 6×4 (left figure) and 6×6 (right figure) random JSPs; the least-squares fit lines are super-imposed.

square versus rectangular JSPs. While our original observation still holds (i.e., the proportion of instances with small backbones grows as $n/m \rightarrow \infty$), we can re-cast our original results in terms of the more accurate static cost model based on $\overline{d}_{\textit{lopt-opt}}$.

7.7 Limitations of the $\overline{d}_{\textit{lopt-opt}}$ Static Cost Model

One deficiency of the $\overline{d}_{\textit{lopt-opt}}$ static cost model, as indicated in Section 7.5.5, is the fact that accuracy appears to be inversely proportional to $\overline{d}_{\textit{lopt-opt}}$, and fails to account for roughly a third of the cost variance in 6×6 random JSPs. In this section, we expose two additional limitations of the $\overline{d}_{\textit{lopt-opt}}$ model. First, in Section 7.7.1, we conclusively demonstrate that accuracy is exceptionally poor for the most difficult, albeit rare, random 6×4 and 6×6 JSPs. Second, we analyze the scalability of the $\overline{d}_{\textit{lopt-opt}}$ to a small set of 10×10 random JSPs. As we show in Section 7.7.2, the accuracy of the $\overline{d}_{\textit{lopt-opt}}$ model is significantly poorer on these larger instances, indicating the failure of the $\overline{d}_{\textit{lopt-opt}}$ model to scale to realistically sized problem instances.

7.7.1 Modeling search cost in exceptionally hard random JSPs

In Section 7.5.5, we provided evidence that the $\overline{d}_{\textit{lopt-opt}}$ model is less accurate for problem instances with large values of $\overline{d}_{\textit{lopt-opt}}$, or equivalently, large c_{Q2}. Of particular concern are the difficult ($c_{Q2} \geq$ 10,000) albeit rare instances appearing in both sides of Figure 7.6; in all but one case, these instances possess the largest residuals under the corresponding regression model. To determine whether large model residuals are typically associated with high-cost random JSPs, we created sets of 6×4 and 6×6 random JSPs with equal proportions of problem instances over the range of c_{Q2}. Specifically, we sub-divided

the possible range of c_{Q2} into the following four contiguous intervals: $[1, 49]$, $[50, 499]$, $[500, 4999]$, and $[5000, \infty]$; these intervals qualitatively correspond to easy, medium, hard, and very hard problem instances, respectively. We then used a simple generate-and-test procedure to construct 500 instances of each sub-class.

In Figure 7.10, we show scatter-plots of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} for 6×4 and 6×6 random JSPs. The r^2 values for the corresponding regression model are 0.7599 and 0.6624, respectively. Because the hard and very hard instances reside in the right-tail of the c_{Q2} distribution, the large relative frequencies of problem instances with values of c_{Q2} near the lower bounds of the corresponding intervals was expected. Although the r^2 values are similar to those obtained for our unfiltered problem sets (0.8044 and 0.6424 for the 6×4 and 6×6 problem sets, respectively), the worst-case residuals are significantly larger, and there exist more instances with large residuals. In both problem sets, we observe a substantial reduction in the accuracy of the $\overline{d}_{lopt\text{-}opt}$ model for hard instances, and an extreme degradation for very hard instances; in the latter case, $\overline{d}_{lopt\text{-}opt}$ is essentially uncorrelated with c_{Q2}. These results more clearly demonstrate the fact that the accuracy of the $\overline{d}_{lopt\text{-}opt}$ static cost model is inversely proportional to both $\overline{d}_{lopt\text{-}opt}$ and c_{Q2}. As a direct consequence, although we are now able to account for much of the variability in search cost for typical random JSPs, an understanding of the fitness landscape features that make certain problems exceptionally difficult for for $TS_{Taillard}$ remains elusive.

Several researchers have reported situations in which problems that are exceptionally difficult for one algorithm are much easier for other algorithms [SG95] [GW94]. To date, this phenomenon has only been observed in constructive search algorithms, and occurs when one algorithm makes a particular sequence of decisions that yields a very difficult sub-problem [SG95]. Although not yet observed in the context of local search, this phenomenon raises an obvious question: "Is the exceptional difficulty of our very hard random JSPs algorithm-independent?". To informally answer this question, we solved both our very hard and typical (i.e., unfiltered) 6×6 instances using two local search algorithms other than $TS_{Taillard}$, and a constructive heuristic search algorithm. Specifically, we considered the following local search algorithms: (1) Nowicki and Smutnicki's state-of-the-art tabu search algorithm [NS96] and (2) van Laarhoven et al.'s simulated annealing algorithm [vLAL92]. We selected Nowicki and Smutnicki's algorithm because it uses a more powerful move operator than $TS_{Taillard}$, and employs an re-intensification mechanism (see Chapter 11); van Laarhoven et al.'s algorithm provides a well-known alternative local search paradigm to tabu search. The constructive algorithm we consider is Beck and Fox's constraint-directed scheduling algorithm [BF00], which was selected because it shares little in common with local search algorithms for the JSP. In all three cases, the search cost (as measured by the median search cost over 1,000 independent trials of the two local search algorithms, and the number of nodes visited by the constructive algorithm) was *generally* larger in the very high-cost instances. However, we did find some exceptional instances that were easily solved by the other algorithms. Upon closer examination, we found that these instances are extremely sen-

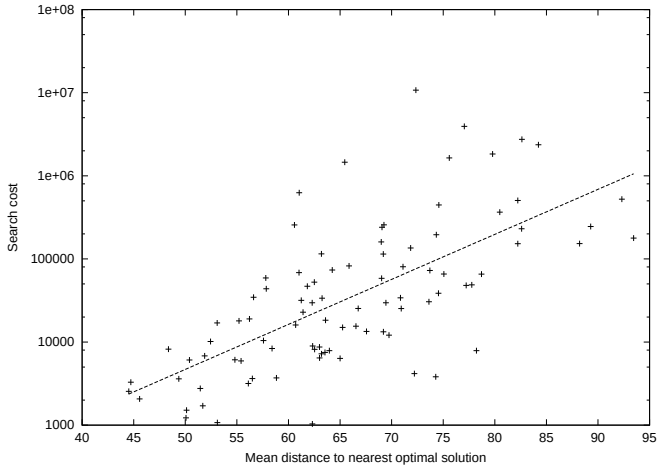

Figure 7.11: Scatter-plot of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} for random 10×10 JSPs; the least-squares fit line is super-imposed.

sitive to the length of the tabu list of $TS_{Taillard}$. We conclude that, with a few exceptions, the difficulty of our very high-cost random JSPs is algorithm-independent.

Finally, we conjecture that the failure of the $\overline{d}_{lopt\text{-}opt}$ model to account for local search cost in very difficult problem instances is likely to extend to MAXSAT. Although Singer et al. do not provide scatter-plots of $\overline{d}_{quasi\text{-}opt}$ versus c_{Q2} for high-cost problem instances (i.e., those with large backbones), their analysis does indicate that the accuracy of their $\overline{d}_{quasi\text{-}opt}$ model is inversely proportional to backbone size (e.g., see Singer et al. (2000), Table 2, p. 249), and as a consequence, to c_{Q2} (as in the random JSP, local search cost and backbone size are positively correlated in MAXSAT). Further, very high-cost SAT instances possess the largest residuals under Singer et al.'s model of backbone robustness (e.g., see Singer et al. (2000), Figure 11, p. 255), which in turn is correlated with $\overline{d}_{quasi\text{-}opt}$.

7.7.2 Assessing Scalability of the $\overline{d}_{lopt\text{-}opt}$ Model

A key unexplored research area in the development of cost models, static or otherwise, is scalability. Reasonably accurate models are functions of all optimal solutions to a problem instance, which in many cases grows exponentially with problem size. Larger problems are typically more difficult, requiring multiple high-cost trials to obtain estimates of either $c_{Q2}/\overline{c}$ or the full run-length distribution. To date, these factors have conspired to prevent researchers from assess model accuracy on larger, more realistically sized, problem instances. Consequently, the question of whether we can expect the accuracy of cost models to scale is current open.

When we initiated our research, available computing power limited our analysis of static cost models to 6×4 and 6×6 instances. Using more recent hardware, we are now able to assess model accuracy on a set of larger 10×10 random JSPs. The number of optimal solutions to these instances is frequently tractable, i.e., less than a few hundred

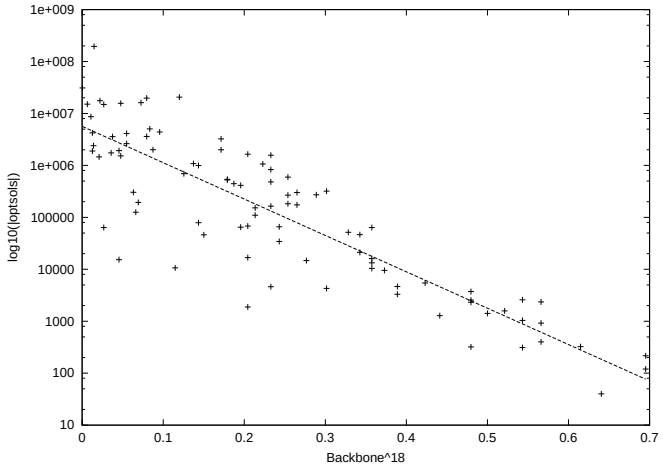

Figure 7.12: Scatter-plot of $|backbone|^{15}$ versus $|optsols|$ for 10×10 random JSPs; the least-square fit line is super-imposed.

	$log_{10}(\|optsols\|)$	$\|backbone\|^{15}$	$\overline{d}_{lopt\text{-}lopt}$	$\overline{d}_{lopt\text{-}opt}$
$log_{10}(\|optsols\|)$	1.0	-0.8672	-0.1327	-0.5314
$\|backbone\|^{15}$	-0.8672	1.0	0.1398	0.5596
$\overline{d}_{lopt\text{-}lopt}$	-0.1327	0.1398	1.0	0.6543
$\overline{d}_{lopt\text{-}opt}$	-0.5314	0.5596	0.6543	1.0

Table 7.4: The correlation (Pearson's) between search space features for 10×10 random JSPs.

million, in contrast to larger rectangular instances, which are still inaccessible.

We computed $\overline{d}_{lopt\text{-}opt}$ for those 92 instances of our 10×10 problem set with ≤ 50 million optimal solutions; the computation is intractable for the remaining 8 instances. Estimates are computed using 5,000 random local optima. We show a scatter-plot of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} for these problem instances in Figure 7.11. The r^2 value for the corresponding regression model is 0.4598, which represents a 33% decrease in model accuracy relative to the 6×6 problem set. Clearly, some additional factor is influencing problem difficulty in larger JSPs.

We observe less drastic reductions in accuracy for the $|optsols|$ ($r^2 = 0.184$ versus 0.2231 for 6×6 JSPs) and $\overline{d}_{lopt\text{-}lopt}$ ($r^2 = 0.2171$ versus 0.2744 for 6×6 JSPs) static cost models. However, in either case, the $\overline{d}_{lopt\text{-}opt}$ model is still significantly more accurate. The strong correlation between the backbone size and the number of optimal solutions persists, as shown in Figure 7.12. However, the relationship is approximately linear in $|backbone|^{15}$ with increasing variance as $|backbone| \rightarrow 0$ (the value '15' minimized residual non-linearities in the corresponding regression model), as opposed to $|backbone|^2$ for the smaller problem sets. Clearly, the relationship between backbone size and the number of optimal solutions is not quadratic, but appears to be an exponential function of problem size. Finally, in Table 7.4 we report the correlation between each of the landscape features analyzed in Section 7.5. Contrasting the results with those in Tables 7.2

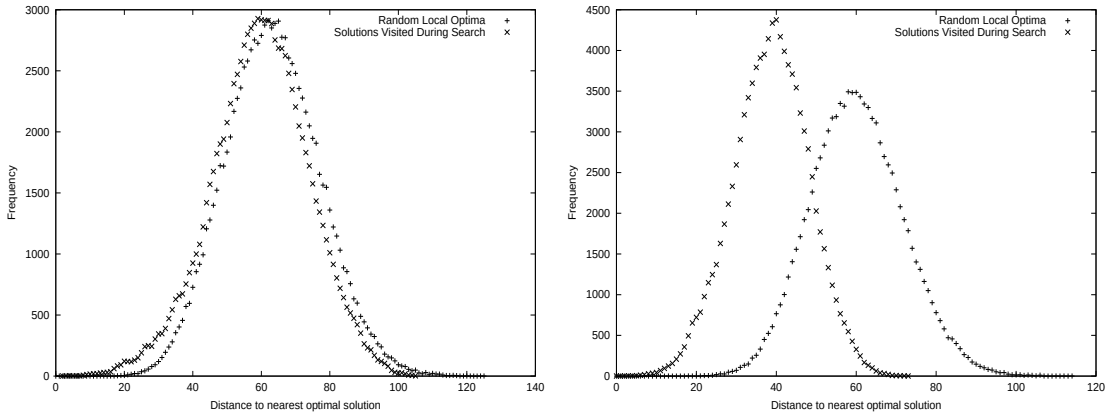

Figure 7.13: Histograms of the distance to the nearest optimal solution (d_{opt}) for (a) 100,000 random local optima and (b) 100,000 solutions visited by *TS-Taillard* for two 10×10 random JSPs.

and 7.3, we find no fundamental changes in the relationships between the features as problem size is increased. Similarly, we found that models based on multiple and/or interacting features failed to improve the accuracy of the basic $\overline{d}_{lopt\text{-}opt}$ model.

7.8 Accounting for Search Bias: A Quasi-Dynamic Cost Model

The deficiencies of the $\overline{d}_{lopt\text{-}opt}$ static cost model raise the possibility that either (1) $\overline{d}_{lopt\text{-}opt}$ is not an entirely accurate indicator of the size of the S_{lopt+} sub-space or (2) the size of the S_{lopt+} sub-space is not completely indicative of search cost, disproving our original hypothesis that *TS-Taillard* behaves as a random walk over S_{lopt+}. We now focus on the first alternative.

In Chapter 6, we demonstrated that random solutions were not representative, in terms of the distribution of their distance to the nearest optimal solution, of the solutions actually visited by a random walk. This result raises the question of whether random local optima are representative of the solutions visited by *TS-Taillard* during search, i.e., in the S_{lopt+} sub-space, and whether we can also improve the accuracy of the $\overline{d}_{lopt\text{-}opt}$ by considering instead the set of "on-line" solutions.

Consider the distribution of the distance to the nearest optimal solution, which we denote d_{opt}, for both random local optima and solutions visited by *TS-Taillard* during a lengthy search. Both distributions are typically Gaussian-like, although we have observed skewed distributions both with and without heavy tails. We provide examples of these distributions for two 10×10 random JSPs in Figure 7.13. Although the two distributions are often identical, they can possess very different means and variances, as occurs for the instance shown in the right side of Figure 7.13. Such differences are rela-

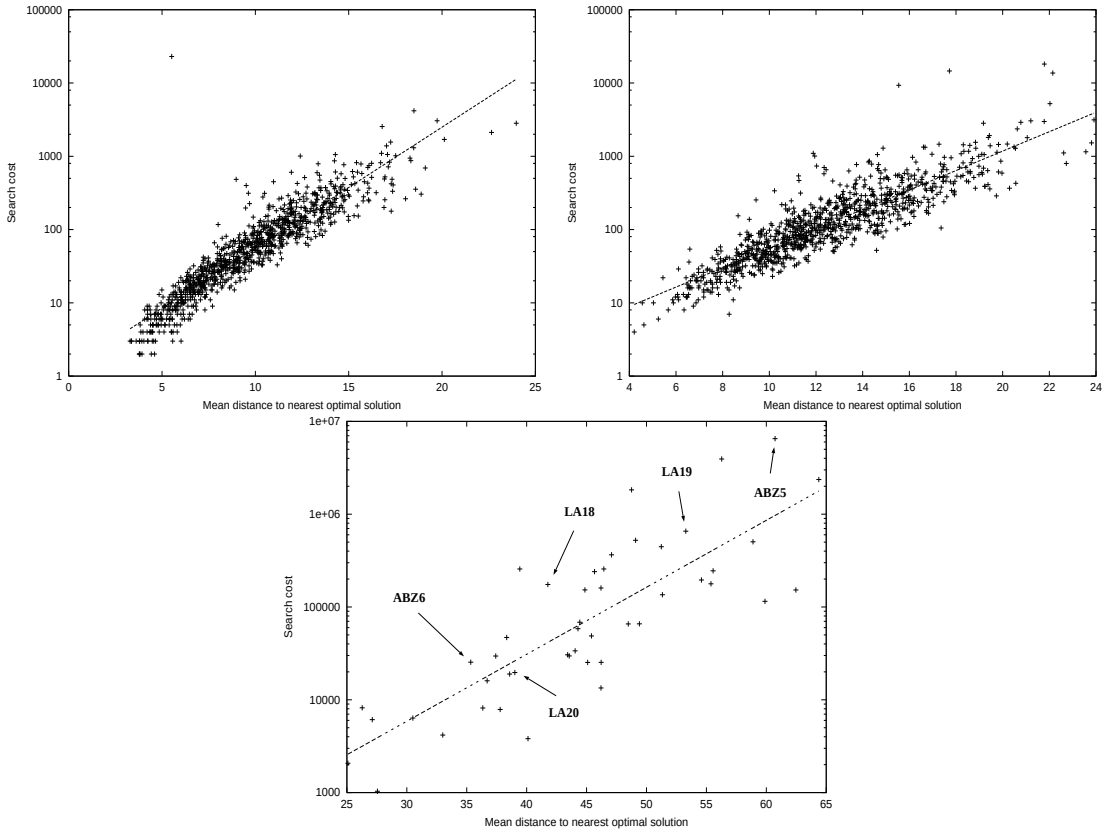

Figure 7.14: Scatter-plots of $\overline{d}_{tabu\text{-}opt}$ versus search cost (c_{Q2}) for 6×4 (upper left figure), 6×6 (upper right figure), and 10×10 (lower figure) random JSPs; the least-squares fit lines are super-imposed.

tively rare in our 6×4 problem set, but occur with increasing frequency in our 6×6 and 10×10 problem sets – the same problems for which the $\overline{d}_{lopt\text{-}opt}$ model is least accurate.

Mirroring the case for *RW* in Section 6.3, these observations led us to conjecture that the mean d_{opt} for solutions visited during search, which we denote $\overline{d}_{tabu\text{-}opt}$, may be a more accurate indicator of the size of S_{lopt+} than $\overline{d}_{lopt\text{-}opt}$. For a given problem instance, we estimate $\overline{d}_{tabu\text{-}opt}$ using a set of 100,000 solutions visited by $TS_{Taillard}$ over a variable number of independent trials. Each trial is initiated from a random local optimum and terminated once an optimal solution is located; we impose the termination criterion (as opposed to continuing search from an optimal solution) because there exist optimal solutions from which no moves are possible under the $N1$ operator [NS96]. We terminate the entire process, including the current trial, once we obtain 100,000 samples.

We show scatter-plots of $\overline{d}_{tabu\text{-}opt}$ versus c_{Q2} for our 6×4 and 6×6 JSPs in the upper portion of Figure 7.14. The respective r^2 values for the corresponding regression models are 0.8441 and 0.7808, representing 5% and 21% increases in accuracy over the $\overline{d}_{lopt\text{-}opt}$ model. In both cases, the absolute accuracy is remarkably high. The variable

impact on the accuracy is due to the frequency of instances in which the distributions of d_{opt} for random local optima and solutions visited during search are dissimilar. Model residuals are typically no larger than $1/2$ an order of magnitude, and we observe fewer and less extreme high-residual instances than under the $\overline{d}_{lopt\text{-}opt}$ model. There is a slight correlation between the magnitude of model errors and $\overline{d}_{tabu\text{-}opt}$. Finally, although not shown, the $\overline{d}_{tabu\text{-}opt}$ model provides similar improvements in the ability to predict the cost of locating sub-optimal solutions to these same problem instances.

We next assess the accuracy of the $\overline{d}_{tabu\text{-}opt}$ model on those 42 of our 10×10 instances with $\leq 100,000$ optimal solutions; the computation of $\overline{d}_{tabu\text{-}opt}$ is intractable for the remaining instances. Given the relatively poor correlation between the number of optimal solutions and search cost, our selection criterion does *not* lead to a clean distinction between "easy" and "hard" problem instances; the hardest instance has approximately 1.5 million optimal solutions. However, on average, instances with $\leq 100,000$ optimal solutions are generally more difficult, with a median c_{Q2} of $65,710$, versus $13,291$ for instances with more than $100,000$ optimal solutions.

A regression model of $\overline{d}_{tabu\text{-}opt}$ versus $log_{10}(c_{Q2})$ for these instances yielded an r_2 value of 0.6641; we show the corresponding scatter-plot in the lower portion of Figure 7.14. The resulting r^2 represents an approximately 41% increase in accuracy over the $\overline{d}_{lopt\text{-}opt}$ model. However, the model residuals typically vary from between $1/2$ and 1 order of magnitude, leaving a moderate proportion of the variability in search cost unexplained. As with the smaller problem sets, the model errors are proportional to $\overline{d}_{tabu\text{-}opt}$, although to a less significant degree than under the $\overline{d}_{lopt\text{-}opt}$ model.

The difference in model r^2 between the 6×6 and 10×10 problem sets is only ≈ 0.14. We have also annotated the scatter-plot with data for five of the seven 10×10 random JSPs found in the OR Library; both `la16` and `la17` have approximately 6.8 and 11.8 million optimal solutions, respectively, making computation of $\overline{d}_{tabu\text{-}opt}$ intractable. The `abz5` and `la19` instances are known to be significantly more difficult than their respective counterparts (i.e., `abz6`, `la18`, and `la20`) for numerous local search algorithms, which is consistent given the observed differences in $\overline{d}_{tabu\text{-}opt}$ [JM99].

Our results clearly demonstrate that the mean distance between solutions visited *during* search and the nearest optimal solution ($\overline{d}_{tabu\text{-}opt}$) is largely responsible for the cost required by Taillard's algorithm to locate optimal solutions to random JSPs. However, two key issues remain. First, as shown by the difference in r^2 obtained for the 6×6 and 10×10 instances, there is still some evidence that the $\overline{d}_{tabu\text{-}opt}$ model may fail to scale to even larger problem instances. Second, as was the case with $\overline{d}_{lopt\text{-}opt}$, it is unclear why, in terms of search dynamics, $\overline{d}_{tabu\text{-}opt}$ is so highly correlated with search cost. To address these issues, we now examine the dynamic behavior of $TS_{Taillard}$ in more detail.

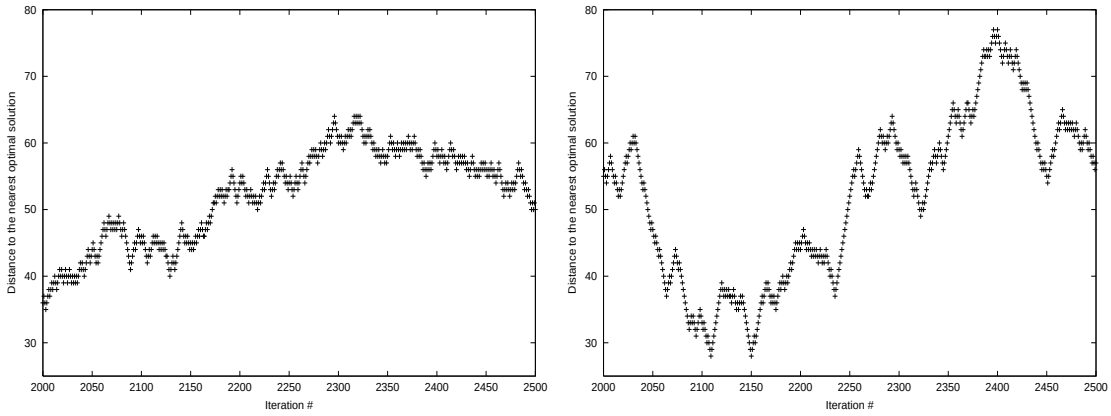

Figure 7.15: Time-series of the distance to the nearest optimal solution for the solutions visited by a random walk (left figure) and $TS_{Taillard}$ (right figure) for a 10×10 random JSP.

7.9 A Dynamic Cost Model

In addition to the indicated technical deficiencies, both the $\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{tabu\text{-}opt}$ cost models have a fundamental conceptual short-coming: neither model provides any *direct* insight into the dynamic run-time behavior of the global search process. In other words, although these models provide strong evidence that $TS_{Taillard}$ is effectively a biased random walk over the S_{lopt+} sub-space, we have yet to propose any specific details, e.g., the set of states in the random walk, the transition probabilities between states, and how the bias differs from that of the baseline random walk algorithm *RW*. We now address the issues by explicitly modeling the behavior $TS_{Taillard}$ as a biased random walk.

The dynamic behavior of *memoryless* local search algorithms, e.g., iterated local search or simulated annealing, can, at least in principle, be modeled using Markov chains: the set of feasible solutions is known, the transition probabilities between neighboring solutions can be computed, and the Markov property is preserved. Although exact, such models require an exponential number of states for NP-hard problems and therefore provide little insight into the qualitative nature of the search process. The challenge is to develop condensed models, in which large numbers of solutions are grouped into individual states, yielding more tractable and consequently understandable Markov chains. We developed such a model for the *RW* algorithm in Chapter 6. A similar approach is also possible when modeling the dynamic behavior of $TS_{Taillard}$ and other tabu search algorithms, although to preserve the Markov property the contents of short-term memory must additionally be embedded into the state definition.

As in Chapter 6, we aggregate solutions based on their distance to the nearest objective, i.e., optimal solution. To model the impact of short-term memory on the search process, we analyze how search progresses in terms of trends either toward or away from the nearest optimal solution. In Figure 7.15, we show a time-series of the distance to the nearest optimal solution for *RW* and $TS_{Taillard}$ for a typical 10×10 random JSP;

similar results are obtained in a limited sampling of our 6×4, 6×6, and 10×10 instances. As expected, the random walk exhibits minimal short-term trending behavior. Due to the limited time horizon displayed, the bias toward solutions that are an average distance from the nearest optimal solution is not evident. In contrast, $TS_{Taillard}$ exhibits distinct short-term trending behavior. We define the (instantaneous) search *gradient* as the difference in the distance to the nearest optimal solution for the current solution and that encountered in the previous iteration. Clearly, $TS_{Taillard}$ is able to maintain a consistent search gradient for extended periods of time, leading to the following hypothesis: $TS_{Taillard}$'s short-term memory mechanism, in conjunction with the core steepest descent heuristic, influences the search process by consistently biasing search either toward or away from the nearest optimal solution.

We define a state $S_{i,x}$ in our Markov model of $TS_{Taillard}$ as a pair representing (1) the set of solutions distance i from the nearest optimal solution and (2) the current search gradient x. Let $d_{opt}(s)$ denote the disjunctive graph distance between between s and the nearest optimal solution. We define $x = grad(s, s_{-1})$ as the difference $d_{opt}(s) - d_{opt}(s_{-1})$ between the current solution s and the solution s_{-1} from the previous iteration. Although $grad(s, s_{-1}) \in [-1, 0, 1]$, for clarity we denote these numeric values symbolically by *closer*, *equal*, and *farther*, respectively. In effect, we are modeling the impact of short-term memory as a simple scalar and embedding this scalar into the state definition. Given a maximum possible distance of D from an arbitrary solution to the nearest optimal solution, our Markov model consists of exactly $3 \cdot (D+1)$ states (the extra state represents the set of optimal solutions).

Next, let the conditional probability $P(S_{i,x'}|S_{j,x})$ represent the probability of *simultaneously* altering the search gradient from x to x' and moving from a solution at distance j from the nearest optimal solution to a solution at distance i away from the nearest optimal solution. The majority of these probabilities obviously equal 0, specifically for any pair of states $S_{i,x'}$ and $S_{j,x}$ with $|i - j| > 1$ or when simultaneous changes in both gradient and distance to the nearest optimal solution are logically impossible, such as from state $S_{i,closer}$ to state $S_{i+1,closer}$. For each i such that $1 \leq i \leq D$, there are exactly 9 possible non-zero transition probabilities:

- $P(S_{i-1,closer}|S_{i,closer})$, $P(S_{i,equal}|S_{i,closer})$, and $P(S_{i+1,farther}|S_{i,closer})$
- $P(S_{i-1,closer}|S_{i,equal})$, $P(S_{i,equal}|S_{i,equal})$, and $P(S_{i+1,farther}|S_{i,equal})$
- $P(S_{i-1,closer}|S_{i,farther})$, $P(S_{i,equal}|S_{i,farther})$, and $P(S_{i+1,farther}|S_{i,farther})$

The set of transition probabilities is also subject to the total-probability constraints:

- $P(S_{i-1,closer}|S_{i,closer}) + P(S_{i,equal}|S_{i,closer}) + P(S_{i+1,farther}|S_{i,closer}) = 1.0$
- $P(S_{i-1,closer}|S_{i,equal}) + P(S_{i,equal}|S_{i,equal}) + P(S_{i+1,farther}|S_{i,equal}) = 1.0$
- $P(S_{i-1,closer}|S_{i,farther}) + P(S_{i,equal}|S_{i,farther}) + P(S_{i+1,farther}|S_{i,farther}) = 1.0$

To complete our Markov model, we impose a reflective barrier at $i = D$ and an absorbing state at $i = 0$ by imposing the constraints $P(S_{D-1,closer}|S_{D,farther}) + P(S_{D,equal}|S_{D,farther}) = 1$ and $P(S_{0,equal}|S_{0,equal}) = 1$, respectively.

For a given problem instance, we estimate the set of transition probabilities by analyzing the set of solutions visited by $TS_{Taillard}$ over a large number of independent trials. After each iteration $k \geq 1$ of each trial ($k = 0$ represents the initial solution), we compute (1) the distance $j = d_{opt}(s)$ from the current solution s to the nearest optimal solution and (2) the search gradient $x = grad(s, s_{-1})$ relative to the current solution, such that s_{-1} is the solution obtained at the end of the previous ($(k-1)$th) iteration. Given the neighboring solution s_{+1} of s selected for the next ($(k+1)$th) iteration, we then compute (3) the distance $i = d_{opt}(s_{+1})$ from s_{+1} to the nearest optimal solution and (4) the search gradient $x' = grad(s_{+1}, s)$ relative to the new solution s_{+1}. We denote the total number of samples observed in state $S_{j,x}$ by $\#(S_{j,x})$, and the total number of observed transitions from a state $S_{j,x}$ to a state $S_{i,x'}$ by $\#(S_{i,x'}|S_{j,x})$; both quantities are tracked over all iterations of all trials. We execute $TS_{Taillard}$ until $\#(S_{i,closer}) \geq 50$ for $1 \leq i \leq rint(\overline{d}_{lopt\text{-}opt})$, where the function $rint(x)$ returns the integer closest to x, rounding is upward if the fractional component of x equals 0.5. Individual trials are initiated from random local optima and terminated once an optimal solution is located. Because $TS_{Taillard}$ is (at least empirically) asymptotically complete, and because there is a non-zero probability of initiating a trial from a random local optimum that is distance $i \geq \overline{d}_{lopt\text{-}opt}$ from the nearest optimal solution, the termination criterion will eventually be satisfied as the number of trials approaches ∞.

We compute estimates of the transition probabilities using formulas such as $P(S_{i-1,closer}|S_{i,closer}) = \#(S_{i-1,closer}|S_{i,closer})/\#(S_{i,closer})$. Empirically, $\#(S_{i,closer}) \geq 50$ frequently occurs for $i > rint(\overline{d}_{lopt\text{-}opt})$. To estimate D, the maximal distance between an arbitrary solution and the nearest optimal solution, we first determine the minimal X satisfying $\#(S_{X,closer}) < 50$, i.e., the smallest distance at which samples are not consistently observed. We then define $D = X - 1$. Omitting states $S_{i,x}$ with $i > X$ has little impact on model accuracy, as we show below. Finally, we observe that our estimates of both the $P(S_{i,x'}|S_{j,x})$ and D are largely insensitive to both the initial solution and the sequence of solutions visited during the various trials, i.e., the statistics appear to be isotropic.

In Figure 7.16, we show the estimated probabilities of moving closer to (left figure) or farther from (right figure) the nearest optimal solution for a 10×10 random JSP. The probability of maintaining an *equal* search gradient is negligible ($p < 0.1$) at any distance i. We observe qualitatively similar results for all of our 6×4, 6×6, and 10×10 problem instances. As with ***RW***, the probability of continuing to move closer to (farther from) the nearest optimal solution is proportional (inversely proportional) to the current distance from the nearest optimum; the probabilities of moving closer to/farther from the nearest optimal solution are roughly symmetric around $D/2$. The result is a bias in $TS_{Taillard}$ toward solutions that are an "average" distance form the nearest optimal solution, and this bias accounts for the Gaussian-like distribution of d_{opt} for solutions visited

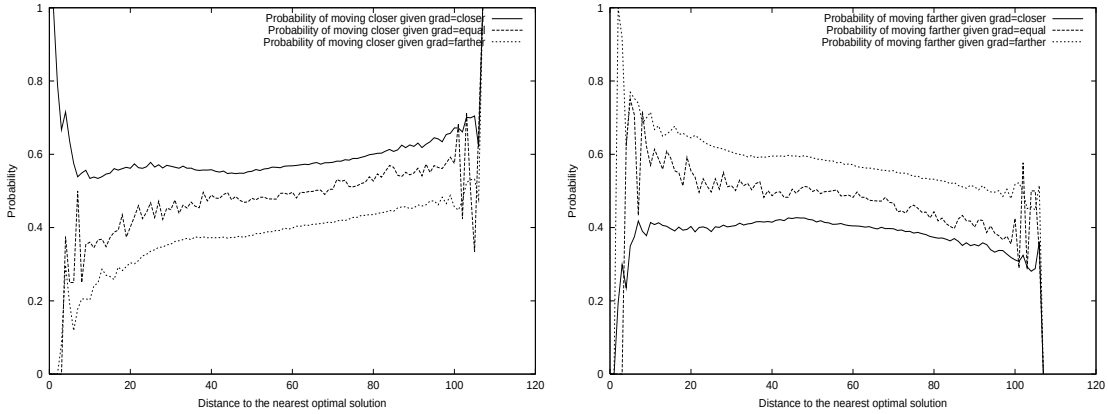

Figure 7.16: The transition probabilities for moving closer to (left figure) or farther from (right figure) the nearest optimal solution under $TS_{Taillard}$ for a 10×10 random JSP.

by $TS_{Taillard}$ during search (e.g., as shown in Figure 7.13). The impact of short-term memory, absent in *RW*, is also evident. Independently of i, the probability of maintaining the current search gradient is high and exceeds 0.5 in nearly all of the problem instances we analyzed. The bias, different from the bias away from optimal solutions, accounts for the strong trending behavior observed in the time-series of d_{opt} shown in the right side of Figure 7.15. The probability of inverting the current gradient is also a function of the distance to the nearest optimal solution and the degree of change. For example, the probability of switching from *equal* to *closer* is higher than the probability of switching from *farther* to *closer*. Finally, in contrast to the transition probabilities under *RW*, the probability of continuing to move closer to an optimal solution actually *rises* as $i \rightarrow 0$, typically approaching 1 after $i \geq 3$.

To validate our Markov model, we first compare the predicted versus actual mean search cost $\overline{c}$ for our 6×4 and 6×6 problem sets. For each problem instance, we compute the predicted $\overline{c}$ by repeatedly simulating the Markov chain defined by the estimated D, the set of states $S_{i,x}$, and the estimated transition probabilities $P(S_{i,x'}|S_{j,x})$. We initiate each simulation from a state $S_{m,n}$, where $m = d_{opt}(s)$ for a random local optimum (independently generated for each simulation trial) and $n =$ *closer* or $n =$ *farther* with equal probability; recall that the probability of maintaining an equi-distant search gradient is negligible. We compute m exactly in order to control for the possible effect of the distribution of random local optima, which tend to be more irregular (i.e., non-Gaussian) for small problem instances. By setting $n = rint(\overline{d}_{lopt\text{-}opt})$, we obtain a slight ($< 5\%$) decrease in model accuracy. We then define the mean predicted cost $\overline{c}$ as the mean number of simulated iterations required to achieve the state absorbing $S_{0,closer}$; statistics are taken over 10,000 simulations.

We show scatter-plots of the predicted versus actual $\overline{c}$ for our 6×4 and 6×6 instances in the top portion of Figure 7.17. The r^2 values of the corresponding log_{10}-log_{10} regression models are 0.9956 and 0.9966, respectively. For all instances the actual $\overline{c}$ is

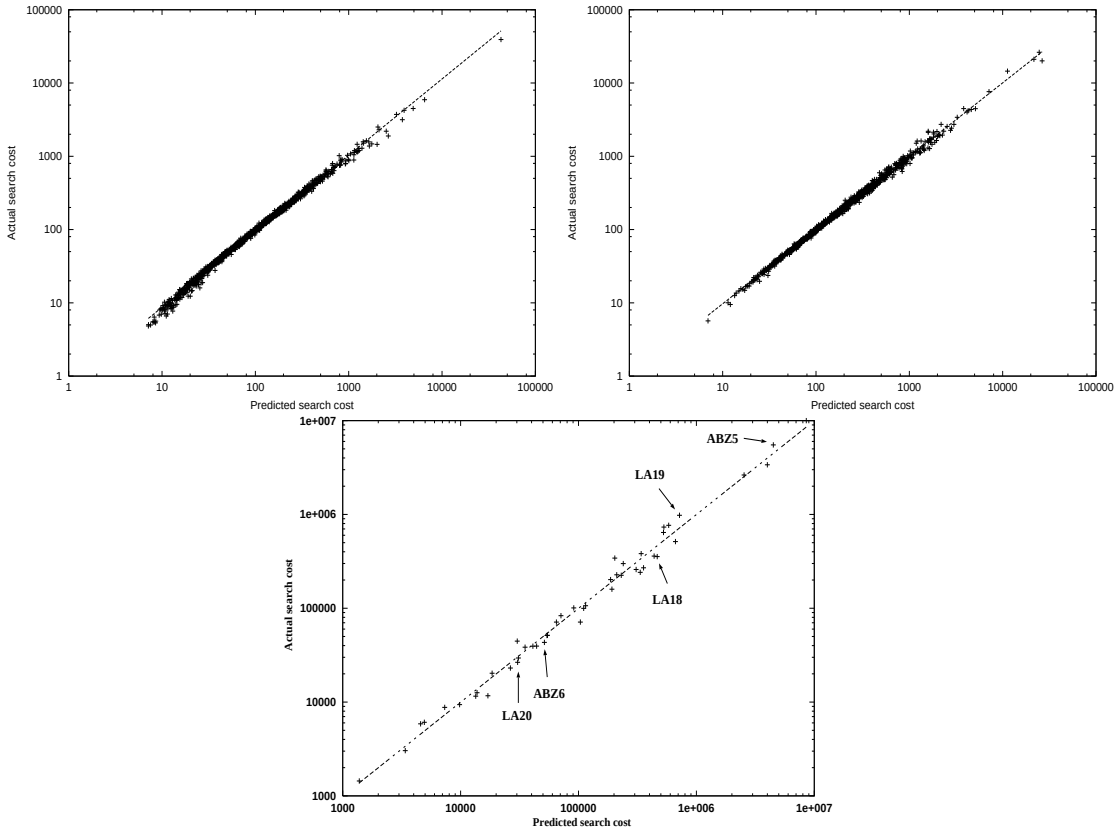

Figure 7.17: Scatter-plots of the predicted versus actual mean cost ($\overline{c}$) required to locate an optimal solution under *TS*$_{Taillard}$ to 6×4 (upper left figure), 6×6 (upper right figure), and 10×10 (lower figure) random JSPs; the least-squares fit lines are super-imposed.

within a factor of 2 (i.e., no more than 2 times and no less than 1/2) of the actual $\overline{c}$. To assess the scalability of our Markov model, we consider those 42 instances in our 10×10 problem set with $\leq$ 100,000 optimal solutions. Due to the large number of iterations required to solve these instances, we terminate our on-line estimation process once $\#(S_{i,closer}) > 30$ (as opposed to 50) for $1 \leq i \leq$ *rint*$(\overline{d}_{lopt\text{-}opt})$. A scatter-plot of the predicted versus actual $\overline{c}$ for these 10×10 instances is shown in the lower portion of Figure 7.17; the r^2 value of the corresponding log_{10}-log_{10} regression model is 0.9877. For reference, we also annotate the plot with the results for those 10×10 benchmark instances with $\leq$ 100,000 optimal solutions (i.e., `la18`-`la20` and `abz5`-`abz6`). As in the smaller problem sets, the predicted $\overline{c}$ is always within a factor of 2 of the actual $\overline{c}$. The slight decrease in accuracy is likely due to the less strict termination criterion for the on-line sampling procedure.

The accuracy of our dynamic model demonstrates that the high-level search process in *TS*$_{Taillard}$ is essentially a biased random walk over the S_{lopt+} sub-space. As with *RW*, the run-time dynamics can be viewed as a diffusion process with a central restoring force toward solutions that are an average distance from the nearest optimal solution. However,

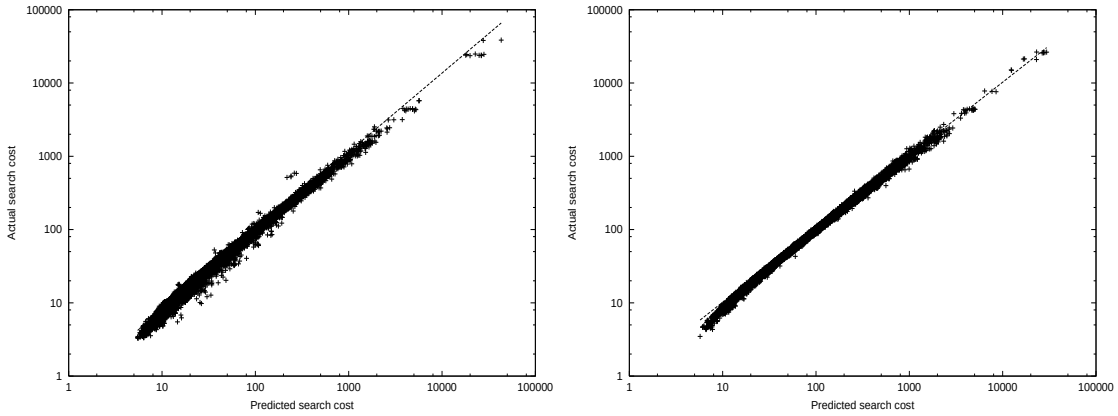

Figure 7.18: Scatter-plots of the predicted versus actual search cost ($\overline{c}$) for sub-optimal 6×4 (left figure) and 6×6 random JSPs; the regression lines are super-imposed.

the short-term memory induces an additional bias such that the current direction of the search gradient is significantly more likely to be maintained than under *RW*, i.e., there are two biases in $TS_{Taillard}$, as opposed to a single bias in *RW*. Interestingly, the impact of short-term memory is akin to a double-edged sword. If search is progressing toward an optimal solution, then short-term memory is beneficial. Conversely, if search is moving away from an optimal solution, the short-term memory is likely to move search even farther away from an optimal solution. Finally, in contrast to both the $\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{tabu\text{-}opt}$ models, our dynamic model appears to be scalable.

7.10 Explanatory Power and Applications of the Dynamic Cost Model

As with the static $\overline{d}_{lopt\text{-}opt}$ model, the power of our dynamic cost model is ultimately measured by its ability to both account for a wide range of observations regarding problem difficulty for $TS_{Taillard}$ and to predict the outcome of new experiments. Toward this goal, we now demonstrate the ability of our dynamic cost model to account for the cost of locating both sub-optimal solutions to typical random JSPs (in Section 7.10.1) and optimal solutions to rare-but-exceptionally-difficult random JSPs (in Section 7.10.2). Additionally, our model predicts a set of conditions under which heuristic initialization can improve the performance of $TS_{Taillard}$. We describe this prediction and its empirical verification in Section 7.10.3.

7.10.1 Modeling the Cost of Locating Sub-Optimal Solutions

Mirroring our analysis in Section 7.6.1, we first consider whether the exceptional accuracy of our dynamic cost model extends to the cost of locating sub-optimal solutions to

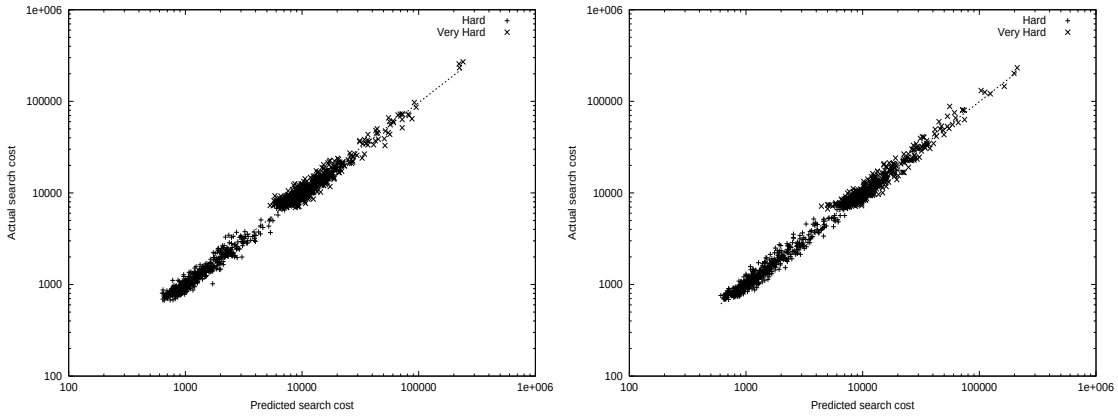

Figure 7.19: Scatter-plots of the predicted versus actual search cost $\overline{c}$ for hard and very hard 6×4 (left figure) and 6×6 (right figure) random JSPs.

problem instances. Using on-line sampling, we estimate both D and the set of transition probabilities $P(S_{i,x'}|S_{j,x})$ for all of the 6×4 and 6×6 sub-optimal random JSPs with $\leq$ 100,000 solutions; recall that the set of solutions to a sub-optimal instance offset X from the optimal makespan C_{max} consists of any solution with a makespan on the interval $[C_{max}, C_{max} + X]$. We perform filtering to ensure computational tractability, although we do retain the termination criterion of $\#(S_{i,closer}) > 50$ for $1 \leq i \leq rint(\overline{d}_{lopt\text{-}opt})$. Filtering eliminates 1,400 and 832 (on average easy) of the 25,000 total instances from the 6×4 and 6×6 problem sets, respectively. We then compute the predicted $\overline{c}$ via the simulation procedure described in Section 7.9.

We show scatter-plots of the predicted versus actual $\overline{c}$ in Figure 7.18. The r^2 values of the corresponding regression models are 0.9916 and 0.9971, respectively. The worst-case deviation is slightly higher in the 6×4 problem set, but occurs for very few instances relative to the total of 24,000. Conversely, for the 6×6 problem set the actual $\overline{c}$ is always within a factor of 2 of the predicted $\overline{c}$, as was the case for optimal solutions. In either case, the results conclusively demonstrate that search under $TS_{Taillard}$ for *both* optimal and sub-optimal solutions is a biased random walk of the form described in Section 7.9. The variability in the cost of locating sub-optimal solutions is primarily due to differences in D. Further, the fact that D is highly correlated with $\overline{d}_{lopt\text{-}opt}$ accounts for the ability of the $\overline{d}_{lopt\text{-}opt}$ static cost model to account for a large proportion of the variability in problem difficulty in the same problem sets.

7.10.2 Accounting for Variability in High-Cost Random JSPs

One key deficiency of the $\overline{d}_{lopt\text{-}opt}$ static cost model is that accuracy is inversely correlated with problem difficulty. As shown in Section 7.7.1, the actual $\overline{c}$ frequently deviates between 2 and 3 orders of magnitude from the predicted $\overline{c}$ for specially sets of hard and very hard 6×4 and 6×6 random JSPs. Given the previous described successes of

the dynamic cost model of $TS_{Taillard}$, it is natural to consider whether the same model can consistently account for the variability in problem difficulty observed for the most difficult random JSPs of a given fixed size. In doing so, the goal is to determine whether there is any evidence for variability in the accuracy of the dynamic cost model as a function of problem difficulty.

To test this hypothesis, the predicted $\overline{c}$ are computed for the *hard* and *very hard* sets of 6×4 and 6×6 random JSPs, using the methodology described in Section 7.9. Scatter-plots of the predicted versus actual $\overline{c}$ are shown in Figure 7.19. The r^2 values of the corresponding $log_{10} - log_{10}$ regression models are 0.9981 and 0.9985, respectively. The decrease in accuracy over the unfiltered 6×4 and 6×6 problem sets is minimal, although still evident (e.g., compare Figure 7.17). In all cases, the actual $\overline{c}$ is within a factor of 3 of the predicted $\overline{c}$. The absolute accuracy of the dynamic cost model remains remarkably high, and provides a dramatic improvement over the $\overline{d}_{lopt\text{-}opt}$ static cost model on the same problem sets, e.g., compare Figure 7.10. In summary, the results indicate that the accuracy of the dynamic cost model is only slightly sensitive to problem difficulty; the model still accounts for nearly all of the variability in problem difficulty.

7.10.3 The Impact of Initialization Method on Performance

Our models of $TS_{Taillard}$ are based on the assumption that search is initiated from a random local optimum. But can our models yield any insight into the impact of heuristic initialization on algorithm performance? Although researchers generally agree that high-quality initial solutions *can* improve the performance of tabu search algorithms (e.g., see [JRM00]), the exact conditions under which improvements can be achieved, and the expected degree of improvement, are poorly understood. In this section, we explore a particular aspect of this broader issue: What impact do different initialization methods, both heuristic and random, have on the cost required by $TS_{Taillard}$ to locate *optimal* solutions to problem instances?

To validate our Markov model, we only computed v_δ: the mean number of iterations required to locate an optimal solution, given a starting point that is (modulo rounding) distance $\overline{d}_{lopt\text{-}opt}$ from the nearest optimal solution. But what does our model predict if search is initiated from a solution that is either closer to or farther from the nearest optimal solution than δ? In Figure 7.20, we show plots of the predicted costs v_i over the full range of i for a 6×6 (left figure) and 10×10 (right figure) random JSP. For the 6×6 instance, search cost rises rapidly between $i = 3$ and $i = 10$, but only gradually increases once $i > 10$. In contrast, search cost for the 10×10 instance rises rapidly between $i = 2$ and $i \approx 15$, but is roughly *constant* (modulo the sampling noise) once $i > 15$. Even when $i = 2$, our model predicts that search cost is still significant: if the initial search gradient is not *closer*, search is driven toward solutions that are an "average" distance from the nearest optimal solution and any benefit of a favorable initial position is lost. We observed qualitatively identical behavior in a large sampling of our problem instances: for easy (hard) instances, the approach toward an asymptotic value as $i \rightarrow D$

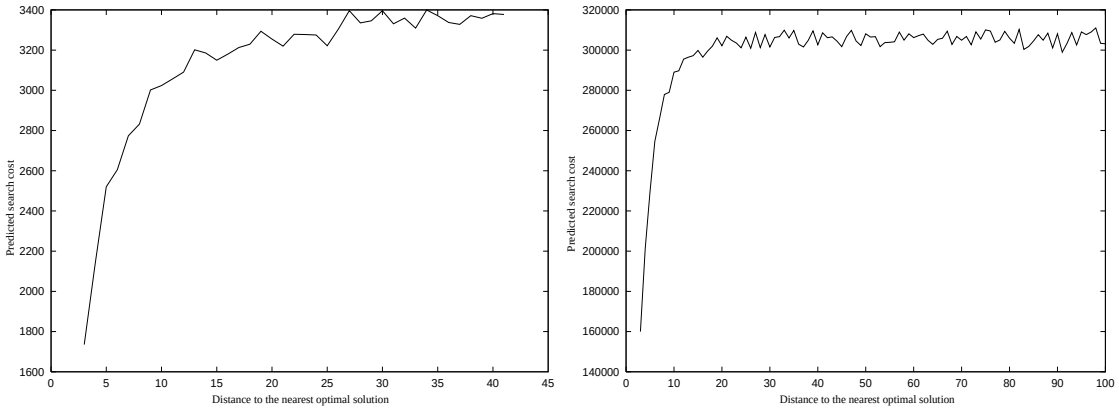

Figure 7.20: Predicted cost required by $TS_{Taillard}$ to locate an optimal solution, given an initial solution that is distance i from the nearest optimal solution, for two 6×6 (left figure) and 10×10 (right figure) random JSPs.

is gradual (rapid).

Our Markov model predicts that the distance to the nearest optimal solution, and not the fitness, dictates the benefit of a particular initialization method. The distinction is especially key in the random JSP, where fitness-distance correlation is known to be comparatively weak (see Chapter 11. In particular, our model predicts that an initialization method will at best have a minimal impact on search cost *unless* the resulting solutions are very close to the nearest optimal solution. To test this hypothesis, we analyzed the performance of $TS_{Taillard}$ using a variety of heuristic and random initialization methods. Following Jain et al. [JRM00], we consider the following set of high-quality priority dispatch rules (PDRs), used in conjunction with Giffler and Thompson's procedure for generating active schedules[GT60]: *FCFS* (First-Come, First-Serve), *LRM* (Longest ReMaining work), *MWKR* (Most WorK Remaining), and *SPT* (Shortest Processing Time). We additionally considered both active and non-delay solutions [GT60] generated using random PDRs, which we respectively denote RND_{active} and $RND_{nondelay}$. We denote our baseline random semi-active solutions by $RND_{semiactive}$. Finally, we examined Taillard's original lexicographic solution method, denoted *LEXICO*, and the insertion procedure introduced by Werner and Winkler [WW95], which we denote by *WW*; the latter is one of the best constructive heuristics available for the random JSP [JRM00]. As with $RND_{semiactive}$, we post-process the resulting solutions by applying a steepest-descent procedure to generate a local optimum.

For each initialization method, we computed $\overline{d}_{lopt\text{-}opt}$ (by substituting the optima generated by a particular initialization method for random local optima) for the forty-two 10×10 instances with $\leq 100,000$ optimal solutions. With the exception of *LEXICO*, all of the methods we consider are stochastic. Consequently, we define $\overline{d}_{lopt\text{-}opt}$ as the mean distance between $5,000$ random solutions and the nearest optimal solution. We show the resulting $\overline{d}_{lopt\text{-}opt}$ for each initialization method in Table 7.5. We also provide the p-values for the statistical significance of the mean difference in $\overline{d}_{lopt\text{-}opt}$ between

Initialization Method				
FCFS	*LRM*	*MWKR*	*SPT*	$RND_{semiactive}$
$\overline{d}_{lopt\text{-}opt}$				
58.49	97.41	97.94	64.97	70.92
Significance of Mean Difference in $\overline{d}_{lopt\text{-}opt}$ relative to $RND_{semiactive}$				
$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p = 0.1256$	$p = 1.0$
Percent Mean Difference in c_{Q2} relative to $RND_{semiactive}$				
1.76	2.32	2.94	1.55	0.0
Significance of Mean Difference in $log_{10}(c_{Q2})$ relative to $RND_{semiactive}$				
$p = 0.0594$	$p = 0.0836$	$p = 0.0727$	$p = 0.0769$	$p = 1.0$

Initialization Method				
LEXICO	RND_{active}	$RND_{nondelay}$	*WW*	$RND_{semiactive}$
$\overline{d}_{lopt\text{-}opt}$				
49.25	64.68	58.55	53.10	70.92
Significance of Mean Difference in $\overline{d}_{lopt\text{-}opt}$ relative to $RND_{semiactive}$				
$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p = 1.0$
Percent Mean Difference in c_{Q2} relative to $RND_{semiactive}$				
1.44	1.07	0.06	-2.79	0.0
Significance of Mean Difference in $log_{10}(c_{Q2})$ relative to $RND_{semiactive}$				
$p = 0.5129$	$p = 0.5730$	$p = 0.5555$	$p = 0.3090$	$p = 1.0$

Table 7.5: The differences in both the mean distance to the nearest optimal solution ($\overline{d}_{lopt\text{-}opt}$) and search cost ($c_{Q2}$) for various initialization methods, measured relative to random semi-active solutions ($RND_{semiactive}$).

the various methods and $RND_{semiactive}$, which we obtained using a Wilcoxon signed rank test. With the exception of *SPT*, we observe significant differences in $\overline{d}_{lopt\text{-}opt}$ between our baseline method ($RND_{semiactive}$) and the other initialization methods. Initially, this data appears to suggest that it may be possible to improve the performance of $TS_{Taillard}$ using initialization methods with low $\overline{d}_{lopt\text{-}opt}$. However, the lowest absolute values of $\overline{d}_{lopt\text{-}opt}$ (obtained using the *LEXICO* and *WW* methods) are still large. For our 10×10 instances, our Markov model predicts that these solutions are typically too far from the nearest optimal solution to have any impact on search cost (see the right side of Figure 7.20).

To test this hypothesis, we computed c_{Q2} using each initialization method (using $1,000$ independent trials of $TS_{Taillard}$) for each of the forty-two 10×10 instances with $\leq 100,000$ optimal solutions. We then computed the percent difference in c_{Q2} for each method relative to our baseline initialization method $RND_{semiactive}$; the results are shown in Table 7.5. We observe a *worst-case* deviation of less than 3%, and the best improvement (obtained under *WW*) is only 2.70%. Further, all observed discrepancies can be attributed to sampling error in the estimates of c_{Q2}, and in no case was the difference in search cost statistically significant (we provide the p-values resulting from a Wilcoxon signed-rank test in Table 7.5). The data clearly support the hypothesis predicted by

our dynamic model: for difficult problems, the choice of initialization method has no significant impact on the performance of $TS_{Taillard}$.

The results presented in this section concern the impact of initialization method on the cost required by $TS_{Taillard}$ to locate *optimal* solutions to *difficult* problem instances. Although beyond the scope of this paper, our Markov model also predicts that initialization methods can significantly impact the cost of locating both optimal solutions to easy or moderate problem instances and good *sub-optimal* solutions to a wide range of problem instances. We have confirmed these predictions experimentally. Finally, we note that our model says nothing about the impact of initialization method on the performance of tabu search algorithms that employ re-intensification, such as Nowicki and Smutnicki's algorithm [NS96]; we are currently investigating this issue.

7.11 The Relationship Between the Models

In hindsight, the initial success of the $\overline{d}_{lopt\text{-}opt}$ cost model was due to the fact that $\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{tabu\text{-}opt}$ are highly correlated for small problem instances. The two measures diverge as problem size increases, ultimately leading to a lack of scalability in the $\overline{d}_{lopt\text{-}opt}$ model. The degree of divergence also appears to be related to problem difficulty, providing a likely explanation for the inverse relationship between accuracy and problem difficulty in the $\overline{d}_{lopt\text{-}opt}$ model.

What remains is to establish a link between the $\overline{d}_{tabu\text{-}opt}$ model and the dynamic model. Recall that the general qualitative form of the estimated transition probabilities, as shown in Figure 7.16, is identical in all of the problem instances we analyzed. Any major differences are due to variability in D, a more direct measure of the size of the S_{lopt+} than either $\overline{d}_{lopt\text{-}opt}$ or $\overline{d}_{tabu\text{-}opt}$. Viewed another way, D defines the overall "ideal" form of the transition probabilities, with smaller-scale deviations due to the details of a particular problem instance. The transition probabilities are also roughly symmetric around $D/2$ such that search in $TS_{Taillard}$ is biased toward solutions that are, on average, approximately distance $D/2$ from the nearest optimal solution. But $D/2$ is approximately equal to $\overline{d}_{tabu\text{-}opt}$; any discrepancies in the accuracies of the two models are likely due to small deviations in the transition probabilities from the ideal form. Thus, we believe the success of the $\overline{d}_{tabu\text{-}opt}$ model is due to the fact that it estimates the parameter D of the dynamic model. To conclude, the relationship between the three models can summarized by the following equation, analogous to an equation linking the static, quasi-dynamic, and dynamic cost models of *RW*: $\overline{d}_{lopt\text{-}opt} \approx \overline{d}_{tabu\text{-}opt} \approx D/2$.

7.12 The Impact of Makespan Estimation

Computing the makespans of neighboring solutions typically consumes over 95% of the overall run-time of any local search algorithm for the JSP. Consequently, several researchers have attempted to reduce the overall run-time by *estimating* the makespans

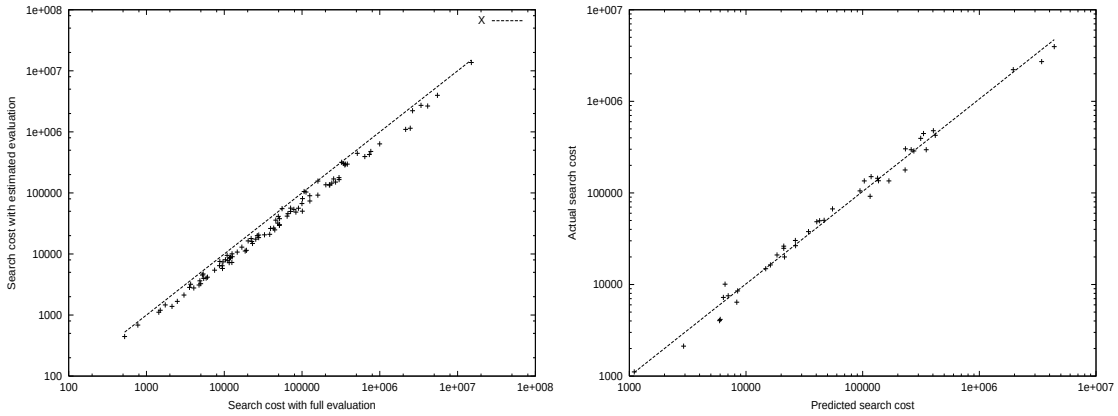

Figure 7.21: Left Figure: Scatter-plot of the search cost $\overline{c}$ for 10×10 random JSPs using exact versus estimated makespans of neighboring solutions; the line $y = x$ is super-imposed. Right Figure: Scatter-plot of the predicted versus actual search cost ($\overline{c}$) for 10×10 random JSPs when using estimated makespans of neighboring solutions; the least-squares fit line is super-imposed.

of neighboring solutions. Taillard introduced a method for estimating the makespans of neighboring solutions under the $N1$ operator (see Section 3.5.2), and used such a scheme in his tabu search algorithm for the JSP. In contrast, we have in all prior experiments used a version of *TS$_{Taillard}$* that computes the makespans of neighboring solutions exactly. Our motivation was to control for the possible impact of makespan estimation on our static cost models, which we now analyze.

In the "estimated" version of *TS$_{Taillard}$*, we use the equations described in Section 3.5.2 to compute a lower bound on the makespan of each neighboring solution $s' \in N1$ of the current solution s, which can be done in time $O(|N1(s)|)$. After selection of a neighbor s' for the next iteration, the heads (est_{ij}) and tails ($tail_{ij}$) of s' are updated using the procedures documented in Section 3.5.2 (at which point the exact makespan of s is known).

First, we analyze the impact of estimation on the cost $\overline{c}$ required to locate optimal solutions to our 10×10 random JSPs; statistics are taken over 1,000 independent trials. For these instances, the cost-per-iteration ratio of actual versus estimated computations in our implementation of *TS$_{Taillard}$* is roughly 7:1. We show a scatter-plot of $\overline{c}$ under actual versus estimated neighbor makespan computation in the left side of Figure 7.21. On average, approximately 33% fewer *iterations* are required when using *TS$_{Taillard}$* with neighbor makespan estimation. Coupled with the cost-per-iteration ratio, estimation clearly provides a significant improvement in performance. Few comparable results have been reported in the literature. Nowicki and Smutnicki [NS01] indicate that for their TSAB tabu search algorithm, estimation actually resulted in a net *decrease* in performance, relative to the number of iterations performed. However, overall performance relative to run-times was not reported, although they indicate the cost-per-iteration ratio was more

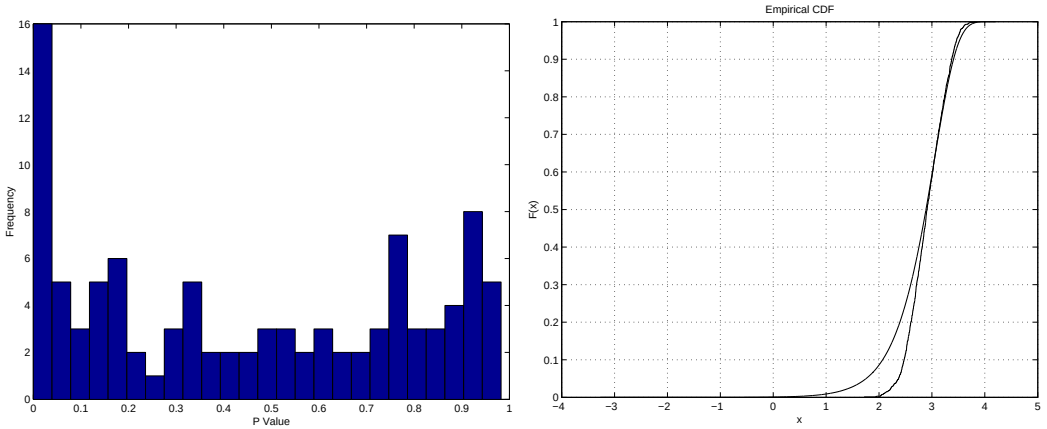

Figure 7.22: Left Figure: p-values for 100 10×10 instances for rejecting the null hypothesis that the actual run-length distributions are exponentially distributed. Right Figure: The actual and exponential run-length distributions for the 10×10 instance with the smallest p-value.

than 10:1.

Next, we consider the impact of estimation on the accuracy and structure of our cost models. We developed a dynamic cost model of the estimated version of $TS_{Taillard}$ for those 42 instances from our 10×10 problem set with $\leq$ 100,000 optimal solutions, computing the conditional transition probabilities $P(S_{i,x'}|S_{j,x})$ and the maximal distance D_{max} using the methodology described in Section 7.9. We show a scatter-plot of the resulting predicted versus actual search cost $\overline{c}$ in the right side of Figure 7.21; the r^2 value of the corresponding regression model is 0.9899, unexpectedly indicating that estimation of neighbor makespans has no impact on the overall accuracy of our dynamic cost model. Rather, makespan estimation has the effect of reducing the run-time costs required per iteration, with little impact on the run-time dynamics.

7.13 Run-Length Distributions

We conclude our analysis of $TS_{Taillard}$ with an investigation of the full run-length distribution of search costs. As we noted in Section 7.3, Taillard ([Tai94], p. 116) indicated that the number of iterations required to locate optimal solutions using his algorithm was approximately exponentially distributed. However, he only reported results for a single problem instance. We now rigorously test whether the actual RLDs under $TS_{Taillard}$ are exponentially distributed for our 10×10 instances. Mirroring the approach in Section 6.8, we test this hypothesis using a two-sample Kolmogorov-Smirnov (KS) goodness-of-fit test. The first sample consists of the actual search costs c obtained over 1,000 independent trials. We generate the second sample by drawing 1,000,000 random samples from an exponential distribution with a mean $\overline{c}$ computed from the first sample.

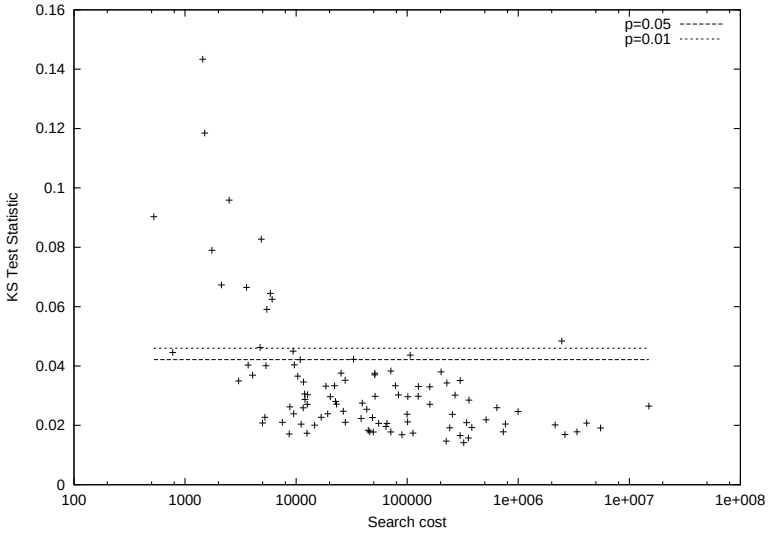

Figure 7.23: Scatter-plot of search cost versus the value of the Kolmogorov-Smirnov test statistic for comparing the actual search cost distribution with that of an exponential distribution. Large values of the test statistic indicate more significant differences. The horizontal lines indicate null hypothesis rejection thresholds at significant $p = 0.01$ and $p = 0.05$.

We show the distribution of the p-values associated with the resulting KS test statistics in the left side of Figure 7.22. At $p \leq 0.01$, we reject the hypothesis that the RLD is exponentially distributed for 11 of the 100 instances, a lower proportion than observed for *RW*. We show the CDF of both the actual and "mean" exponential distribution for the instance with the smallest (i.e., worst-case) p-value in the right side of Figure 7.22. Mirroring the results for *RW*, most differences are concentrated in the left-tail mass of the two distributions. Further, the largest differences are associated with the easiest problem instances, as shown in Figure 7.23.

Next, we analyze whether our dynamic cost model can effectively predict the full RLD under $TS_{Taillard}$, and not just the mean search cost $\overline{c}$. Again, we use a two-sample Kolmogorov-Smirnov test to test the hypothesis that the actual and predicted RLDs originate from the same underlying distribution. As with our test for exponential fit, the first sample consists of the actual c observed over 1,000 independent trials. Analogously, the second sample consists of those 10,000 c values used to generate the predicted $\overline{c}$ (as discussed in Section 7.9). We only report results for those 42 instances for which transition probability estimation was tractable. For all but 6 of the 42 instances, we reject the null hypothesis that the two distributions are identical at $p \leq 0.01$; despite the accuracy of the dynamic cost model, it generally fails to account for the full cost distribution. However, the predicted and actual RLDs are, with a few exceptions, qualitatively identical: the actual RLD can be closely approximated by shifting the predicted RLD along the x-axis. Consequently, as was the case for *RW*, such discrepancies are likely due

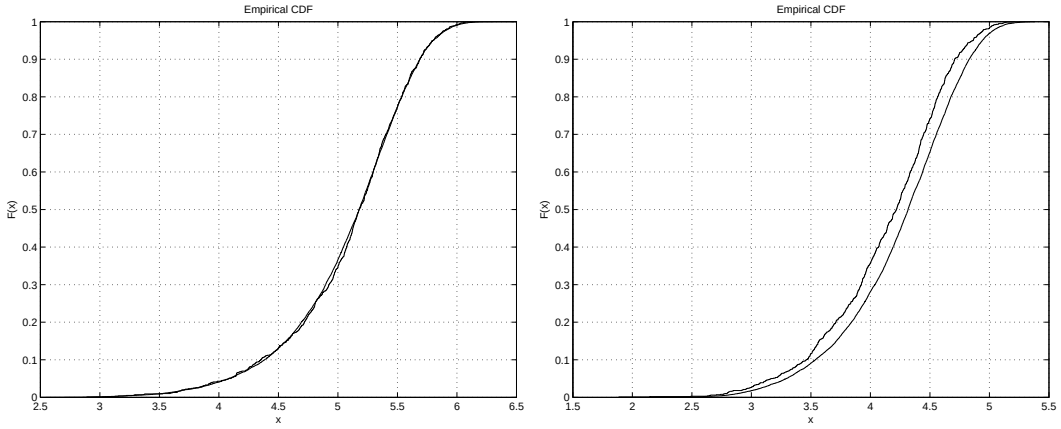

Figure 7.24: CDFs of the predicted and actual RLDs for two 10×10 instances. The p-values for the KS test statistic are respectively 0.4506 and 4.2062×10^{-8}.

to inaccuracies introduced in our on-line estimation process, and not to some inherent structural flaw in our dynamic model – as expected (Section 4.5), near-perfect dynamic cost models are necessary to achieve statistically accurate prediction of the full RLD.

Chapter 8

Iterated Local Search

Computational studies of the JSP indicate that local search algorithms in general, and tabu search algorithms in particular, provide the best overall performance on the range of available benchmark instances [BDP96] [JM99]. We now turn to the question of whether other meta-heuristics *can* in fact compete with tabu search on the JSP, or if other meta-heuristics somehow induce biases that are inherently more detrimental to search effectiveness. The success of tabu search may be due in part to historical contingency. The first effective local search algorithms for the JSP were based on tabu search. These initial algorithms fueled a wave of follow-on research, often to the exclusion of other meta-heuristics. The intense focus on tabu search has also led to numerous highly efficient implementations, which may mask the true potential of other meta-heuristics with less efficient implementations.

In this chapter and in Chapter 9, we develop new local search algorithms for the JSP that *are* competitive with tabu search. To explain their effectiveness, we develop accurate cost models of these algorithms and contrast them with cost models of tabu search. The models indicate that the search biases associated with these algorithms are no less detrimental, and possibly more advantageous, than those observed for tabu search. Specifically, we consider the development and analysis of a new iterated local search (ILS) algorithm for the JSP, which we denote *I-JAR*. We begin in Section 8.1 with an overview of the iterated local search meta-heuristic, and review prior research on iterated local search and the JSP in Section 8.2. We motivate, define, and analyze the performance of a version of *I-JAR* based on the $N1$ operator in Section 8.3. Cost models of *I-JAR* are developed in Section 8.4. In Section 8.6, we assess scalability by introducing a version of *I-JAR* based on the high-performance $N5$ move operator.

8.1 An Overview of Iterated Local Search

One promising alternative to tabu search is iterated local search, or ILS. The origins of ILS date to Baum [Bau86]. Widespread interest in the ILS meta-heuristic is largely due to Martin et al. [MOF91], who introduced an effective implementation of ILS for

```
function Iterated Local Search (MaxIters )
        s' = s_init
        s'' = LocalSearch(s', N_small)
        s = s''
        s_best = s
        i = 0
        repeat
                s' = GenPerturbation(s, N_large)
                s'' = LocalSearch(s', N_small)
                if F(s'') < F(s_best) then
                        s_best = s''
                s = ChooseNext(s, s'')
                i = i + 1
        until i eq MaxIters
        return s_best
```

Figure 8.1: Pseudo-code for the iterated local search meta-heuristic.

the TSP; variations of Martin et al.'s algorithm have also been developed and analyzed by Johnson and McGeoch [JM97]. Since the introduction of Martin et al.'s algorithm, researchers have developed ILS implementations for a number of combinatorial optimization problems, often achieving competitive if not state-of-the-art performance. Lourenco et al. [LMS03] provide an excellent overview of the motivation, history, and recent literature on ILS.

In contrast to tabu search and simulated annealing, ILS employs *two* move operators: N_{small} and N_{large}. As in iterated descent, ILS uses the N_{small} move operator in conjunction with greedy descent to convert solutions into local optima. However, instead of re-initiating greedy descent from a new random solution, ILS relies on the N_{large} move operator to generate perturbations to the current local optimum s. Ideally, the perturbations are large enough to escape the attractor basin of s, such that if greedy descent (under N_{small}) is applied to a neighbor $s' \in N_{large}(s)$, the resulting local optimum s'' will differ from s. An acceptance criterion is then applied to determine whether the next iteration will begin from s'' or s.

We provide pseudo-code for the ILS meta-heuristic in Figure 8.1. As in all single-solution meta-heuristics for local search, ILS proceeds via a sequence of incremental modifications to some initial solution s_{init}. ILS is executed for *MaxIters* iterations, where *MaxIters* is a user-specified parameter. The best local optimum encountered during search (s_{best}) is tracked and returned upon termination. To generate an instantiation of ILS for a particular problem, an algorithm designer must define three functions. The first function, LocalSearch, transforms the input solution into a local optimum using N_{small}.

The second function, GenPerturbation, selects a neighbor $N_{large}(s)$ of the input solution s. The third function, ChooseNext, selects one of the two input solutions to serve as the starting point for the next iteration.

In practice, the LocalSearch function is typically a variant of greedy descent, although more complex strategies have been used. The complexity of the GenPerturbation function varies considerably; the most straightforward method simply selects a random neighbor $s' \in N_{large}$. Finally, three common implementations of the ChooseNext function are found in the literature on ILS:

- **always**, in which s'' is always accepted.
- **better-than**, in which s'' is accepted only if $F(s'') \leq F(s)$.
- **Metropolis**, in which s'' is always accepted if $F(s'') \leq F(s)$, and otherwise with probability $e^{(F(s)-F(s''))/T}$; T is a user-supplied constant known as the *temperature*.

8.2 Iterated Local Search and the JSP: Prior Research

The few reported implementations of ILS for the JSP do appear to exhibit good overall performance, suggesting that ILS may in fact be a viable alternative to tabu search. However, several factors conspire to prevent an accurate assessment of performance, such that ILS is now generally accepted as inferior to tabu search [BDP96] [JM99]. First, one of the two implementations of ILS for the JSP is a hybridization with tabu search, and the impact of ILS on the performance of the aggregate algorithm was not analyzed [LZ96]. Second, experimental methodologies used to evaluate existing ILS algorithms are very weak, e.g., limited to single or a few short trials on each problem instance [Lou93]. Third, direct performance comparisons of ILS with other meta-heuristics have been limited to relatively short runs of simulated annealing [Lou95].

Lourenço [Lou93] [Lou95] introduced a series of ILS algorithms for the JSP, differing in the type of large-step move operator, local search algorithm, acceptance criterion, and initialization method. In all variants $N1$ served as the small-step move operator, and the initial solution was generated via the well-known shifting bottleneck procedure [ABZ88]. Two methods were used to post-process solutions resulting from large-step moves: next-descent and limited-time simulated annealing. Each of the acceptance criteria discussed in Section 8.1 was considered. Using a factorial experiment design, Lourenço analyzed the impact of the various components on ILS performance and concluded that (1) simulated annealing out-performed next-descent as the local search component and (2) the acceptance criterion had no discernible impact on performance. However, it is difficult to make general conclusions from these experiments, as the run-times were limited such that at most a few thousand iterations of ILS were executed, often for a single trial, and for a limited number of problem instances.

In terms of algorithm design, Lourenço's research was notable because it introduced the first large-step move operators for the JSP. Two of Lourenço's large-step operators generate neighboring solutions by re-sequencing the jobs on a pair of random machines by solving single-machine scheduling problems to optimality, using either the well-known algorithm defined by Carlier [Car82] or a novel approach introduced by Lourenço. Both large-step operators yield reasonable performance in experimental studies. Lourenço also introduced two less effective large-step move operators, one that takes a different approach to re-sequencing the jobs on a single random machine and another that inverts adjacent pairs of critical and non-critical operations.

Lourenço and Zwijnenburg [LZ96] report results for an implementation of ILS that uses tabu search to post-process the solutions resulting from the large-step move operator, indicating that the hybrid was able to outperform the stand-alone tabu search algorithm, in addition to ILS in conjunction with simulated annealing. Jain [Jai98] also uses a version of the Carlier-based large-step operator to explore the immediate neighborhood around high-quality solutions, but does not use the operator in conjunction with ILS.

8.3 *I-JAR*: Iterated Jump-And-Redescend

The analysis presented in Section 5.2 indicates that the attractor basins of local optima in the JSP can be escaped with high probability by accepting a short sequence of monotonically dis-improving moves. This fact has an immediate implication for the design of large-step move operators for ILS: strong "kick-moves" are simply not required to escape local optima. This observation raises two hypotheses regarding the behavior of ILS in particular, and local search algorithms in general. First, it seems unlikely that complex large-step moves operator based on problem-specific knowledge, e.g., those introduced by Lourenço [Lou93][Lou95], are necessary to achieve high-performance implementations of ILS for the JSP. Second, because meta-heuristics differ primarily in their approach to escaping local optima, it seems unlikely that complex meta-heuristics are required to achieve high-performance local search *in general*. In other words, we are hypothesizing that many meta-heuristics are currently over-engineered for their intended purpose: to escape local optima. We now exploit the relative weakness of attractor basins in the JSP by introducing a new iterated local search algorithm for the JSP and evaluating its performance relative to both optimal solutions of benchmark problems and a well-known tabu search algorithm introduced by Taillard [Tai94]. By analyzing the resulting performance, we obtain evidence for the two hypotheses proposed above. We begin with a description of the algorithm.

```
function I-JAR(MaxIters, k, OWL_nom, OWL_inc)

    s' = s_init
    s'' = nextDescent(s', N1)
    s = s''
    s_best = s
    i = 1
    oem=false
    repeat
        if (i mod OCI) eq 0 then
            if C_max(s) eq C_max(s_-x) then
                oem=true
                OWL = OWL_nom
        if oem eq true then
            s'=genRandMove(s,OWL,N1)
        else
            s'=genEscapeMove(s,k,N1)
        s''=nextDescent(s',N1)
        if oem eq true then
            if C_max(s'') eq C_max(s) then
                OWL = OWL + OWL_inc
            else
                oem=false
        if C_max(s'') < C_max(s_best) then
            s_best = s
        s = s''
        i = i + 1
    until i eq MaxIters;
    return s_best;
```

Figure 8.2: Pseudo-code for the *I-JAR* iterated local search algorithm.

8.3.1 *I-JAR*: Algorithm Definition

I-JAR stands for *I*terated *J*ump *A*nd *R*edescend. Although potential applications for *I-JAR* extend beyond the JSP, we refer to the general concept and the specific implementation for the JSP interchangeably. The small-step operator is $N1$, while the large-step operator simply accepts a random sequence of at most k monotonically disimproving neighbors of the current local optimum s. *I-JAR* is initiated from a random local optimum, and continues until a user-specified termination criterion is satisfied. All

large-step moves are accepted. The novelty of *I-JAR* stems from the fact that the large-step move operator is (1) based on search space properties rather than problem-specific knowledge and (2) is expressed directly in terms of the small-step

To enable escape from local minima that are also either local maxima or near local maxima, or on large plateaus (see Section 5.2.4), *I-JAR* continually monitors the progress of search. After every *OCI* iterations of search, *I-JAR* compares the makespan of the current solution s with that of the solution s_{-OCI} encountered *OCI* iterations prior. If $C_{max}(s) = C_{max}(s_{-OCI})$, then search is likely either trapped on a large plateau or in a difficult-to-escape local optimum; the criterion is conservative, as the solutions s and s_{-OCI} may be members of different plateaus, potentially in distant regions of the search space. If the condition is satisfied, *I-JAR* enters a random walk mode. Let *OWL* and OWL_{nom} denote the current and nominal walk length, respectively. Initially, $OWL = OWL_{nom}$. In each iteration of the random walk mode, *I-JAR* accepts a random sequence of at most *OWL* moves from the current solution s, yielding a new solution s'; the qualifier is necessary because it is possible, albeit unlikely, to encounter an optimal solution from which no further moves are possible. As during normal operation (i.e., iterations when *I-JAR* is not performing a random walk), s' is transformed into a local optimum s'' via randomized next-descent. If $C_{max}(s) = C_{max}(s'')$, then *OWL* is incremented by a factor OWL_{inc}; otherwise, the normal application of the large-step operator is resumed in the next iteration.

Pseudo-code for *I-JAR* is shown in Figure 8.2. The behavior of the functions escapeMove, randomMove, and nextDescent is self-explanatory. The variable *oem* (an acronym for *o*ptima *e*scape *m*ode) tracks whether the large-step move in the current iteration is a sequence of random solutions or strictly dis-improving solutions. Most of the complexity of *I-JAR* is dedicated to handling the most infrequently encountered cases, when search becomes trapped.

I-JAR also is related to variants of simulated annealing that have been reported in the literature. In "basin-hopping" [WS99], a variable-strength perturbation is applied to the current local optimum to generate an intermediate solution s', which is then transformed via greedy descent into a local minimum s''. The new local minimum s'' is accepted with a probability dictated by the Metropolis rule. The strength of the perturbation is increased when $s'' = s$, and decreased when $s'' \neq s$. Search in *I-JAR* is also related to implementations of simulated annealing in which the temperature oscillates between 0 and ∞, e.g., see [BK94]. The random walk mechanism also bears some similarity to the adaptive short-term memory mechanisms found in reactive tabu search algorithms [BT94].

8.3.2 Escape Probabilities Under *I-JAR*

I-JAR is based on the assumption that with very few exceptions, local optima can be escaped by accepting a short sequence of monotonically dis-improving moves; the random walk mechanism serves to handle the remaining cases. Although we have established

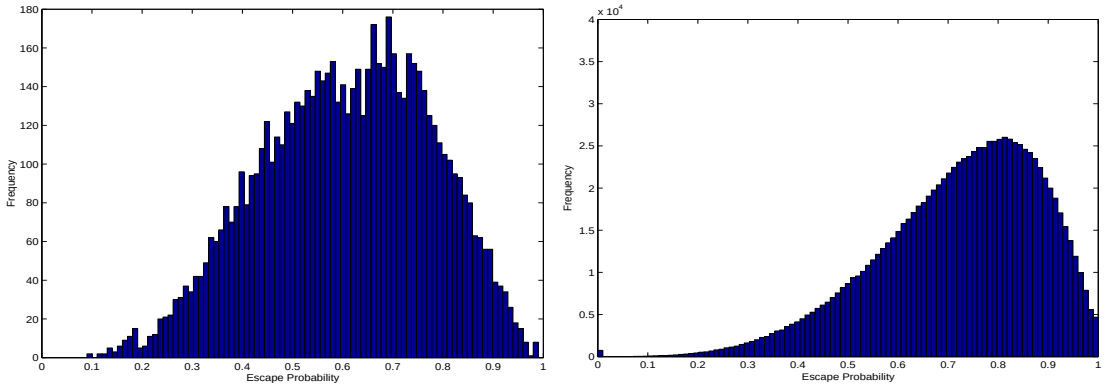

Figure 8.3: Left figure: Histogram of the escape probabilities (next-descent, $k = 3$) for local optima visited by *I-JAR* during search; results are for a typical 10×10 JSP. Right figure: Histograms of the escape probabilities (next-descent, $k = 3$) for random local optima; aggregate data for 100 10×10 JSPs are displayed.

this fact for unbiased samples of random local optima, it remains unclear whether the same observation holds for high-quality local optima, local optima close to the nearest optimal solution, or even local optima visited by *I-JAR* during search.

We consider the escape probabilities for local optima visited by *I-JAR* on our 10×10 random JSPs. For each problem instance, we execute a variable number of independent trials of *I-JAR*, tracking the total number of local optima X_i generated at distance i from the nearest optimal solution, for $2 \leq i \leq \textit{rint}(\overline{d}_{\textit{lopt-opt}})$. We terminate each trial once an optimal solution is located (again, because no moves may exist under $N1$ from an optimal solution) and terminate all trials once $X_i > 50$ for $2 \leq i \leq \textit{rint}(\overline{d}_{\textit{lopt-opt}})$. Using the methodology described in Section 5.2.4, we compute estimates of the escape probabilities for the *first* 50 local optima resulting from successful escape iterations at each distance i; the latter condition is imposed to prevent over-representation of strong local optima. This methodology yields a uniform sample of local optima visited by *I-JAR*, ranging from both small-to-medium distances from the nearest optimal solution and high-to-moderate quality.

In the left side of Figure 8.3, we show the distribution of escape probabilities from the resulting local optima for a typical 10×10 random JSP; data for 7,096 local optima are shown, generated using $k = 3$ under randomized next-descent. We compare this distribution with the corresponding distribution for random local optima, as shown in the right side of Figure 8.3 (the latter is identical to the figure appearing in the center portion of Figure 5.6 in Chapter 5). The presence of a left-skew indicates that local optima visited by *I-JAR* are somewhat more difficult to escape than random local optima, given a fixed k. Similar results hold for k ranging from 1 to 5, and for escape probabilities under steepest-descent.

Next, we consider the correlation between both the distance to the nearest optimal solution d_{opt} and the makespan C_{max}, and P_{escp}. Scatter-plots for the 10×10 instance

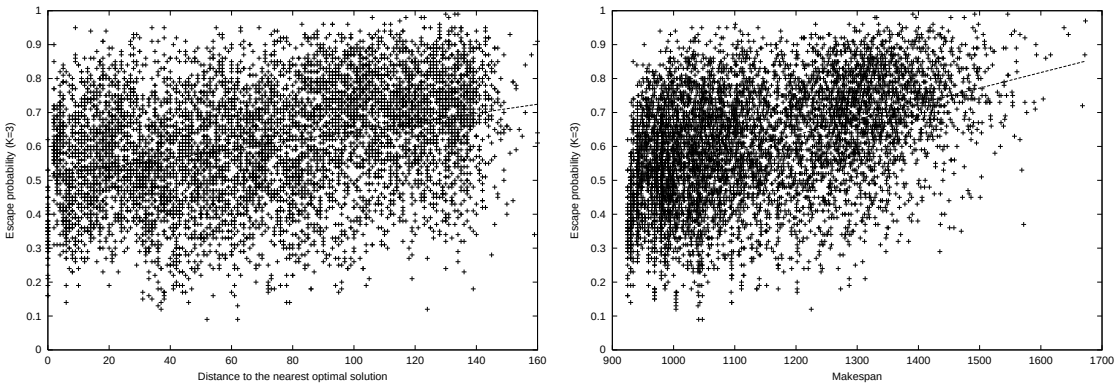

Figure 8.4: Scatter-plots of the distance to the nearest optimal solution (left figure) and makespan (right figure) versus the escape probability under next-descent for a typical 10×10 random JSP; the least-squares fit lines are super-imposed. The corresponding (Pearson's) r-values are 0.3271 and 0.4269, respectively.

analyzed above are shown in Figure 8.4; the data were obtained using $k = 3$ under randomized next-descent. The respective (Pearson's) r-values are 0.3271 and 0.4269. Clearly, the correlation is not significant enough to exploit by dynamically adjusting k in response to the makespan $C_{max}(s)$ of the current optimum s. Finally, we observe that the correlation is proportional to k, achieving $r \approx 0.5$ once $k = 5$, and falling to $r \approx 0.1$ when $k = 1$.

Our experimental results provide no *a priori* evidence to suggest that *I-JAR* might fail to escape local optima in random JSPs, or otherwise become trapped in any particular region of the fitness landscape. Consequently, we would expect the behavior of *I-JAR* to be asymptotically complete. Our results necessarily limit this conjecture to 10×10 random JSPs; we provide indirect evidence (via a demonstration of performance) of the escapability of local optima in larger JSPs below in Sections 8.3.3 and 8.6.

8.3.3 Assessing the Potential of *I-JAR*

The performance of *I-JAR* is dictated primarily by k, the maximal length of the jump taken before next-descent is re-initiated. The escape probability is proportional to k: as k increases, the proportion of iterations that yield escapes from the current local optimum increases. But the CPU time required per iteration also increases, due to increases in both k and the length of the re-descent. Thus, any increase in k must be balanced by a decrease in the total number of iterations c required to achieve a particular objective. Consider the mean number of iterations $\overline{c}$ required to locate optimal solutions to our 10×10 random JSPs using *I-JAR*, computed using the results of 1,000 independent trials. Increasing k from 2 to 3 and 4 yield 17.26% and 16.31% reductions in $\overline{c}$, respectively, while the mean CPU time per iteration (using the computing environment described below) increases approximately 40% and 60%. The result is a net *increase* in run-

		I-JAR			$TS_{Taillard}$		
Instance	Optimal Makespan	Min.	Mean	Max.	Min.	Mean	Max.
ta01	1231	1233	1242	1246	***1231***	**1237.4**	1244
ta02	1244	***1244***	**1244.7**	1247	***1244***	1250.6	1256
ta03	1218	***1218***	**1221.5**	1224	***1218***	1222.3	1223
ta04	1175	***1175***	**1180.4**	1181	***1175***	1180.8	1182
ta05	1224	1229	**1232.4**	1233	*1228*	1232.8	1236
ta06	1238	*1239*	**1243.2**	1246	1240	1243.3	1246
ta07	1227	*1228*	**1228**	1228	*1228*	**1228**	1228
ta08	1217	***1217***	**1218.1**	1222	***1217***	1220.1	1227
ta09	1274	***1274***	**1281.6**	1287	***1274***	1282.3	1289
ta10	1241	***1241***	**1243.6**	1244	1244	1250.2	1256

Table 8.1: Statistics for the makespans of the best solutions obtained by *I-JAR* and $TS_{Taillard}$ on Taillard's 15×15 benchmark instances. Statistics are taken over 30 independent trials. Bold-faced entries in a "Min" column indicate equality with the optimal makespan. Italicized entries in a "Min" Column indicate the best makespan achieved by either algorithm. Bold-faced entries in a "Mean" column indicate the mean makespan was less than or equal to that of the competing algorithm.

time for $k > 2$. Similar results hold for the 15×15 instances analyzed below. With $k = 2$, roughly 60% of all iterations yield an escape from the current local optimum in 10×10 and 15×15 JSPs. The fraction of iterations in which a random walk is performed is less than 0.1%. The impact of k on larger problem instances is considered in Section 8.6. In all of our experiments, $OCI = 50$, $OWL_{nom} = 3$, and $OWL_{inc} = 1$. These particular values were arrived at via testing, although the performance of *I-JAR* is largely insensitive to any particular reasonable (i.e., not exceedingly large) selection of values.

To demonstrate the effectiveness of *I-JAR*, we measure performance relative to our own implementation of the $TS_{Taillard}$ tabu search algorithm analyzed in Chapter 7. Our implementations of *I-JAR* and $TS_{Taillard}$ are based on identical code for computing the makespans of neighboring solutions under the $N1$ operator, which for both algorithms consume over 98% of the total run-time of any given trial. By using a common implementation of the move operator, we are able to mitigate the impact of implementation efficiency on our results. All algorithmic trials are executed on 1.5 GHz Pentium IV hardware with 512 Mb of memory, running the Linux 2.4.18-5 operating system; all code was compiled using gcc 3.2 with level 3 optimization. For both 10×10 and 15×15 random JSPs, iterations of *I-JAR* are roughly 3.1 times more expensive than iterations of $TS_{Taillard}$. Iterations in *I-JAR* are more costly because they consist of an ascent/descent phase, in addition to a full neighborhood evaluation of at least once solution – the local optimum s''. In contrast, an iteration in $TS_{Taillard}$ consists of a full neighborhood evalua-

tion of a single solution.

We first consider the mean number of iterations $\overline{c}$ required to locate optimal solutions to Fisher and Thompson's infamous `ft10` instance; statistics are taken over 30 independent trials. Although `ft10` is not a random JSP (it is approximately workflow), the effectiveness of a new algorithm for the JSP is historically first measured relative to this instance. For $TS_{Taillard}$, we let $L_{min} = 8$, $L_{max} = 14$, and update the tabu tenure every 15 iterations; these parameter settings are identical to those used by Taillard in his analysis of `ft10` [Tai94]. For *I-JAR* and $TS_{Taillard}$, $\overline{c}$ respectively equaled 3.5 and 16.7 million iterations. As with random 10×10 JSPs, iterations of *I-JAR* on `ft10` are approximately 3 times more expensive than iterations $TS_{Taillard}$, indicating that *I-JAR* actually *outperforms* $TS_{Taillard}$ on this particular instance.

Next, we consider a set of ten 15×15 instances from the OR Library, introduced by Taillard [Tai93]. Although these instances are closed, in that the optimal makespans are known, they are by no means easy: even state-of-the-art algorithms can have difficulty consistently locating optimal solutions. For each instance, we ran 30 independent of trials of *I-JAR* and $TS_{Taillard}$ for 10 million and 30.1 million iterations, respectively. The tabu tenure parameters of $TS_{Taillard}$ are set via the heuristic described in Section 7.3. The CPU run-times for all trials are roughly equal, each requiring approximately 2 hours. We use a large number of iterations in order to assess the best possible performance of each meta-heuristic. We express performance in terms of the number of iterations (as opposed to raw CPU time) to enable replicability. Undoubtedly, the efficiency of our $N1$ evaluation code can be improved, but our objective is algorithm analysis.

We report statistics for the makespans of the best solutions obtained for the various trials in Table 8.1. Both algorithms achieve excellent performance in an absolute sense, each locating optimal solutions to (a different set of) 6 of the 10 instances. No one algorithm dominates in terms of consistently locating the best overall solution. In terms of mean solution quality, *I-JAR* out-performs $TS_{Taillard}$ on 8 of the 10 instances, although in no case is the difference statistically significant. The performance of *I-JAR* is slightly less variable than that of $TS_{Taillard}$, e.g., for instances `ta02` and `ta10`. In any case, we conclude that, given equal run-times, *I-JAR* performs *no worse* than $TS_{Taillard}$. In other words, given identical move operators and under controlled experimental conditions, iterated local search can be competitive with tabu search on the JSP.

8.4 Cost Models of *I-JAR*

The performance of *I-JAR* suggests that a simple mechanism for escaping the attractor basins of local optima is sufficient to yield a high-quality implementation of a local search algorithm for the JSP. However, we have yet to provide insight into why this should be the case. By definition, search in *I-JAR* is restricted to the sub-space $S_{opt} \subseteq S$ of local optima. As with $TS_{Taillard}$, any high-level or global search strategy is emergent or implicit. Consequently, the logical null hypothesis is that *I-JAR* is simply performing

	I-JAR				*TS*$_{Taillard}$			
Problem Set	$\|optsols\|$	$\overline{d}_{lopt\text{-}lopt}$	$\overline{d}_{lopt\text{-}opt}$	$\overline{d}_{ijar\text{-}opt}$	$\|optsols\|$	$\overline{d}_{lopt\text{-}lopt}$	$\overline{d}_{lopt\text{-}opt}$	$\overline{d}_{tabu\text{-}opt}$
6×4	0.4826	0.2146	0.8059	0.7874	0.5365	0.2415	0.8044	0.8441
6×6	0.1647	0.2511	0.6591	0.6552	0.2223	0.2715	0.6424	0.7808
10×10	0.1740	0.1667	0.3934	0.6859	0.184	0.2744	0.4598	0.6641

Table 8.2: r^2 values of static and quasi-dynamic cost models for *I-JAR* and *TS*$_{Taillard}$.

a random walk over the sub-space S_{opt}. As with *RW* and *TS*$_{Taillard}$, we expect search cost in I-JAR to be proportional to the effective size of the search space, taking into account the number and/or distribution of optimal solutions. The primary unknown in modeling *I-JAR* as a random walk is whether biases exist in the transition probabilities that explain its effectiveness, and how the transition probabilities are related to those observed for other meta-heuristics. We specifically examine whether there exists pressure toward solutions that are an average distance from the nearest optimal solution and if I-JAR somehow induces additional search biases.

We test the random walk hypothesis by developing a series of cost models of *I-JAR*, mirroring our prior analysis of *RW* and *TS*$_{Taillard}$. We develop these models using our 6×4, 6×6, and 10×10 sets of random JSPs. The number of iterations c required by *I-JAR* to locate optimal solutions is a random variable with an approximately exponential distribution, as shown below in Section 8.5. Thus, we run 1,000 independent trials of *I-JAR* on each instance to achieve stable estimates of the mean ($\overline{c}$) or median (c_{Q2}) search cost. *I-JAR* is, at least empirically, asymptotically complete; in no case has *I-JAR* failed to eventually locate an optimal solution to these problem instances.

8.4.1 Static and Quasi-Dynamic Cost Models

We first consider static cost models of *I-JAR* based on three fitness landscape features related to the effective size of the S_{opt} sub-space: the number of optimal solutions ($|optsols|$), the mean distance between random local optima ($\overline{d}_{lopt\text{-}lopt}$), and the mean distance between random local optima and the nearest optimal solution ($\overline{d}_{lopt\text{-}opt}$). All measures are computed for each of the 6×4 and 6×6 instances. As indicated in Chapter 5.3, we are only able to compute $|optsols|$ for those 97 of the 100 10×10 instances with ≤ 250 million optimal solutions, and $\overline{d}_{lopt\text{-}opt}$ for those 92 10×10 instances with ≤ 50 million optimal solutions; we computed $\overline{d}_{lopt\text{-}lopt}$ for all 10×10 instances, as it is independent of the number of optimal solutions.

We show the r^2 values for the corresponding regression models in Table 8.2; for reference, we also include results obtained for *TS*$_{Taillard}$, as reported in Chapter 7. In all cases, the dependent variable is the base-10 logarithm of the median search cost c_{Q2}; given an exponential distribution, the median is a more stable estimate of "average" search cost than $\overline{c}$. For all three measures, we observe the same qualitative trends in accuracy for both *I-JAR* and *TS*$_{Taillard}$. All models fail to scale with increases in problem size. Independently, neither $|optsols|$ (which measures the number of "targets" in S_{opt})

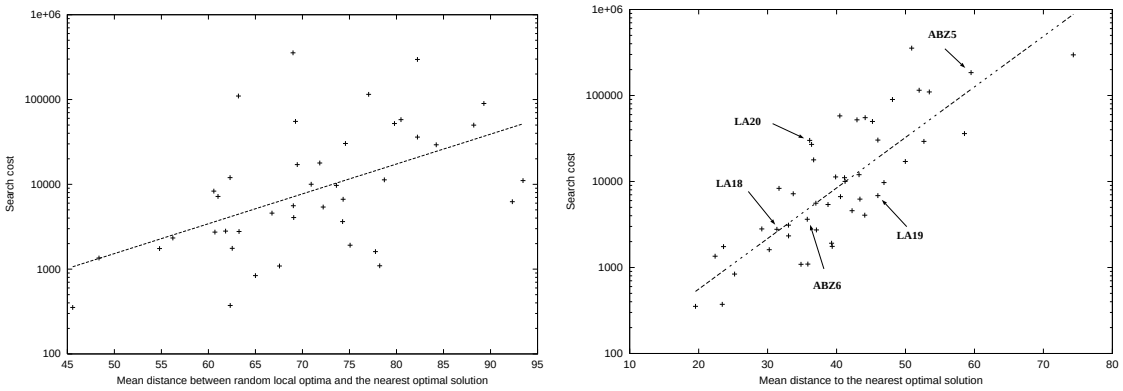

Figure 8.5: Scatter-plots of $\overline{d}_{lopt\text{-}opt}$ (left figure) and $\overline{d}_{ijar\text{-}opt}$ (right figure) versus search cost (c_{Q2}) for 10×10 random JSPs; the least-squares fit lines are super-imposed.

nor $\overline{d}_{lopt\text{-}lopt}$ (which measures the absolute size of S_{opt}) account for any significant proportion of the variability in search cost. The most accurate model is based on $\overline{d}_{lopt\text{-}opt}$, which accounts for roughly 40% of the variability in search cost for 10×10 instances. For reference, we show a linear-log scatter-plot of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} in the left side of Figure 8.5. Mirroring the results obtained for $TS_{Taillard}$, the actual c_{Q2} typically deviates from the predicted c_{Q2} by as much as one order of magnitude, exceeding 1.5 orders of magnitude in the worst case. Overall, the results provide only relatively weak support of our random walk hypothesis, as the $\overline{d}_{lopt\text{-}opt}$ cost model, which measures the effective size of the S_{opt}, fails to scale to larger problem instances.

Analogous to the situation for both *RW* (in Section 6.3) and $TS_{Taillard}$ (in Section 7.8), random local optima are *not* representative of the set of local optima visited by *I-JAR* during search. In particular, the distribution d_{opt} of the distance to the nearest optimal solution for random local optima and those generated by *I-JAR* diverges as problem size increases, possibly accounting for the failure of the $\overline{d}_{lopt\text{-}opt}$ model to scale. Let $\overline{d}_{ijar\text{-}opt}$ denote the mean d_{opt} for local optima visited by *I-JAR*. For each problem instance, $\overline{d}_{ijar\text{-}opt}$ is estimated from 100,000 local optima. The optima are generated by executing *I-JAR* for a variable number of independent trials. Trials are initiated from random local optima and terminated once an optimal solution is located; the latter condition is required for situations in which no moves are possible from an optimal solution. The entire process, including the current trial, is terminated once 100,000 iterations have been performed; samples consist of the local optimum generated at each iteration of each trial.

The r^2 values for the quasi-dynamic $\overline{d}_{ijar\text{-}opt}$ model are also shown in Table 8.2; the values for the analogous $\overline{d}_{tabu\text{-}opt}$ model of $TS_{Taillard}$ are included for purposes of comparison. For the 6×4 and 6×6 problem sets, the difference in accuracy between the $\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{ijar\text{-}opt}$ models is insignificant, which is consistent with minimal observed differences in the distribution of d_{opt} for random local optima and local optima generated by *I-JAR*. However, in contrast to the results for $TS_{Taillard}$ (and for *RW*, as discussed

in Section 6.3), we observe *no* increase in model accuracy when considering $\overline{d}_{ijar\text{-}opt}$ over $\overline{d}_{lopt\text{-}opt}$; in fact, the $\overline{d}_{lopt\text{-}opt}$ model is actually slightly *more* accurate than the $\overline{d}_{ijar\text{-}opt}$ model. We currently have no explanation for this discrepancy.

For the 10×10 instances, computation of $\overline{d}_{ijar\text{-}opt}$ is only tractable for those 42 instances with ≤ 10 million optimal solutions. Here, in contrast to the smaller problem sets, we obtain roughly a 250% increase in accuracy over the static $\overline{d}_{lopt\text{-}opt}$ cost model. This is consistent with the often significant differences between the distributions of d_{opt} for random local optima and local optima generated by *I-JAR*. A linear-log scatter-plot of $\overline{d}_{ijar\text{-}opt}$ versus c_{Q2} for these instances is shown in the right side of Figure 8.5. The r^2 value indicates that $\overline{d}_{ijar\text{-}opt}$ model accounts for roughly 2/3 of the variability in search cost, with the actual search cost deviating from the predicted value by no more than an order of magnitude, and in most cases far less. Further, there is little evidence of increasing variability as $\overline{d}_{ijar\text{-}opt} \rightarrow \infty$.

As in *RW* and $TS_{Taillard}$, search in *I-JAR* is driven toward solutions that are, on average, quite distant from the nearest optimal solution. Post-hoc intuition suggests that search cost is therefore proportional to the amount of bias or pressure away from optimal solutions that must be overcome. This would appear to contradict the random walk hypothesis, which posits that search cost is proportional to the effective size of the S_{opt} sub-space. However, as we indicate below, $\overline{d}_{ijar\text{-}opt}$ is really nothing more than a measure of the effective size of the S_{opt} sub-space; the interpretation of the measure as the amount of bias that search must overcome is due to the qualitative form of the transition probabilities, which we discuss next in Section 8.4.2.

8.4.2 A Dynamic Cost Model

The static and quasi-dynamic cost models only provide indirect and approximate evidence for the hypothesis that search in *I-JAR* is simply a random walk over the sub-space S_{opt} of local optima. We now support this hypothesis directly by constructing a dynamic cost model, which enables us to identify the presence and relative magnitudes of any search biases and to account for precisely why $\overline{d}_{ijar\text{-}opt}$ (and to a lesser extent $\overline{d}_{lopt\text{-}opt}$) is so highly correlated with search cost. The model also enables a more fundamental analysis of the dynamical run-time behavior of *RW*, *I-JAR*, and $TS_{Taillard}$.

Because it is memoryless, a dynamic run-time behavior of *I-JAR* can be modeled as a simple Markov chain. Solutions (i.e., local optima) are aggregated based on their distance i from the nearest optimal solution. Let S_i, $2 \leq i \leq D$, denote a state representing the set of all local optima that are distance i from the nearest optimal solution. The parameter D represents the maximal possible distance between a local optimum and the nearest optimal solution. Let $P(S_i|S_j)$ denote the condition probability of a transition from a state S_j to a state S_i. The set of conditional probabilities is subject to the total-probability constraints $\sum_{x=0}^{D} P(S_x|S_j) = 1$ for $1 \leq j \leq D$. To complete the Markov model, we introduce a reflective barrier at $i = D$ and an absorbing state at $i = 0$ by imposing the constraints $P(S_x|S_i) = 0$ for all i such that $x > D$ and $P(S_0|S_0) = 1$,

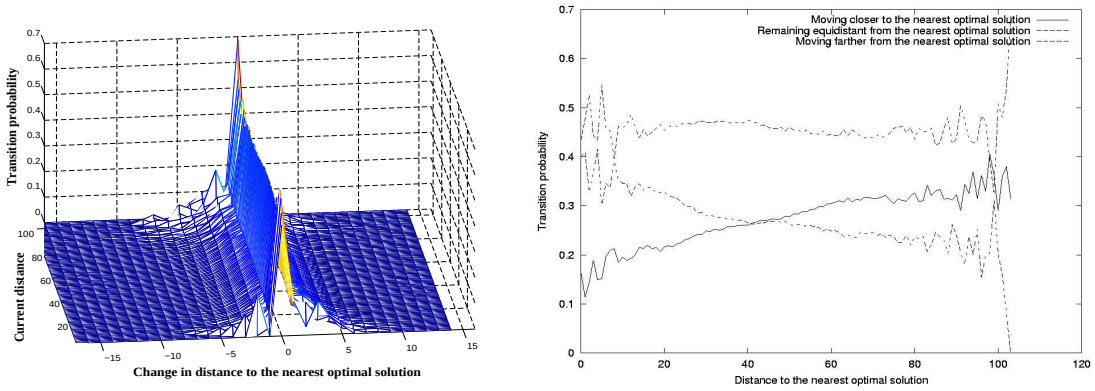

Figure 8.6: Left figure: Transition probabilities -full. Right figure: Aggregate transition probabilities for the same problem instance.

respectively.

In *RW* and $TS_{Taillard}$, random local optima are not representative of solutions visited during search – in terms of both the distribution of their distance to the nearest optimal solution and their transition probabilities. Consequently, sample-based estimation of transition probabilities yields inaccurate dynamic cost models. In preliminary experimentation, we found the same holds in I-JAR. To correct for this deficiency, we again turn to on-line estimation. As indicated in Chapters 6 and 7, this process is not circular. First, transition probabilities are estimated using a different set of trials than those used to compute either the mean search cost $\overline{c}$ or the full run-length distribution. Second, there is no guarantee that dynamic models based on analyses of mean run-time behavior will reflect the actual search process.

For a given problem instance, the set of transition probabilities is estimated via an analysis of the local optima generated by *I-JAR* over a variable number of independent trials. At each iteration of a trial, the distance i between the current local optimum s and the nearest optimal solution is determined. Given the local optimum s'' generated from s, we then compute the distance j from s'' to the nearest optimal solution. Let $\#(S_i)$ denote the number of occurrences of state S_i, and let $\#(S_i|S_j)$ denote the number of occurrences of a successor state S_i given a current state S_j. We execute *I-JAR* until $\#(S_i) \geq 50$ for $2 \leq i \leq$ *rint*$(\overline{d}_{lopt\text{-}opt})$; individual trials are initiated from random local optima, and terminated once an optimal solution is located. The function *rint* returns the integer nearest to the input value, rounding up when the fractional component equals 0.5. The significance of the *rint*$(\overline{d}_{lopt\text{-}opt})$ term is discussed below. Although not theoretically guaranteed, e.g., the state S_2 need never be visited, the termination criterion is empirically satisfied for all of our test instances. Estimates of the transition probabilities are computed using the formula $P(S_i|S_j) = \#(S_i|S_j)/\#(S_j)$. Because $\#(S_i) \geq 50$ is possible, if not likely, for $i >$ *rint*$(\overline{d}_{lopt\text{-}opt})$, we take $D = X - 1$, where X is the minimal value satisfying $\#(S_X) < 50$. To yield a consistent model (necessary due to the conservative estimation of D), for all $i > D$ we add $\#(S_i|S_j)$ to $\#(S_D|S_j)$

before estimating the transition probabilities.

Empirically, omitting states S_i with $i > \textit{rint}(\overline{d}_{\textit{lopt-opt}})$ has negligible impact on model accuracy. As discussed below, the qualitative form of the transition probabilities indicates that *I-JAR* rarely encounters such states, and there is no evidence that search is likely to become trapped in these states. Further, due to their rarity, accurate estimation of transition probabilities is problematic. Finally, we observe that our estimates of the transition probabilities are largely insensitive to both the initial local optima and the sequence of local optima visited during individual trials; i.e., the statistics appear to be isotropic.

The estimated transition probabilities for a typical 10×10 random JSP are shown in the left side of Figure 8.6. Given a state S_j, $P(S_i|S_j) \approx 0$ for $|j - i| > 5$, subject to the obvious boundary conditions; for $|j - i| \leq 5$, the probabilities are approximately Gaussian with mean S_i. Similar observations hold in a limited sampling of our 6×4, 6×6, and 10×10 problem sets. Although not immediately apparent, there also exists a significant bias in the relative values of $\sum_{i=0}^{j-1} P(S_i|S_j)$ and $\sum_{i=j+1}^{D} P(S_i|S_j)$, in that the probability of moving either closer to or farther from the nearest optimal solution varies with i. The aggregate transition probabilities for the same 10×10 instance indicated above are shown in the right side of Figure 8.6. Although the overall escape probability is roughly constant at ≈ 0.5, the probability of moving closer to (farther from) the nearest optimal solution is proportional (inversely proportional) to the current distance i. In other words, there is a restoring force that biases search toward local optima that are an "average" distance from the nearest optimal solutions. This is the same bias – due to the structure of the binary hypercube – that exists in both *RW* and $\textit{TS}_{\textit{Taillard}}$.

To validate our dynamic cost model, we simply compare the predicted versus actual mean search costs $\overline{c}$. The predicted $\overline{c}$ for each instance is computed by repeatedly simulating the Markov chain defined by D, the set of states S_i, and the estimated transition probabilities $P(S_i|S_j)$. Each simulation is initiated from a state S_i, where i is the distance between a random local optima s (specific to the individual simulation) and the nearest optimal solution, and terminated once the state S_0 is entered. The cost of an individual simulation is defined as the total number of iterations c. The aggregate $\overline{c}$ is then computed using 10,000 independent simulations. We use the distance between actual random local optima and the nearest optimal solution to control for any possible impact of the distribution of $d_{\textit{opt}}$ for random local optima. Slightly less accurate results, particularly on the smaller problem sets, are obtained via repeated simulation from state S_x, where $x = \textit{rint}(\overline{d}_{\textit{lopt-opt}})$, i.e., when search is initiated from solutions that are, on average, approximately distance $\overline{d}_{\textit{lopt-opt}}$ from the nearest optimal solution.

We first consider the results for our 6×4 and 6×6 problem sets. Scatter-plots of the predicted versus actual $\overline{c}$ are shown in the top portion of Figure 8.7. The r^2 values of the corresponding $\textit{log}_{10}$-$\textit{log}_{10}$ regression models are 0.9673 and 0.9854, respectively. Although the actual $\overline{c}$ is *typically* within a factor of two of the predicted $\overline{c}$, there exist instances for which the actual $\overline{c}$ is off by factors of 4 and 3 in the 6×4 and 6×6 problem sets, respectively. Somewhat unexpectedly, model accuracy is actually higher in

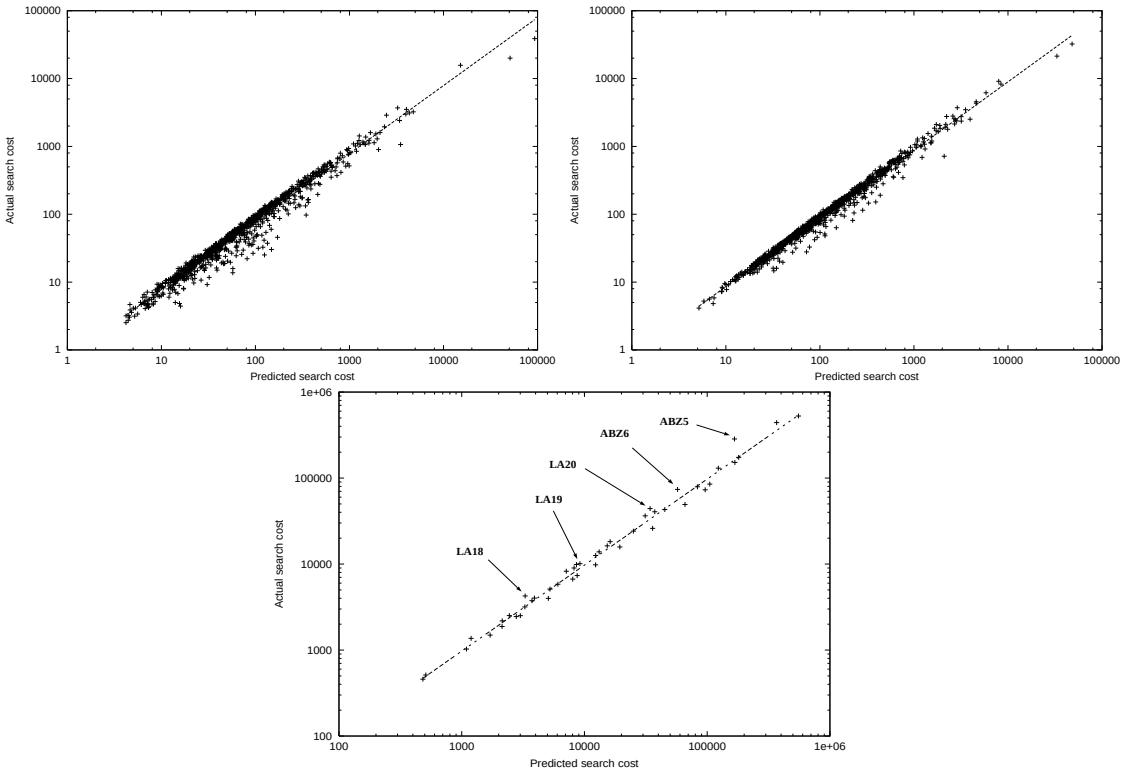

Figure 8.7: Scatter-plots of the observed versus predicted mean cost to locate an optimal solution under IJAR 6×4 (upper left figure), 6×6 (upper right figure), and 10×10 (lower figure) random JSPs; the least-squares fit lines are super-imposed.

the more difficult 6×6 problem set. In both cases, absolute accuracy is high, although slightly lower than that observed for the dynamic cost models of both *RW* and $TS_{Taillard}$. We currently have no explanation for unexpectedly high inaccuracies on particular problem instances.

Next, we analyze the results for our restricted 10×10 problem set (i.e., only containing instances with $\leq$ 100,000 optimal solutions). A scatter-plot of the predicted versus actual $\overline{c}$ is shown in the bottom portion of Figure 8.7; the r^2 value of the corresponding log_{10}-log_{10} regression model is 0.9935. For reference, we also include results for those 10×10 random JSP benchmark instances with $\leq$ 100,000 optimal solutions. In all cases, the actual $\overline{c}$ is within a factor of two of the predicted $\overline{c}$. There is no evidence of high-residual instances such as those observed in the 6×4 and 6×6 problem sets, although there is a slight chance that this may be due to the decrease in the sample size (from 1,000 to 42). The absolute accuracy is equivalent to the dynamic cost models developed for both *RW* and $TS_{Taillard}$ on the same 10×10 instances.

These results provide strong evidence for the hypothesis that search in *I-JAR* is nothing more than a random walk over the space of local optima. More surprisingly, the runtime dynamics are qualitatively identical to those of *RW*. In both *RW* and *I-JAR*, search

is biased toward solutions that are roughly equi-distant between optimal solutions and solutions that are maximally distant from optimal solutions. This bias is largely due to the structure of the binary hypercube. The key difference between the two algorithms is the search space: search *I-JAR* is constrained to S_{opt}, while search in *RW* is unrestricted within S, the space of feasible solutions. Even though *I-JAR* is theoretically capable of large-distance jumps, in practice they do not occur, further minimizing the differences between the two algorithms. Similarly, the fundamental difference between $TS_{Taillard}$ and *I-JAR* is the presence (in the former) of a secondary bias that acts to move search either consistently toward or away from the nearest optimal solution.

Finally, the relationship between the static, quasi-dynamic, and dynamic cost models that was identified in Sections 6.7 and 7.11.

8.5 Run-Length Distributions under *I-JAR*

As with *RW* and $TS_{Taillard}$, we found strong evidence that search cost, as measured by the number of iterations required to locate an optimal solution, is *approximately* exponentially distributed for *I-JAR*. As in Sections 7.13 and 6.8, we test this hypothesis using a two-sample Kolmogorov-Smirnov (KS) goodness-of-fit-test. The first sample consists of the actual search costs c obtained over 1,000 independent trials. We generate the second sample by drawing 1,000,000 random samples from an exponential distribution with a mean $\overline{c}$ computed from the first sample.

The distribution of resulting p-values for our 100 10×10 random JSPs is shown in the left side of Figure 8.8. At $p \leq 0.01$, the null hypothesis that the run-length distribution (RLD) is exponentially distributed is rejected for 36 of the 100 instances, higher than the numbers rejected for $TS_{Taillard}$ (11 of 100 on the same problem set) and *RW* (22% of our 6×6 instances). Where differences do exist, they are concentrated in the left-tail mass of the two distributions, and are not extremely large. For illustrative purposes, we show the CDF of the actual and "mean" exponential distribution for the 10×10 instance with the smallest p-value in the right side of Figure 8.8. Finally, the largest differences are associated with the easiest problem instances, as shown in Figure 8.9.

Next, we analyze the ability of our dynamic cost model to account for the full RLD under *I-JAR*, as opposed to just the mean cost $\overline{c}$. We use a two-sample Kolmogorov-Smirnov test to test the null hypothesis that the actual and predicted RLDs originate from the same underlying distribution. The first sample consists of the actual c observed over 1,000 independent trials, while the second sample consists of those 10,000 c values used to generate the predicted $\overline{c}$. For 25 of the 42 10×10 instances (in contrast to 6 of 42 instances under $TS_{Taillard}$), the KS test statistic was significant at $p \leq 0.01$, indicating that our dynamic cost model cannot in general account for the full distribution of search cost. However, as with *RW* and $TS_{Taillard}$, the worst-case deviations are minimal, and the two distributions are often nearly identical; we show CDFs of the predicted and actual RLDs for the instances with the smallest and largest p-value in Figure 8.10. Where differences

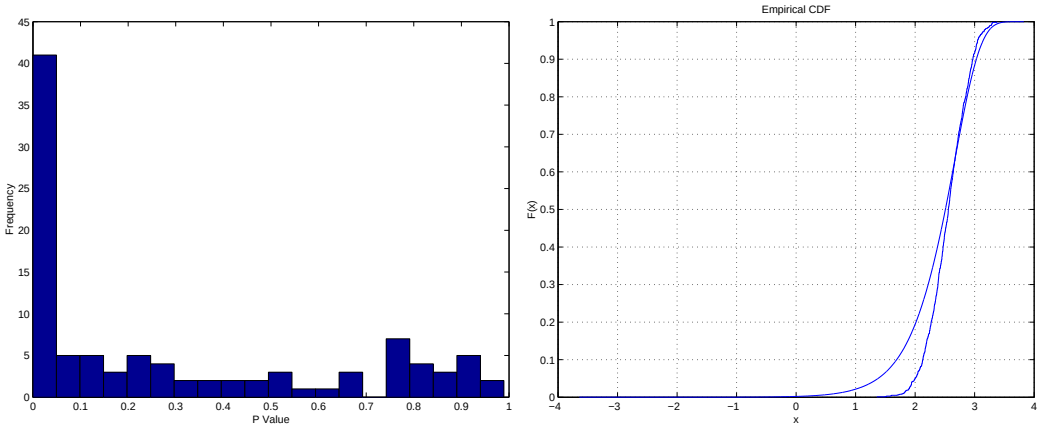

Figure 8.8: Left Figure: p-values for 100 10×10 instances for rejecting the null hypothesis that the actual run-length distributions are exponentially distributed. Right Figure: The actual and exponential RLDs for the 10×10 instance with the smallest p-value (1.92x10-22).

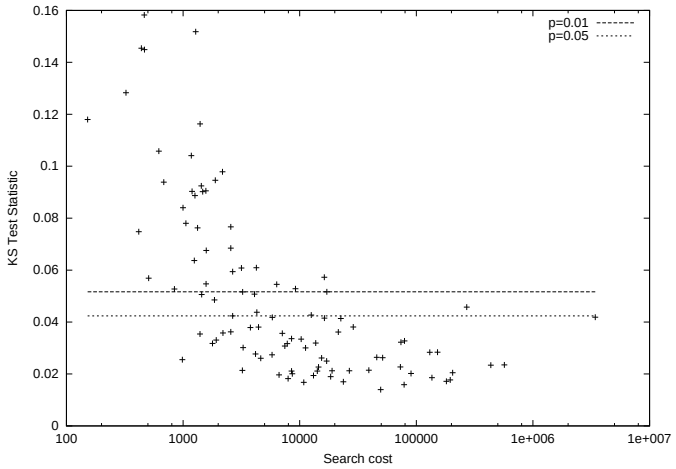

Figure 8.9: Scatter-plot of search cost versus the value of the Kolmogorov-Smirnov test statistic for comparing the actual search cost distribution with that of an exponential distribution. Large values of the test statistic indicate more significant differences. The horizontal lines indicate null hypothesis rejection thresholds at significant $p = 0.01$ and $p = 0.05$.

do exist, they can be minimized by simply shifting one of the two distributions by a constant offset, indicating that any discrepancies are likely due to deficiencies in our estimation process, and not to structural flaws in our dynamic cost model.

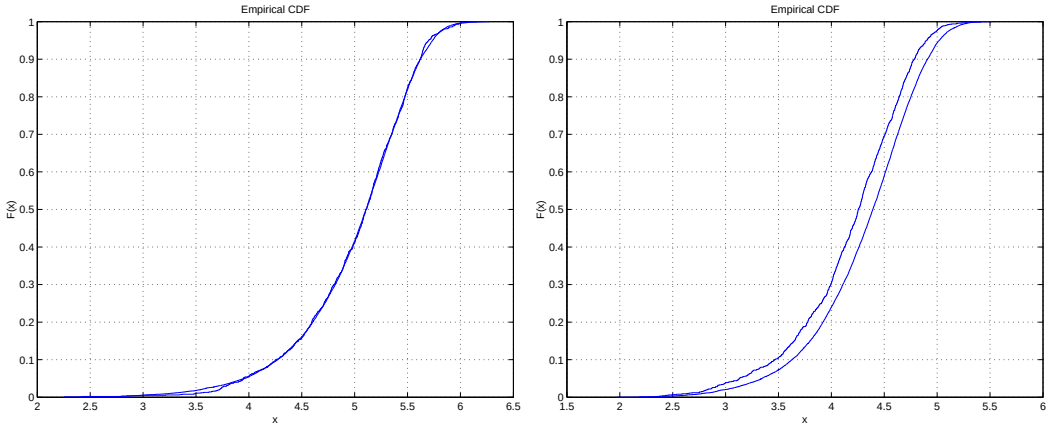

Figure 8.10: CDFs of the predicted and actual RLDs for two 10×10 instances. The p-values for the KS test statistic are respectively 0.861 and 2.3649×10^{-12}.

8.6 An Analysis of the Scalability of *I-JAR* Performance

The experiments reported in Section 8.3.3 indicate that the performance of *I-JAR* is competitive with that of $TS_{Taillard}$ on Taillard's 15×15 benchmark instances. We view these results as a demonstration of the *potential* effectiveness of iterated local search for the JSP. Key open questions relate to scalability and extensibility: Do similar results hold for larger, more difficult square problem instances, or for a range of rectangular problem instances? We address these questions by comparing the performance of iterated local search and tabu search on the full range of Taillard's benchmark instances. Because the $N1$ operator yields many moves that are provably non-improving, state-of-the-art tabu search algorithms are based on the significantly more restrictive $N5$ operator. However, the search space induced by the $N5$ operator is disconnected, with certain implications for the design of local search algorithms based on $N5$; these implications are discussed in Section 8.6.1. An extension of *I-JAR* based on $N5$ is introduced in Section 8.6.2, while an $N5$-based analog of $TS_{Taillard}$ is described in Section 8.6.3. The comparative methodology is discussed in Section 8.6.4, and the relative performance of the two algorithms is reported in Section 8.6.5.

8.6.1 The Implications of $N5$ for Local Search Algorithm Design

While it is known that the $N5$ operator induces a disconnected search space, such that there does not always exist a path from an arbitrary solution to an optimal solution, the implications of this fact for the design of local search algorithms based on $N5$ have not been fully explored. In experimentation with preliminary versions of the algorithms described in Section 8.6.2 and 8.6.3, we observed that search periodically encountered situations in which a small number (≤ 10) of solutions were repeatedly visited, with the algorithms unable to break the cycle. Upon closer examination, this behavior is due to

the existence of small, fully connected sub-components that are isolated from the rest of the fitness landscape. For example, consider a solution s with a single critical block of size ≥ 2 on machine 1, such that $|N5(s)| = 1$; the sole neighbor s' is constructed by inverting the order of the last pair of operations (o_i, o_j) on the critical block. But s' also contains a single critical block on machine 1, with o_j and o_i as the last two operations in the block. Further, $C_{max}(s) = C_{max}(s')$. Clearly, once *any* local search algorithm based strictly on $N5$ encounters either s or s', search will forever cycle between the two solutions. We refer to these sub-components of the fitness landscape as *traps*.

Although this example may appear contrived, it and related generalizations were frequently observed in Taillard's benchmark instances, particularly in the 30×20 problems. Once entered, it is clearly impossible for local search based strictly on $N5$ to escape a trap; recall that $N5$ induces asymmetric neighborhoods. Because we have found no empirical evidence of large, e.g., > 10 solution, sub-components in random JSPs, the two key requirements of local search algorithms based on $N5$ is that they (1) detect when search is in a trap and (2) initiate some form of escape mechanism.

Nowicki and Smutnicki developed local search algorithms based on the $N5$ move operator [NS96] [NS03]; one of the algorithms was introduced simultaneously with the $N5$ operator. Their algorithms, both based on tabu search, identify traps by checking for cycles in the makespans of recently visited solutions. If a cycle is detected, search from the current solution is terminated and re-started from a previously encountered high-quality solution. The two algorithms also perform re-starts if the makespan of the best-so-far solution is not improved over some number of iterations. Consequently, search will eventually escape a trap, even if a cycle is not detected.

The algorithms described in Sections 8.6.2 and 8.6.3 take a different, less drastic approach to escaping traps. To detect a trap, both algorithms monitor the *mobility* of search over time. Mobility [BT94] is defined as the distance between two solutions visited some k number of iterations apart. Both iterated local search and tabu search generally maintain high search mobility for a wide range of parameter settings. Consequently, if the mobility falls below some relatively small threshold, then search is likely contained in a trap, and an escape sequence can be initiated. Instead of re-starting search from some previously encountered high-quality solution, we perform a small modification to the current solution using the canonical $N1$ operator. Because the $N1$ operator induces a connected landscape, the algorithm is able to remain close to the current solution while simultaneously moving search out of the trap.

8.6.2 *I-JAR*$_{N5}$: Algorithm Definition

Conceptually, traps can be detected by analyzing the makespans of recently visited solutions. In practice, the detection criterion is complex, specifically because of the stochastic components of a local search algorithm. Specifically, an algorithm is not guaranteed to exhibit cycling behavior, but may instead repeatedly visit a small set of solutions in a random order. To avoid traps when *I-JAR* is used in conjunction with the $N5$ move

```
function I-JAR_N5(k, MaxIters, TCI, M_thrs, MWL_nom, MWL_inc, OCI, OWL_nom, OWL_inc)
    s_best = s = nextDescent(s_init, N5)
    i = 1, oem=false, vte=false
    repeat
        gte=false
        if i mod TCI then
            M = D(s, s_-TCI)
            if (vte eq true) and (M > M_thrs) then
                vte = false
            else if (vte eq false) and (M ≤ M_thrs) then
                wl = MWL_nom, gte = true
            else if (vte eq true) and (M ≤ M_thrs) then
                wl = wl + MWL_inc, gte = true
        if (i mod OCI) eq 0 then
            if (C_max(s) eq C_max(s_-OCI)) and (oem eq false) then
                oem=true, OWL = OWL_nom
        if gte eq true then
            s'=genRandMove(s,MWL,N1)
        else if oem eq true then
            s'=genRandMove(s,OWL,N5)
        else
            s'=genEscapeMove(s,k,N5)
        s''=nextDescent(s',N1)
        if oem eq true then
            if C_max(s'') eq C_max(s) then
                OWL = OWL + OWL_inc
            else
                oem=false
        if C_max(s'') < C_max(s_best) then
            s_best = s
        s = s'', i = i + 1
    until i eq MaxIters;
    return s_best;
```

Figure 8.11: Pseudo-code for the *I-JAR*$_{N5}$ iterated local search algorithm.

operator, we use a straightforward approach that detects whether search is restricted to a small sub-component of the fitness landscape. Let X_i denote the solution at iteration i. Every *TCI* (*T*rap *C*heck *I*nterval) iterations, the mobility M (i.e., disjunctive graph distance) between the current solution X_i and the solution at iteration X_{i-TCI} is com-

puted. If the M reaches or falls below some threshold M_{thrs}, then a random walk from the current solution is initiated. However, the walk is with respect to the $N1$ operator, which does not induce landscapes with traps. If the mobility remains low after another *TCI* iterations, the length of the random walk is increased; the procedure is iterated until super-threshold mobility is achieved.

We denote the resulting algorithm by *I-JAR*$_{N5}$. Pseudo-code for *I-JAR*$_{N5}$ is shown in Figure 8.11. The algorithm is a simple derivative of *I-JAR*, with logic added to detect and escape from traps. The function D returns the disjunctive graph between the two input solutions; the notation s_{TCI} indicates the solution *TCI* iterations prior. The variable vte indicates that search should verify escape from a trap after *TCI* additional iterations, while the variable gte indicates whether a random walk should be performed in the current iteration. Finally, the function walk(s, x) performs a random walk of length x from solution s, using the $N1$ move operator.

8.6.3 *TS*$_{NS\text{-}A}$: Algorithm Definition

Mirroring the approach taken in Section 8.3.3, we compare the performance of *I-JAR*$_{N5}$ relative to a tabu search algorithm based on the $N5$ move operator. Two such algorithms have been developed by Nowicki and Smutnicki; both algorithms, known as *TSAB* and $i-$*TSAB*, use a straightforward implementation of tabu search in conjunction with periodic re-intensification around previously encountered high-quality solutions. Given strong qualitative similarities in the dynamics of tabu search and iterated local search algorithms for the JSP (as detailed in Sections 7.9 and 8.4.2), it seems likely that re-intensification can significantly improve the performance of *both* meta-heuristics. Consequently, we control for the impact of re-intensification on performance by comparing *I-JAR*$_{N5}$ with a version of *TSAB* (or equivalently $i-$*TSAB*) lacking re-intensification.

In contrast to *TS*$_{Taillard}$, *TSAB* uses a fixed-duration tabu tenure, set to 8 in both Nowicki and Smutnicki's original and more recent experiments [NS96] [NS03]. As indicated previously, *TSAB* is augmented with code for heuristically detecting cycles. When a cycle is identified, search from the current solution is terminated and the re-intensification process is initiated. By detecting and breaking cycles, *TSAB* rectifies two potential problems: (1) cycling due to the use of a fixed-duration tabu tenure (which can occur even if the move operator induces a connected fitness landscape) and (2) cycling due to the existence of traps in the fitness landscape. The tight link between cycle prevention and re-intensification significantly complicates the development of a version of *TSAB* without re-intensification: if re-intensification is not employed, it is unclear how search should proceed when either type of cycle is detected.

To avoid the first type of cycling, we simply borrow the dynamic tabu tenure scheme present in *TS*$_{Taillard}$; the effectiveness of this approach for tabu search algorithms in general is well-known. To avoid cycling due to traps, we use a similar approach to that defined for *I-JAR*$_{N5}$. Let X_i denote the solution at iteration i. Every *TCI* iterations, the mobility M (i.e., disjunctive graph distance) between the current solution X_i and the

solution at iteration X_{i-TCI} is computed. If the M reaches or falls below some threshold M_{thrs}, then a random walk from the current solution is initiated. However, the walk is with respect to the $N1$ operator, which does not induce landscapes with traps. If the mobility remains low after another *TCI* iterations, the length of the random walk is increased; the procedure is iterated until super-threshold mobility is achieved.

```
function TS_NS-A(MaxIters, TCI, M_thrs, MWL_nom, MWL_inc)
    s = s_init
    s_best = s_init
    clear(M)
    i = 1
    vte = false
    wl = 0
    while i ≤ MaxIters do
        gte = false
        if i mod TCI then
            M = D(s, s_-TCI)
            if (vte eq true) and (M > M_thrs) then
                vte = false
            else if (vte eq false) and (M ≤ M_thrs) then
                wl = MWL_nom
                gte = true
            else if (vte eq true) and (M ≤ M_thrs) then
                wl = wl + MWL_inc
                gte = true
        if gte eq true then
            s=walk(wl,s)
        else
            n_all = N(s)
            n_tabu=tabu(n_all,M)
            s = argmin_{x∈n_all\n_tabu}(C_max(x))
        if C_max(s) < C_max(s_best) then
            s_best = s
        update(M,s)
        i = i + 1
    return s_best
```

Figure 8.12: Pseudo-code for the $TS_{NS\text{-}A}$ tabu search algorithm.

We denote the resulting algorithm by $TS_{NS\text{-}A}$. Pseudo-code for $TS_{NS\text{-}A}$ is shown in Figure 8.12. With the exception of the logic dedicated to escaping traps and the

$N5$ move operator, the algorithm is identical to $TS_{Taillard}$. The function D returns the disjunctive graph distance between the two input solutions; the notation s_{-TCI} indicates the solution *TCI* iterations prior. The variable vte indicates that search should verify escape from a trap after *TCI* additional iterations, while the variable gte indicates whether a random walk should be performed in the current iteration. The function walk(s, x) performs a random walk of length x from solution s, using the $N1$ move operator.

Jain et al. [JRM00] also investigate the performance of a variant of *TSAB* without re-intensification, which continues to search until 2,500 iterations have passed without an improvement in the best-so-far solution. Their algorithm does not check for cycles and maintains a fixed tabu tenure. Consequently, although Jain et al. fail to address the issue, search is likely to exhibit cyclic behavior due to either the fixed tabu tenure or the presence of traps.

8.6.4 Comparative Methodology

In all trials involving $I\text{-}JAR_{N5}$, we let $OCI = 20$, $OWL_{nom} = 5$, and $OWL_{inc} = 1$. The parameter settings related to trap detection and escape mechanisms are as follows: $TCI = 50$, $M_{thrs} = 10$, $MWL_{nom} = 5$, and $MWL_{inc} = 1$. Variants of $I\text{-}JAR_{N5}$ with $k = 1$ and $k = 2$ are analyzed. Empirically, performance is largely insensitive to the choice of MWL_{nom} and MWL_{inc}. In contrast, performance is impacted by the ratio of *TCI* to M_{thrs}; the selected values were arrived at via testing on Taillard's benchmark instances.

When executing TS_{N5}, we let $TCI = 50$, $M_{thrs} = 10$, $MWL_{nom} = 5$, and $MWL_{inc} = 1$. These settings enable $TS_{NS\text{-}A}$ to consistently escape traps, as evidenced by the resulting performance reported below in Section 8.6.4. To maintain consistency with Nowicki and Smutnicki's settings for *TSAB* (who use a fixed tabu tenure of duration 8, which was arrived at via extensive experimentation), we let $L_{min} = 6$, $L_{max} = 10$, and re-sample the tabu tenure from the interval $[L_{min}, L_{max}]$ every 15 iterations.

We compare the relative performance of $I\text{-}JAR_{N5}$ and $TS_{NS\text{-}A}$ on Taillard's [Tai93] 80 random JSP benchmark instances. The instances, which are denoted `ta01` through `ta80` and range in size from 15×15 to 100×20, can be partitioned into two categories. The first category contains instances `ta01` through `ta50`, 10 instances apiece of size 15×15, 20×15, 20×20, 30×15, and 30×30. The optimal makespan is only known for 17 of these 50 instances. Further, even state-of-the-art algorithms have difficulty consistently locating optimal solutions to these closed instances (e.g., see [NS03]). The remaining instances, in particular the 20×20 and 30×20 instances, are widely regarded as the most difficult random JSP benchmarks available. The second category contains instances `ta51` through `ta80`, 10 instances apiece of size 50×15, 50×20, and 100×20. In sharp contrast to those in the first category, the optimal makespans to all of these instances are known. Further, with only one exception (`ta62`), all instances can be consistently solved to optimality by the more effective JSP algorithms, including $I\text{-}JAR_{N5}$ and $TS_{NS\text{-}A}$.

	Problem Size							
	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20
I-JAR$_{N5}(k=1)$	1.48	1.42	1.38	1.62	1.61	1.32	1.47	1.51
TS$_{NS\text{-}A}$	3.67	3.2	3.11	2.95	2.98	2.78	2.26	2.38

Table 8.3: CPU cost-per-iteration multipliers between the baseline *I-JAR*$_{N5}(k=2)$ and both *I-JAR*$_{N5}(k=1)$ and *TS*$_{NS\text{-}A}$ for Taillard's random JSP benchmark instances.

As is the case for state-of-the-art algorithms, nearly all of the instances in the first category elude consistent solution by both *I-JAR*$_{N5}$ and *TS*$_{NS\text{-}A}$. Consequently, because there exists sufficient "headroom" to demonstrate the superiority of one algorithm over the other, we compare performance on these instances by analyzing the makespan statistics (minimum and mean) of the best solution encountered over 10 independent fixed-duration runs. In contrast, due to their relative simplicity, we compare performance on instances in the second category based on the mean number of iterations required to locate an optimal solution; statistics are also taken over 10 runs. For the exceptional `ta62` instance, we revert to analyzing makespan statistics.

Extensive run-time profiling of *I-JAR*$_{N5}$ (under either $k=1$ or $k=2$) and *TS*$_{NS\text{-}A}$ indicates that approximately 99% of the run-time is devoted to computing the makespan of neighboring solutions (as described in Section 3.5.2). Consequently, by limiting trials of each algorithm to a fixed CPU time or an CPU-equivalent number of iterations, a relatively fair comparison can be achieved. Most algorithmic or implementation enhancements to the routines for computing neighboring solution makespans, such as those described in [NS02] or [NS03], benefit all local search algorithms for the JSP, albeit not necessarily to the same extent. Of course, it is always possible to perform algorithm-specific "tweaks", such that it is very difficult to demonstrate the inherent superiority of one meta-heuristic over another. To minimize such criticism, we use straightforward implementations of the meta-heuristic and move operator components of both algorithms. In particular, we resist the temptation to perform enhancements to *I-JAR*$_{N5}$, placing it at a disadvantage; as indicated in Section 8.3.3, caching can be used to improve performance – significantly more so than in tabu search.

All trials and supporting experiments are executed on 1.5 GHz Pentium IV hardware with 512 Mb of memory, running the Linux 2.4.18-5 operating system; all code was compiled using gcc 3.2 with level 3 optimization. Our baseline algorithm is *I-JAR*$_{N5}$ with $k=2$, which we run for 10 million iterations. Via extensive empirical tests on Taillard's instances, we have computed estimates of the cost-per-iteration multiplier for both *I-JAR*$_{N5}(k=1)$ and *TS*$_{NS\text{-}A}$ relative to *I-JAR*$_{N5}(k=2)$. The multipliers, shown in Table 8.3, are used to determine the number of iterations allocated to each trial of *I-JAR*$_{N5}(k=1)$ and *TS*$_{NS\text{-}A}$. For example, the results in Table 8.3 indicate that for Taillard's 20×20 instances, 3.11 iterations *TS*$_{NS\text{-}A}$ can be completed, on average, in the time it takes to complete a single iteration of *I-JAR*$_{N5}(k=2)$. To yield accurate comparison, 31.1 million iterations of each trial of *TS*$_{NS\text{-}A}$ are performed for the 20×20 instances. Although we have not analyzed the underlying cause for multiplier variabil-

ity on different-sized problem instances in any detail, the results are consistent with intuition. Considering *I-JAR*$_{N5}$, the ascent phase is necessarily cheaper under $k = 1$ than $k = 2$. Similarly, *I-JAR*$_{N5}$ is performing significantly more work per iteration (on average several moves are accepted) than *TS*$_{NS\text{-}A}$.

By using multipliers we are trying to allocate equivalent amounts of CPU time to each trial. By indirectly achieving time-limited trials, the goal is to enable replicability, as the number of iterations is a platform and implementation-independent measure. With the exception of [NS02] and [NS03], the number of trials (and equivalently, the CPU time) is large relative to experimental results previously reported in the literature. In part, this is possible due to advances in computing power. However, we allocate a large number of iterations primarily in order to assess the long-run potential of each algorithm. Finally, we note that our implementations are *not* nearly as efficient as those reported by Nowicki and Smutnicki. For example, using the data reported by Nowicki and Smutnicki [NS02] [NS03], their $i-$*TSAB* algorithm executes 31.1 million iterations on Taillard's 20×20 instances in roughly 787 seconds on a 900MHz Pentium III. In contrast, our implementation of *TS*$_{NS\text{-}A}$ (which uses the same move operator as $i-$*TSAB*) requires roughly 9,134 seconds on a 1.5GHz Pentium IV. The differences are due to a number of implementation and algorithmic enhancements in the computation of neighboring solution makespans. Our primary goal is analysis, and not to develop high-performance implementations of local search algorithms for the JSP. That said, the same enhancements can also be used to significantly improve the performance of our algorithms.

8.6.5 Quantifying the Scalability of *I-JAR*

We first consider the results for Taillard's 15×15 and 20×15 instances, shown in Table 8.4. The optimal makespan is known for all of the 15×15 instances, but for only two of the 20×15 instances. Collectively, these are the easiest of Taillard's benchmark instances. However, as previously noted, even state-of-the-art algorithms fail to consistently locate optimal or best-known solutions to these instances. We first compare the results of *I-JAR*$_{N5}$ under $k = 1$ and $k = 2$. In absolute terms, the $k = 1$ variant provides better mean solution quality, although performance is statistically indistinguishable under either paired two-sample t-tests or non-parametric Wilcoxon signed-rank tests. Further, no one variant consistently yields the overall best solution found in their respective 10 trials. Similarly, mirroring the results presented Section 8.3.3, the performance of either variant was indistinguishable from that of *TS*$_{NS\text{-}A}$, although *I-JAR*$_{N5}(k = 1)$ obtained lower absolute mean solution quality. All three algorithms locate optimal solutions to at least six of the ten 15×15 instances, and solutions within 20 time units of the best-known upper bound for the 20×15 instances.

Next, we analyze the results for Taillard's 20×20, 30×15, and 30×20 instances, shown in Table 8.5. These instances are among the most difficult random JSP benchmarks in existence. For example, few algorithms can consistently locate solutions within

		I-JAR$_{N5}(k=1)$			*I-JAR*$_{N5}(k=2)$			*TS*$_{NS\text{-}A}$		
Instance	Opt.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
ta01	1231	***1231***	1238.2	1242	***1231***	1237.1	1243	***1231***	**1231**	1231
ta02	1244	***1244***	**1244**	1244	***1244***	1244.1	1245	***1244***	1244.4	1246
ta03	1218	*1219*	**1220.8**	1222	*1219*	1221	1222	*1219*	1221.8	1223
ta04	1175	***1175***	**1175**	1175	***1175***	**1175**	1175	***1175***	**1175**	1175
ta05	1224	1229	1230.9	1233	1233	1233	1233	*1228*	**1229.8**	1232
ta06	1238	***1238***	**1238.8**	1240	***1238***	1241.7	1244	***1238***	1240.7	1244
ta07	1227	*1228*	**1228**	1228	*1228*	**1228**	1228	*1228*	**1228**	1228
ta08	1217	***1217***	**1217**	1217	***1217***	1217.3	1219	***1217***	**1217**	1217
ta09	1274	1281	1282.2	1283	1281	1282.6	1286	*1279*	**1281.8**	1283
ta10	1241	***1241***	**1242.9**	1244	***1241***	1243.4	1244	***1241***	1244.3	1247

		I-JAR$_{N5}(k=1)$			*I-JAR*$_{N5}(k=2)$			*TS*$_{NS\text{-}A}$		
Instance	Bounds/ Opt.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
ta11	1323-1361	1373	1377.5	1380	1375	1378	1383	*1368*	**1374.4**	1379
ta12	1351-1367	*1377*	1379.1	1382	*1377*	1380.6	1383	*1377*	**1377.1**	1378
ta13	1282-1342	1352	**1354.7**	1356	1355	1356	1357	*1350*	1355.4	1359
ta14	1345	***1345***	**1345**	1345	***1345***	1345.1	1346	***1345***	**1345**	1345
ta15	1304-1340	*1346*	**1355.2**	1361	1356	1359.7	1361	1353	1356	1358
ta16	1302-1360	*1362*	**1367.1**	1370	1368	1371.6	1375	1368	1371.4	1376
ta17	1462	1474	1476.2	1479	1473	1477.4	1481	*1469*	**1475**	1479
ta18	1369-1396	*1409*	**1411.9**	1415	1413	1416.4	1422	1414	1416.4	1420
ta19	1297-1335	*1341*	**1344.9**	1348	1346	1348.7	1352	1343	1346.5	1351
ta20	1318-1351	1358	**1361.8**	1366	1361	1365.2	1369	*1357*	**1361.8**	1364

Table 8.4: Statistics for the makespans of the best solutions obtained by *I-JAR*$_{N5}$ and *TS*$_{NS\text{-}A}$ to Taillard's small (15×15 – upper portion, 20×15 – lower portion) benchmark instances. Statistics are taken over 10 independent trials. The second column indicates either the optimal makespan, or lower and upper bounds on the optimal makespan. Bold-faced entries in the 'Min' columns indicate equality with the optimal makespan. Italicized entries in the 'Min' columns indicate the best makespan achieved by any algorithm. Bold-faced entries in a 'Mean' column indicates the mean makespan was less than or equal to that of any competing algorithm.

		I-JAR$_{N5}(k=1)$			*I-JAR*$_{N5}(k=2)$			*TS*$_{NS\text{-}A}$		
Instance	Bounds	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
ta21	1539-1644	1658	1661.1	1664	1654	1663.7	1668	*1648*	**1654.6**	1658
ta22	1511-1600	*1602*	**1612.1**	1619	1613	1617.7	1621	1613	1620.3	1624
ta23	1472-1557	*1558*	1568.9	1575	1568	1574.9	1579	1563	**1565.9**	1568
ta24	1602-1647	*1654*	**1654.2**	1656	*1654*	1655.6	1659	*1654*	1656.2	1659
ta25	1504-1595	*1599*	1603.6	1609	1603	1608	1613	*1599*	**1601.2**	1611
ta26	1539-1645	*1655*	1661.6	1666	1659	1662.6	1668	1655	**1661**	1664
ta27	1616-1680	1697	1703	1709	1697	1706.1	1710	*1691*	**1694.8**	1698
ta28	1591-1614	*1617*	**1620.6**	1625	1620	1625.2	1630	1618	1621.4	1623
ta29	1514-1625	1629	1631.1	1634	*1629*	1633.2	1637	*1628*	**1629.2**	1632
ta30	1473-1584	1592	1600.8	1607	1593	1604.3	1610	*1590*	**1591.7**	1596

		I-JAR$_{N5}(k=1)$			*I-JAR*$_{N5}(k=2)$			*TS*$_{NS\text{-}A}$		
Instance	Bounds / Opt.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
ta31	1764	***1764***	**1764**	1764	***1764***	1765.3	1766	***1764***	1764.3	1766
ta32	1774-1796	*1811*	1819.7	1825	1821	1827	1830	*1811*	**1816.8**	1824
ta33	1778-1793	1809	1812.5	1817	1816	1818.9	1822	*1806*	**1811.6**	1814
ta34	1828-1829	*1833*	1834.1	1836	1835	1835.9	1837	*1833*	**1833.7**	1834
ta35	2007	***2007***	**2007**	2007	***2007***	**2007**	2007	***2007***	**2007**	2007
ta36	1819	***1819***	1822	1825	1821	1825	1828	***1819***	**1819.6**	1823
ta37	1771-1778	*1793*	**1794.2**	1796	1795	1797.1	1800	*1793*	1794.6	1796
ta38	1673	1683	1684.9	1687	1684	1690.3	1693	*1675*	**1681.5**	1684
ta39	1795	1797	1798.7	1802	1797	1799.3	1801	***1795***	**1797.7**	1799
ta40	1631-1674	1695	1696.2	1698	1698	1701.5	1704	*1691*	**1692.5**	1697

		I-JAR$_{N5}(k=1)$			*I-JAR*$_{N5}(k=2)$			*TS*$_{NS\text{-}A}$		
Instance	Bounds	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
ta41	1859-2018	2042	2045.5	2050	2047	2052.9	2057	*2034*	**2039.1**	2044
ta42	1867-1956	1974	1976	1979	1974	1980.8	1989	*1966*	**1968.6**	1972
ta43	1809-1859	1889	1890.3	1896	1889	1895.5	1899	*1876*	**1882**	1889
ta44	1927-1984	2006	2009.2	2011	2009	2013.5	2016	*2002*	**2005.7**	2009
ta45	1997-2000	2007	2012.4	2014	2009	2014.6	2016	*2005*	**2009.4**	2013
ta46	1940-2021	*2035*	**2038.9**	2044	2038	2045.7	2050	*2035*	2039.3	2042
ta47	1789-1903	*1925*	1931.3	1937	1933	1942.4	1950	1926	**1929.6**	1936
ta48	1912-1952	1975	1980.2	1982	1983	1986.6	1992	*1968*	**1971.9**	1975
ta49	1915-1968	1991	1992.8	1996	1994	2001.1	2005	*1980*	**1985.1**	1989
ta50	1807-1926	1941	**1943.9**	1950	1948	1953.2	1956	*1940*	1945.6	1951

Table 8.5: Statistics for the makespans of the best solutions obtained by *I-JAR*$_{N5}$ and *TS*$_{NS\text{-}A}$ to Taillard's medium-sized (20×20 – upper portion, 30×15 – middle portion, and 30×20 – lower portion) benchmark instances. Statistics are taken over 10 independent trials. The second column indicates either the optimal makespan, or lower and upper bounds on the optimal makespan. Bold-faced entries in the 'Min' columns indicate equality with the optimal makespan. Italicized entries in the 'Min' columns indicate the best makespan achieved by any algorithm. Bold-faced entries in a 'Mean' column indicates the mean makespan was less than or equal to that of any competing algorithm.

Instance	*I-JAR*$_{N5}(k=1)$ Mean Evals	*I-JAR*$_{N5}(k=2)$ Mean Evals	*TS*$_{NS\text{-}A}$ Mean Evals	*I-JAR*$_{N5}(k=1)$ Multiplier	*TS*$_{NS\text{-}A}$ Multiplier
ta51	15002	5498	38070	2.73	6.92
ta52	16552	6090	15061	2.72	**2.47**
ta53	9919	3659	9538	2.71	**2.61**
ta54	83702	5593	155321	14.97	27.77
ta55	26502	11570	37611	2.29	3.25
ta56	19678	8494	29906	2.32	3.52
ta57	7840	3545	10461	2.21	2.95
ta58	10576	3280	8456	3.22	**2.58**
ta59	26050	9415	28486	2.77	3.03
ta60	12925	5623	11948	2.36	**2.12**
Instance	*I-JAR*$_{N5}(k=1)$ Mean Evals	*I-JAR*$_{N5}(k=2)$ Mean Evals	*TS*$_{NS\text{-}A}$ Mean Evals	*I-JAR*$_{N5}(k=1)$ Multiplier	*TS*$_{NS\text{-}A}$ Multiplier
ta61	153302	47946	102127	3.20	**2.13**
ta62	N/A	N/A	N/A	N/A	N/A
ta63	29135	14633	39235	1.99	2.68
ta64	114027	23131	185517	4.93	8.02
ta65	92800	34188	135599	2.71	3.97
ta66	48036	20179	96025	2.38	4.76
ta67	180378	133451	409617	**1.35**	3.07
ta68	30742	13637	46043	2.25	3.38
ta69	13911	7080	23093	1.96	3.26
ta70	185266	105528	321459	1.76	3.05
Instance	*I-JAR*$_{N5}(k=1)$ Mean Evals	*I-JAR*$_{N5}(k=2)$ Mean Evals	*TS*$_{NS\text{-}A}$ Mean Evals	*I-JAR*$_{N5}(k=1)$ Multiplier	*TS*$_{NS\text{-}A}$ Multiplier
ta71	17030	6799	20871	2.50	3.07
ta72	10184	4348	23421	2.34	5.39
ta73	33060	8181	64574	4.04	7.90
ta74	14761	4197	9539	3.52	**2.27**
ta75	22400	8743	27376	2.56	3.13
ta76	11314	4691	12836	2.41	2.74
ta77	5546	2181	7888	2.54	3.62
ta78	11007	4608	11768	2.39	2.55
ta79	11817	3554	15908	3.32	4.48
ta80	18408	7827	25133	2.35	3.21

Table 8.6: Statistics for the mean number of evaluations required by *I-JAR*$_{N5}$ and *TS*$_{NS\text{-}A}$ to locate optimal solutions to Taillard's large benchmark instances. Statistics are taken over 10 independent trials. The baseline algorithm is *I-JAR*$_{N5}$ with $k = 2$. The final two columns respectively indicate the ratio of the mean number of evaluations required by the respective algorithm to the mean number of evaluations required by *I-JAR*$_{N5}(k = 2)$. Bold-faced entries indicate the ratio is lower than the corresponding CPU time-per-iteration ratio (as shown in Table 8.3), i.e., the respective algorithm outperforms *I-JAR*$_{N5}(k = 2)$ on that particular instance.

40 time units of the best-known makespans to the 30×20 instances; Nowicki and Smutnicki's $i - TSAB$ algorithm is one exception [NS03]. The optimal makespan is known for only 5 of the 30×15 instances. As with Taillard's smaller instances, we observed no statistically significant difference in the mean solution quality obtained by the $k = 1$ and $k = 2$ variants of *I-JAR*$_{N5}$. However, in no case does the $k = 2$ variant yield lower mean solution quality than the $k = 1$ variant, or a lower minimum solution quality. This result is consistent with the observed decreases in attractor basin strength as problem size is increased. Similarly, the difference in mean solution quality obtained under *I-JAR*$_{N5}(k = 1)$ and *TS*$_{NS\text{-}A}$ is statistically insignificant. However, the results for the 30×20 instances provide some (although by no means conclusive) evidence that *TS*$_{NS\text{-}A}$ outperforms *I-JAR*$_{N5}(k = 1)$ by a slim margin: for all but one of the instances (`ta50`), *TS*$_{NS\text{-}A}$ locates the best solution obtained by any of the three algorithms. Discounting the 30×15 instances for which the optimal makespan is known, none of the three algorithms locates a solution with a makespan equivalent to the best-known upper bound.

We now examine the results for Taillard's large rectangular 50×15, 50×20,and 100×20 instances, shown in Table 8.6. For all but one of the instances (`ta62`), all three algorithms consistently located the optimal solution within the allocated number of iterations. This result is consistent with the observation that random JSPs become easier as m/n approaches ∞. Despite its relative underperformance on Taillard's smaller instances, the $k = 2$ variant of *I-JAR*$_{N5}$ typically requires statistically *fewer* evaluations to locate optimal solutions than the $k = 1$ variant, as confirmed by a paired-sample Wilcoxon test. Further, the ratio of $k = 1$ to $k = 2$ evaluations exceeds the corresponding CPU cost-per-iteration (shown in Table 8.3) in all but one case. We would expect the converse to hold, given that (1) the $k = 1$ variant either outperforms or equals the $k = 2$ variant on Taillard's smaller benchmarks and (2) attractor basin strength decays with increases in problem size. We currently have no explanation for this apparent contradiction. The $k = 2$ variant of *I-JAR*$_{N5}$ always requires statistically fewer iterations than *TS*$_{NS\text{-}A}$. When adjusting for the cost-per-iteration multiplier, *I-JAR*$_{N5}$ outperforms *TS*$_{NS\text{-}A}$ in an absolute sense on 24 of the 30 instances.

The results presented in Tables 8.4 through 8.6 clearly indicate that the relative performance of *I-JAR* with respect to tabu search scales with increases in problem size. The surprisingly competitive performance of *I-JAR*$_{N5}$ under $k = 1$ also reinforces the hypothesis that for moderate to large random JSPs, a *single* dis-improving move is sufficient to escape the attractor basins of most local optima. Strong performance also provides evidence for the success of the walk mechanisms for escaping both very strong local optima (see Section 8.3) and traps in the fitness landscape (see Section 8.6.2), i.e., there is no evidence that search in *I-JAR*$_{N5}$ ever becomes trapped in a restricted region of the fitness landscape.

Finally, we compare the aggregate performance of both *I-JAR*$_{N5}$ and *TS*$_{NS\text{-}A}$ with that of other local search algorithms for the JSP. To do so, we compare the mean relative error *MRE* on sets of Taillard's instances of equal size (e.g., `ta01-ta10`). Let s denote

Group	$I\text{-}JAR^b_{N5}$	$I\text{-}JAR^m_{N5}$	$TS^b_{NS\text{-}A}$	$TS^m_{NS\text{-}A}$	$i-TSAB$ (20M)	$i-TSAB$ (50M)	*PEZ*	*BAL*	BAL^b
ta01-10	0.11	0.22	0.08	0.26	0.11	0.11	0.45	0.25	0.16
ta11-20	2.92	3.20	3.13	3.40	2.81	2.81	3.47	3.34	2.81
ta21-30	6.03	6.27	5.93	6.27	5.69	5.68	6.52	6.57	6.10
ta31-40	1.02	1.28	1.04	1.28	0.85	0.78	1.92	1.13	0.80
ta41-50	5.18	5.37	4.97	5.24	4.97	4.7	6.04	5.71	5.20

Table 8.7: Mean relative error (MRE) of various algorithms on Taillard's difficult benchmark instances. See text for details.

a solution obtained by an algorithm on a given problem instance. Define the relative error *RE* of s as $\frac{C_{max}(s)-C^*_{max}}{C^*_{max}}$, where C^*_{max} is the optimal makespan of the problem instance or the largest known lower bound. The *MRE* is simply the average *RE* taken over a set of 10 problem instances.

Based on a comparative analysis of published performance results, Nowicki and Smutnicki (2003)[NS03] indicate that the best available approximation algorithms for the JSP are as follows: (1) $i-TSAB$, (2) a tabu search algorithm based on the shifting bottleneck heuristic [PM00], which they denote *PEZ*, and (3) a guided local search algorithm based on the shifting bottleneck heuristic [BV98], which they denote *BAL*. Ignoring CPU time, these algorithms yield lower *MRE*s on Taillard's benchmark suite than all competitors.

The *MRE* for the 5 sets of Taillard's difficult benchmark instances are shown in Table 8.7. The columns labeled $I\text{-}JAR^b_{N5}$ and $I\text{-}JAR^m_{N5}$ respectively report the *MRE* for the best and mean solution quality obtained over 10 trials; analogous notations are used for $TS_{NS\text{-}A}$. The columns labeled $i-TSAB(20M)$ and $i-TSAB(50M)$ contain results for single runs of $i-TSAB$ limited to 20 and 50 million iterations, respectively, as reported in [NS03]. The *PEZ* and BAL results are from a single trial of each algorithm (taken from [PM00] and [NS03], respectively), while the BAL^b results are the best solution obtained by numerous trials over a variety of parameter settings (also taken from [NS03]. Twenty million iterations of $i-TSAB$ is sufficient to beat all known competitors; 50 million iterations yields even stronger relative performance.

Both $I\text{-}JAR_{N5}$ and $TS_{NS\text{-}A}$ yield solutions that are competitive with those generated by *BAL*, and better than those generated by *PEZ*. This is surprising, given the fact that both *BAL* and *PEZ* use significant amounts of problem-specific knowledge, above and beyond that embodied in the $N5$ operator. Further, due in part to the extensive use of problem-specific knowledge, the complexity of both *BAL* and *PEZ* is significantly higher than that of $I\text{-}JAR_{N5}$ and $TS_{NS\text{-}A}$. Analogous results hold when we consider the best solutions obtained over numerous independent trials, e.g., compare the columns labeled $I\text{-}JAR^b_{N5}$, $TS^b_{NS\text{-}A}$, and BAL^b.

Chapter 9

Metropolis Sampling and Simulated Annealing

Tabu search and iterated local search are both predated by simulated annealing, a local search meta-heuristic independently introduced by Kirkpatrick et al. in 1983 [KGV83] and Cěrny in 1985 [Cě85]. Soon after its introduction, researchers proved that under certain theoretical (albeit practically unattainable) conditions, simulated annealing would converge to a globally optimal solution. The existence of a convergence proof led to significant interest in simulated annealing, which was applied to a wide range of NP-hard problems between 1985 and the early 1990s. The general conclusion of this research was that simulated annealing could indeed locate high-quality and even optimal solutions, but at the expense of very large run-times – which are required to approximate the theoretical conditions for convergence. The introduction of faster, more effective meta-heuristics in the late 1980s (tabu search and genetic algorithms in particular), caused a significant decline in the level of research devoted to simulated annealing, such that new publications involving the application or analysis of simulated annealing are now relatively rare.

Although generally regarded as ineffective relative to more recent meta-heuristics, the reasons for the inferior performance are less clear. Further complicating the situation is the fact that simulated annealing is closely related to more modern meta-heuristics – often to the point that one would not expect significant differences in performance. For example, *I-JAR* is equivalent to a certain parameterization of simulated annealing, and the former is capable of locating high-quality solutions to difficult problems.

This chapter is largely devoted to identifying and understanding some of the causal factors contributing to the inferior performance of simulated annealing algorithms for the JSP. We begin in Section 9.1 with a general overview of simulated annealing, and survey prior research specific to the JSP in Section 9.2. The algorithms and methodologies used in our analysis are discussed in Section 9.3. In Sections 9.4 and 9.5, we investigate the impact of local optima on the behavior of simulated annealing. One unexplored question related to simulated annealing is whether fixed-temperature Metropolis sam-

pling can achieve competitive performance with simulated annealing. Experimental results presented in Section 9.6 indicate that in fact annealing is *not* necessary; Metropolis sampling alone can yield performance that is competitive with both *I-JAR* and $TS_{Taillard}$. A causal explanation for the success of Metropolis sampling is introduced in Section 9.4. A dynamic cost model of Metropolis sampling is developed in Section 9.7, which indicates that the underlying search dynamics are not significantly different from either *I-JAR* or $TS_{Taillard}$. The dynamic cost model can account for some of the full run-length distribution, as analyzed in Section 9.8. We conclude in Section 9.6 with an analysis of the scalability of Metropolis sampling to the most difficult random JSPs, solved using a variant based on the $N5$ move operator.

9.1 An Overview of Metropolis Sampling and Simulated Annealing

In iterated local search, and to a slightly lesser extent tabu search, the descent and escape phases are distinct: greedy descent guides search toward local optima, while an escape mechanism enables search to accept short sequences of dis-improving moves. However, such a clean separation is not characteristic of all meta-heuristics for local search. In the Markov-Chain Monte-Carlo (MCMC) meta-heuristic, dis-improving moves can be accepted at any point in search, such that there are no identifiable descent or escape phases. MCMC is also known as the Metropolis algorithm [MRR+53], which was originally introduced to simulate complex systems in statistical mechanics (a specialized field of physics). The terms MCMC, Metropolis algorithm, and Metropolis sampling are used interchangeably. MCMC is a variant of next-descent search in which a neighbor $s' \in N(s)$ of the current solution s is accepted with probability

$$\begin{array}{cc} 1 & \textbf{\textit{if}} \quad F(s') - F(s) \leq 0 \\ e^{-(F(s')-F(s))/T} & \textbf{\textit{if}} \quad F(s') - F(s) > 0 \end{array} \tag{9.1}$$

where T is a user-specified parameter known as the *temperature*. At each iteration of MCMC, a neighbor $s' \in N(s)$ of the current solution s is selected at random, and Equation 9.1 is applied to determine whether s' is accepted as the current solution in the next iteration. When $T = 0$, MCMC is identical to next-descent; when $T = \infty$, MCMC is identical to a random walk. At intermediate temperatures, the probability of accepting dis-improving neighbors is inversely proportional to T.

One drawback to MCMC is that performance is clearly sensitive to the specific value of T. If T is sufficiently large, MCMC may fail to descend deep enough into attractor basins to detect the associated local optima. Similarly, low values of T can prevent MCMC from escaping attractor basins with any significant non-zero probability. The simulated annealing meta-heuristic [KGV83] [Cˇ85] corrects this deficiency. Simulated annealing can be viewed as an iterative series of MCMC searches, each performed at a different temperature. In MCMC, the probability of escaping an attractor basin is

```
function Simulated Annealing
    s = s_init
    s_best = s_init
    T=initTemp()
    repeat
        numIters=itersAtTemp(T)
        (s_iterbest, s)=MCMC(s,T,numIters)
        if F(s_iterbest) < F(s_best) then
            s_best = s_iterbest
        T=nextTemp(T)
    until T < finalTemp()
    return s_best
```

Figure 9.1: Pseudo-code for the simulated annealing meta-heuristic. See text for details.

inversely proportional to T, and as $T \rightarrow 0$, the escape probability approaches 0. In simulated annealing, T is initially set very high, such that there is little bias toward improving solutions. As time progresses, the temperature is gradually lowered toward 0, eventually driving search toward local, and ideally global, optima. The intent of the initial exploration is to identify regions of the search space with high-quality solutions. Subsequent lower-temperature exploration then searches this region for good local, and ideally global, optima.

Pseudo-code for the simulated annealing meta-heuristic is shown in Figure 9.1. Simulated annealing operates using a single move operator N and proceeds via iterative modifications to some initial solution s_{init}. The current solution is denoted by s. Together, the functions initTemp, nextTemp, and finalTemp define the *cooling schedule*, i.e., the sequence of temperatures through which search progresses. The function MCMC executes Metropolis sampling from the input solution at the specified temperature for a given number of iterations, returning both the best solution encountered and the terminating solution. The function itersAtTemp specifies how long MCMC is to be executed at each temperature. As with the meta-heuristics considered in previous chapters, simulated annealing tracks and returns the best solution located during search, i.e., any invocation of MCMC.

In practice, simulated annealing generally proceeds from random initial solutions, although heuristically constructed solutions do appear to improve performance in some circumstances, e.g., see [JAMS89]. Details of the cooling schedules can vary significantly between implementations, although simple geometric cooling schedules are most prevalent. In a geometric schedule, T is initialized to a relatively high value and is decreased by a constant factor until the value falls below some threshold T_{final}. Cooling schedules typically define a monotonically non-increasing sequence of temperatures, al-

though adaptive cooling schedules have been analyzed by a number of researchers, e.g., see [YRN94]. The number of iterations of MCMC executed at each temperature is generally defined as a multiple of the maximal neighborhood size $|N|$; however, prior to the 1990s, the number of iterations executed at each temperature was frequently variable.

Simulated annealing is widely regarded as the first significant meta-heuristic for local search; prior to its introduction in 1983, the dominant meta-heuristic was iterated descent. Subsequently, researchers introduced implementations of simulated annealing for numerous well-known NP-hard optimization problems. Interest in simulated annealing was largely due to the following result: under certain theoretical conditions, simulated annealing is guaranteed to converge to a global optimum from an arbitrary initial solution [AK89] [vLA88]. The conditions for convergence, which include an infinitely slow cooling schedule and an infinite number of iterations at each temperature, are unattainable in practice. With large enough run-times, such that the theoretical conditions for convergence are approximated, simulated annealing is able to produce high-quality solutions to a number of difficult problems. However, other meta-heuristics such as tabu search and iterated local search generally produce equally good solutions with substantially shorter run-times. For further details regarding the empirical behavior of simulated annealing on a number of well-known NP-hard problems, we defer to a series of excellent landmark studies performed by Johnson et al. [JAMS89, JAMS91]. Overviews of the theoretical foundations of simulated annealing are provided by Aarts and Korst [AK89], van Laarhoven and Aarts [vLA88], and Salamon et al. [SSF02].

9.2 Simulated Annealing and the JSP

Although it is the last meta-heuristic analyzed in this thesis[1], algorithms based on simulated annealing represent the first successful application of local search to the JSP. van Laarhoven et al. introduced the first simulated annealing algorithm for the JSP in 1988 [vLAL88], later refined in [vLAL92]; an algorithm based on a variant of simulated annealing called controlled-temperature simulated annealing, in which all dis-improving moves are accepted with equal probability, was introduced in 1988 by Matsuo et al. [MSS88]. van Laarhoven et al's algorithm is based on the $N1$ move operator, and executes MCMC for $|N1|$ iterations at each temperature, where $|N1|$ is the maximal neighborhood size under the $N1$ operator. The initial temperature is determined via a reverse annealing procedure: starting from a small non-zero positive initial value, the temperature is increased until the ratio of accepted-to-generated moves exceeds some pre-specified threshold (e.g., 0.95). The exact details of the cooling schedule, including

[1]This decision was based on numerous reports in the literature that the performance of simulated annealing was inferior to that of tabu search and iterated local search.

the rules for decrement and termination, are documented in [AvL85].

In the context of local search and the JSP, van Laarhoven et al's algorithm is noteworthy for several reasons. First, the algorithm represents the first use of the $N1$ operator in conjunction with local search algorithms for the JSP. Consequently, the 1992 article documenting both the algorithm and the $N1$ operator [vLAL92] is among the most widely cited in the JSP literature. Second, van Laarhoven et al. demonstrated that their algorithm could outperform iterated greedy descent (given equivalent CPU times), which indicated that more complex meta-heuristics could yield effective local search algorithms for the JS. This result is non-trivial, as simulated annealing does *not* always outperform simple iterated greedy descent, e.g., see [JAMS91]. Third, van Laarhoven et al's empirical results indicate that simulated annealing can be at least as effective as more tailored heuristics for the JSP (specifically the shifting bottleneck procedure [ABZ88] and Matsuo et al's [MSS88] controlled-search simulated annealing algorithm). However, the increased effectiveness does come at the expense of larger run-times, which van Laarhoven et al. justify as follows: "We consider the disadvantage of large running times to be compensated for by the simplicity of the algorithm, by its ease of implementation, by the fact that it requires no deep insight into the combinatorial structure of the problem, and, of course, by the high quality of the solutions it returns" ([vLAL92], p. 124).

Since the introduction of van Laarhoven et al's algorithm, a number of researchers have introduced implementations of simulated annealing for the JSP. The most noteworthy algorithms are described below; see the survey of Jain and Meeran [JM99] for a more comprehensive survey. Yamada et al. [YRN94] describe a simulated annealing algorithm based on a powerful critical-block-based move operator. Borrowing from Nowicki and Smutnicki's [NS96] TSAB tabu search algorithm for the JSP, Yamada et al. also reintensify search when one of the following two conditions is met: (1) the ratio of uphill-to-generated moves over the last X iterations falls below some threshold, indicating that search is trapped in a local optimum, or (2) Y iterations have passed without locating a solution that improves upon the best-so-far makespan. When either condition is satisfied, search is re-initiated from the best-so-far solution. Yamada et al. provide empirical evidence that their algorithm outperforms van Laarhoven et al's original algorithm.

Yamada and Nakano [YN95] later extended Yamada et al's simulated annealing algorithm by introducing a new critical path move operator that hybridized the $N1$ operator with the shifting bottleneck procedure. Additionally, once a move to each neighbor has been attempted (but not accepted), their algorithm accepts a single move with a probability proportional to the corresponding weight under the Metropolis criterion (i.e., Equation 9.1). This enhancement, similar to a mechanism found in rejectionless Markov-Chain Monte-Carlo algorithms [LB00], reduces the chance that search will become trapped in a local optimum for extended periods of time. Most recently, Kolonko [Kol99] proved that because the $N1$ move operator is asymmetric, van Laarhoven et al's algorithm in fact is *not* guaranteed to converge to a global optimum under theoretical

(let alone practical) conditions; this detail was overlooked in van Laarhoven et al's original analysis. Additionally, Kolonko introduced a variant of simulated annealing that periodically increases the temperature in order to escape local optima, and investigated the performance of the resulting algorithm when used in conjunction with a genetic algorithm. Yamada and Nakano have also developed local search algorithm hybrids that incorporate simulated annealing, e.g., see [YN95].

Comparative studies of local search algorithms for the JSP support the general conclusion that simulated annealing is capable of locating high-quality solutions given sufficiently large run-times [AvLLU94] [VAL96] [BDP96] [JM99]. However, other local search algorithms, e.g., tabu search, can locate equally good solutions in substantially smaller run-times. Consequently, simulated annealing is currently not viewed as a competitive approach for the JSP, although there is evidence that hybridized variants [Kol99] can provide state-of-the-art performance.

9.3 The *MCMC* Algorithm: Definition and Methodological Issues

The analysis presented below in Sections 9.4 through 9.9 is largely based on the core Metropolis algorithm that forms the foundation of simulated annealing. The motivation for such an indirect analysis of simulated annealing is two-fold. First, because search in simulated annealing proceeds via short runs of the Metropolis algorithm over a series of different temperatures, it should be possible to gain insight into the behavior of simulated annealing by analyzing the behavior of Metropolis sampling at various temperatures. Second, experimental results described in Section 9.6 indicate that Metropolis sampling, when executed at a suitable temperature, can outperform simulated annealing.

All experimental results reported in this chapter involve a variant of the basic Metropolis algorithm, denoted *MCMC* for Markov-Chain Monte-Carlo. *MCMC* differs from the standard Metropolis algorithm in two respects. First, neighbors $s' \in N(s)$ of the current solution s are visited sequentially in a random order, such that no neighbor is considered more than once. Experimental results, e.g., see [JAMS89] indicate that a random, non-repeating visitation sequence can improve the performance of simulated annealing, and such a scheme has been employed in several local search algorithms that incorporate Metropolis sampling, e.g., see [RY98]. Second, once all neighbors in $N(s)$ have been tested (and necessarily rejected) once, *MCMC* uses a scheme found in rejectionless Monte-Carlo algorithms to accept a single neighbor $s' \in N(s)$. Let $PM(s', s, T)$ denote the probability of the Metropolis algorithm accepting a neighbor $s' \in N(s)$ from s at temperature T, as given by Equation 9.1. Under the rejectionless scheme, a neighbor $s' \in N$ of s is accepted with a probability distribution according to:

$$P(s'|s) = \frac{PM(s', s, T)}{\Sigma_{x \in N(s)} PM(x, s, T)} \tag{9.2}$$

The rejectionless scheme prevents *MCMC* from repeated test/rejects of the neighbors of s, which occurs frequently at low T; further, the accept probabilities are in the limit identical to those under the basic Metropolis algorithm. Although rejectionless Monte-Carlo could be used at each iteration, the algorithm also requires computation of the makespan of each neighbor in $N(s)$, which at all but very low T is generally not required before a neighbor is accepted.

Pseudo-code for the *MCMC* meta-heuristic is provided in Figure 9.2. *MCMC* is based on the $N1$ move operator; s_{init} is assumed to be a random semi-active solution. Search proceeds via iterative modifications to a single solution; the current solution is denoted s, while the best-so-far solution is denoted s_{best}. The number of iterations is expressed in terms of the number of *accepted* moves. The function genPermutation(x,y) returns a permutation of the integers on the interval $[x, y]$. The function selectRejectionless applies Equation 9.2 to select a neighbor of the current solution s, while the function acceptMetropolis uses Equation 9.1 to determine if a particular neighbor $s' \in N1$ of s should be accepted. Finally, the generate function is responsible for computing the makespan of solution corresponding to the input index i; the pseudo-code assumes a well-defined ordering of the neighbors $N1(s)$.

As in previous chapters, problem difficulty under *MCMC* is characterized in terms of the number of iterations c required to locate an optimal solution. Mirroring *RW*, $TS_{Taillard}$, and *I-JAR*, search cost under *MCMC* is approximately exponentially distributed. Consequently, we define search cost as either the mean $\overline{c}$ or median c_{Q2} number of iterations required to locate an optimal solution, depending on the context. Due to the sensitivity of both measures to extremal values of c, statistics are taken over 1,000 independent trials. Due to the use of the $N1$ move operator, *MCMC* is asymptotically complete for $T > 0$.

In all experiments involving *MCMC*, the temperature T is computed using a reverse-annealing procedure similar to that described by Yamada et al. [YRN94]. In a given fixed-length run of Metropolis sampling (*not MCMC*) at temperature T, let $X \in [0, 1]$ denote the ratio of (1) the total number of moves that are both accepted and yield a neighbor with a makespan strictly worse than the current solution and (2) the total number of moves that are tested, but not necessarily accepted. In the reverse-annealing procedure, T is initially set to 1.0, Metropolis sampling is executed for N iterations, and the ratio X for the sampling period is computed. If X is greater than some user-specified threshold *UAG* (*U*phill-*A*ccepted to *G*enerated), then the current temperature T is returned. Otherwise, T is inflated by a constant factor $\alpha > 1$, and the procedure is repeated. In our experiments, we let N =1,000 and $\alpha = 1.1$. The notation *UAG* and the first temperature T yielding $X >$ *UAG* are used interchangeably.

```
function MCMC(T, MaxIters)
    s = s_init
    s_best = s_init
    numAccepted = 0
    moveAccepted = true
    declare numNeighbors
    declare nextIndex
    declare array neighbors
    while numAccepted ≤ MaxIters do
        if moveAccepted eq true then
            neighbors.clear()
            numNeighbors = #N1(s)
            visitSequence = genPermutation(1, numNeighbors)
            nextIndex = 1
        if nextIndex eq numNeighbors + 1 then
            s = selectRejectionless(s, neighbors)
            moveAccepted = true
        else
            neighbors[visitSequence[nextIndex]] = generate(visitSequence[nextIndex])
            ΔV = F(neighbors[visitSequence[nextIndex]]) − F(s)
            if ΔV ≤ 0 then
                s = neighbors[visitSequence[nextIndex]]
                moveAccepted = true
            else if acceptMetropolis(s, T, neighbors[visitSequence[nextIndex]] then
                s = neighbors[visitSequence[nextIndex]]
                moveAccepted = true
            else
                moveAccepted = false
            nextIndex = nextIndex + 1
        if moveAccepted eq true then
            numAccepted = numAccepted + 1
            if F(s) < F(s_best) then
                s_best = s
    return s_best
end
```

Figure 9.2: Pseudo-code for the *MCMC* meta-heuristic. See text for details.

	UAG					
Instance	0.30	0.25	0.20	0.15	0.10	0.05
	$T \approx 130$	$T \approx 55$	$T \approx 35$	$T \approx 22$	$T \approx 15$	$T \approx 8$
la16	5.3/24.5	4.2/19.8	3.1/15.3	1.9/9.9	1.0/5.7	0.4/2.4
la17	5.1/23.6	4.1/18.8	3.1/14.1	2.1/8.5	1.3/4.3	0.6/1.1
la18	5.6/24.1	4.4/19.3	3.3/14.3	2.2/8.7	1.1/3.7	0.3/0.4
la19	5.0/21.9	4.0/17.6	3.0/12.7	2/0/7.5	1.2/3.5	0.4/0.7
la20	5.3/23.2	4.2/18.4	3.0/12.8	2.1/8.2	1.1/3.3	0.1/0.2
abz5	5.0/22.0	4.2/18.1	3.2/13.2	2.2/8.3	1.3/4.2	0.4/0.4
abz6	4.4/20.1	3.5/15.8	2.6/11.1	1.8/6.8	1.0/3.1	0.5/0.1
ft10	7.7/30.8	6.1/25.4	4.5/19.4	3.0/12.2	1.6/6.0	0.4/0.7

Table 9.1: Search depth and mobility statistics for *MCMC* at various temperatures on well-known 10×10 JSP benchmark instances. An X/Y entry in a cell indicates a search depth of X and a mobility of Y. Statistics are computed for individual trials, each consuming 1,000,000 iterations.

9.4 Local Optima and *MCMC*: A Qualitative Analysis of Run-Time Behavior

The intent of allowing simulated annealing to periodically accept dis-improving moves is, at least ostensibly, to escape the attractor basins of local optima. However, it is unclear to what extent local optima are even relevant to the run-time dynamics of simulated annealing. Specifically, due to the weakness of attractor basins in the JSP, at all but the lowest temperatures the possibility of simulated annealing becoming slowed or trapped in local optima seems relatively remote.

To investigate the relevance of local optima to the behavior of simulated annealing on the JSP, we analyze both the depth and mobility of *MCMC* as a function of search temperature (expressed as *UAG*) on a set of 10×10 benchmark instances. For each instance, 1,000,000 iterations of *MCMC* are executed for temperatures ranging from *UAG* $= 0.05$ to *UAG* $= 0.30$ in increments of 0.05. The *search depth* relative to an individual solution s is defined as the number of iterations of randomized next-descent required to transform s into a local optimum. The search depth over a given trial of *MCMC* is simply the mean search depth averaged over all solutions (i.e., the search depth is computed each time a move is accepted). As indicated in Chapter 7, the actual depth relative to a given solution is stochastic; we ignore this detail because we are averaging over a very large number of samples. *Mobility* is defined as the disjunctive graph distance between two solutions s and s_{-X} encountered X iterations apart. We sample the mobility every 100 iterations, and compute the mean mobility over the entire trial.

The search depth and mobility results for the set of available 10×10 benchmark instances are shown in Table 9.1. As a point of reference, the mean depth of random

		UAG						Mean Depth
Size	Instance	0.30	0.25	0.20	0.15	0.10	0.05	Rnd. Solutions
15×15	ta01	9.3/37.4	7.3/31.1	5.5/24.1	4.0/17.2	2.1/8.3	0.6/1.9	24.4
20×15	ta11	13.2/47.8	11.8/44.6	9.4/38.8	6.7/29.6	3.8/19.3	1.2/6.3	30.1
20×20	ta21	13.7/47.8	11.6/42.2	9.0/34.7	6.5/25.3	3.6/14.0	1.0/3.4	34.1
30×15	ta31	17.3/59.3	15.0/55.4	11.8/49.4	8.8/41.7	5.9/31.3	2.3/13.8	39.3
30×20	ta41	18.1/60.2	15.9/55.8	12.7/49.4	9.5/41.2	6.1/29.1	2.5/13.2	46.2
50×15	ta51	19.7/65.9	18.3/60.3	15.7/62.5	13.5/57.0	9.3/53.4	4.9/39.9	45.1
50×20	ta61	21.3/72.4	18.1/71.5	15.9/64.7	13.3/59.5	8.9/48.5	4.6/31.8	55.5
100×10	ta71	23.9/80.4	20.1/73.8	20.4/67.6	15.9/62.0	10.1/55.2	5.3/38.5	71.3

Table 9.2: Search depth and mobility statistics for *MCMC* at various temperatures on a subset of Taillard's benchmark instances. An X/Y entry in a cell indicates a search depth of X and a mobility of Y. Statistics are computed for individual trials, each consuming 1,000,000 iterations.

semi-active solutions on these same instances is ≈ 13, significantly higher than the results for *MCMC* at *UAG* $= 0.30$, i.e., even at *UAG* $= 0.30$ search is biased, albeit not strongly, toward local optima. Both the depth and mobility monotonically decrease as *UAG* $\rightarrow 0$. Recall that the mobility is defined in terms of the distance between the last 100 accepted, as opposed to tested, solutions. Consequently, a drop in mobility indicates that *MCMC* is expending more effort re-visiting solutions as the temperature is lowered, as is consistent with intuition. Given that local optima in 10×10 random JSPs can be escaped with high probability by accepting only two dis-improving moves, the results suggest that *MCMC* is highly unlikely to become trapped in local optima at moderate-to-high temperatures (i.e., *UAG* $= 0.15$ through *UAG* $= 0.30$). Even at *UAG* $= 0.10$, *MCMC* is consistently 1 to 1.5 moves away from local optima, indicating that search is likely to both encounter and escape from local optima. The only evidence of any significant problem occurs at *UAG* $= 0.05$. Here, search is typically *no more* than 1.5 moves away from a local optimum (the standard deviation of depth for these instances is ≈ 0.6), and the very low mobility indicates that search is repeatedly re-descending to local optima after accepting a dis-improving move.

We also extend the analysis reported in Table 9.1 to one instance of each size represented in Taillard's benchmark suite. The results are reported in Table 9.2, in addition to the mean depth of random semi-active solutions for each instance, as estimated from 1,000,000 independent samples. The qualitative trends are identical to those observed for the 10×10 instances; both depth and mobility are inversely proportional to *UAG*. Additionally, for a given *UAG*, both depth and mobility are proportional to problem size. Given the fact that attractor basin strength is inversely correlated with problem size, this result suggests that *MCMC* should be able to escape attractor basins at temperatures even lower than *UAG* $= 0.05$ on Taillard's larger problem instances.

The ideal value of *UAG* should enable *MCMC* to remain deep enough in the search space to detect local optima, but not too deep as to prevent search from easily escaping the attractor basins of local optima. At moderate-to-high temperatures, search remains relatively distant from local optima, such that it is unlikely to descend deep enough

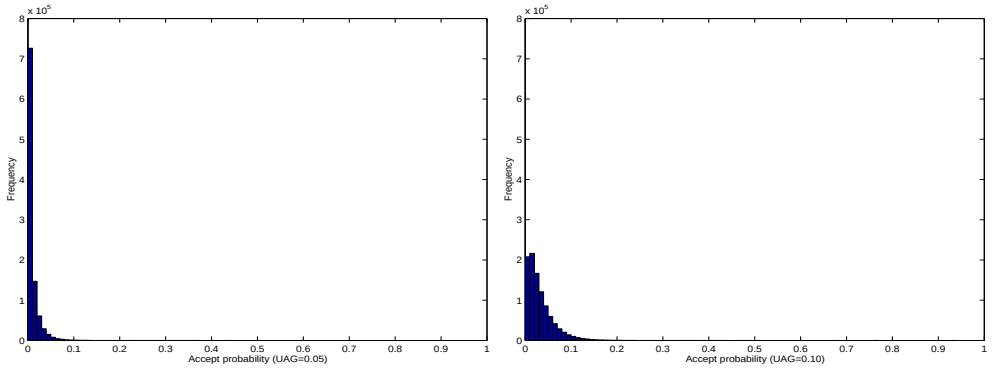

Figure 9.3: Distribution of the probability of accepting two monotonically dis-improving moves under *MCMC* for 10×10 random JSPs, for temperatures corresponding to *UAG* $= 0.05$ (left figure) and *UAG* $= 0.10$ (right figure).

into attractor basins to detect the corresponding local optima. Such search is loosely analogous to attempting to locate the deepest valley in a terrain while traveling in a passenger jet at 30,000 feet; fine details simply cannot be ascertained. At very low temperatures, *MCMC* quickly detects local optima, but can only escape their attractor basins with a significant level of effort. The results presented in Tables 9.1 and 9.2 lead us to hypothesize that *MCMC* is likely to perform best at moderately low temperatures, i.e., *UAG* ≤ 0.10.

9.5 Fitness-Based Escape Probability of Local Optima in the JSP

Local optima are clearly relevant to the dynamics of *MCMC* at relatively low temperatures, raising the question: How hard it is to escape local optima using *MCMC* at low temperatures? Experimental results described in Chapter 5 indicate that the local optima of random JSPs can be escaped with high probability by first accepting a very short random sequence X of monotonically dis-improving neighbors, and subsequently initiating greedy descent. Because the escape mechanism in *MCMC* and its derivatives (i.e., simulated annealing) are fitness-based, these results are not immediately applicable in the context of *MCMC*. Consequently, it is necessary to extend the escapability analysis presented in Chapter 5 to additionally account for the change in fitness involved at each step in the sequence X.

Given a local optimum s, let $X = \{x_0, x_1, ..., x_m\}$ denote a particular random sequence of at most k monotonically dis-improving moves from a local optimum x_0, such that $m \leq k$. The probability $P(X, T)$ of accepting the sequence X under *MCMC* at a fixed temperature T is given by $P(X, T) = \prod_{i=1}^{m} e^{-(F(x_i)-F(x_{i-1}))/T}$. Let $P_{acpt}(x_0, k, T)$ denote the probability of *MCMC* accepting an *arbitrary* random sequence of at most k

monotonically dis-improving neighbors of x_0 at a fixed temperature T. $P_{acpt}(s, k, T)$ is straightforwardly estimated by sampling Y random sequences, and computing the mean $P(X, T)$ of the resulting sequences; in what follows $Y = 100$. Note that in the computation of P_{escp} (defined in Section 5.2.4), it is assumed that $P_{acpt}(x_0, k, T) = 1$, independent of T. Consequently, P_{escp} represents an upper bound on the probability of escaping a local optimum under *MCMC*; in contrast to *I-JAR*, the dependence of the escape mechanism on fitness can only reduce the escape probability.

The distribution of the estimated $P_{acpt}(x_0, k, T)$ for 10,000 random local optima for each of our 100 10×10 random JSPs is shown in Figure 9.3; k is fixed at 2, with T corresponding to $UAG = 0.05$ (left figure) and $UAG = 0.10$ (right figure). In either case, the probability of actually accepting one or two dis-improving neighbors – which is typically required to escape local optima – is surprisingly low, most often below 0.1. Thus, we conjecture that the fitness-based escape mechanism employed by *MCMC* actually causes, by reducing search mobility, poor performance at very low temperatures.

9.6 Is Annealing Necessary to Achieve Competitive Performance?

For both the JSP and other NP-hard problems, very low and very high temperature search is present in most implementations of simulated annealing. Yet, intuition suggests that such search is likely to be ineffective: at very high temperatures, simulated annealing is equivalent to a random walk, while at very low temperatures the probability of escaping local optima is likely to be negligible. Several researchers have confirmed this intuition through detailed experimentation. For example, in their analysis of simulated annealing for the graph partitioning problem, Johnson et al. ([JAMS89], p. 882) conclude that "Although much of the time spent by annealing at very high and very low temperatures seems unproductive ... the amount of time spent at temperatures *between* these limits [has] a large effect on the quality of the solution obtained." However, before attempting to characterize the range of productive temperatures, there is a more fundamental question that can be asked: Is annealing even *necessary*? In other words, are there specific temperatures at which MCMC *alone* can achieve performance that is competitive with, or exceeds that, of simulated annealing.

Generally, it is presumed that simulated annealing is more effective than the MCMC algorithm upon which it is based. Instead, performance is assessed relative to other baselines, or radically different search algorithms. For example, van Laarhoven et al. [vLAL92] only compare the performance of simulated annealing to that of iterative greedy descent and the shifting bottleneck procedure [ABZ88].

To obtain evidence for (or against) the hypothesis that annealing may not be required to achieve effective local search, we execute *MCMC* on a variety of random JSPs at various temperatures, and compare the makespans of the best solutions located during each trial. The resulting performance statistics for well-known 10×10 benchmark

		UAG					
Instance	Optimal Makespan	0.30 $T \approx 130$	0.25 $T \approx 55$	0.20 $T \approx 35$	0.15 $T \approx 22$	0.10 $T \approx 15$	0.05 $T \approx 8$
la16	945	975.7	958.9	949.9	945.1	945.1	955.3
la17	784	785.1	784.1	784	784	784	785.3
la18	848	859.2	852.3	848	848	848	855.3
la19	842	854.4	848.6	842.2	842	842	842.7
la20	902	910.5	905.7	902.1	902	902	921.6
abz5	1234	1247.9	1240.4	1237.1	1234.9	1234.7	1240.1
abz6	943	944.6	943	943	943	943	958.7
ft10	930	1001.4	970.4	946.6	936.9	932.2	950

Table 9.3: The performance of *MCMC* at various temperatures on benchmark 10×10 random JSPs; results for ft10 are also shown. Entries represent the mean makespan of the best solutions found in 30 independent trials.

instances are shown in Table 9.3. With the exception of `ft10`, all of the instances are random JSPs; `ft10` bears strong resemblance to a workflow JSP. The sampling temperature is varied between *UAG* $= 0.05$ and *UAG* $= 0.30$ in increments of 0.05. Each trial of *MCMC* is executed for 10 million iterations (corresponding to no more than one minute of CPU time on a 1.5 Ghz Pentium IV). The entries in Table 9.3 represent the mean makespan of the best solutions found in 30 independent trials.

Analysis of the data in Table 9.3 yields several general observations. First, even relatively long runs of *MCMC* at very high (i.e., *UAG* $= 0.30$ or *UAG* $= 0.25$) and very low (i.e., *UAG* $= 0.05$) temperatures are, with few exceptions, unable to locate optimal or even near-optimal solutions to any of the benchmark instances. Second, at moderately low temperatures (i.e., *UAG* $= 0.10$ and *UAG* $= 0.15$), *MCMC* is able to consistently locate optimal or very near-optimal solutions to *all* of the benchmark instances, including `ft10` (25 of the 30 runs located the optimal solution to the latter instance). Third, with the exception of *UAG* $= 0.05$, performance is inversely proportional to the sampling temperature. Relating the performance of *MCMC* to the results presented in Table 9.1, we observe that *MCMC* performs best at *UAG* $= 0.10$, when the search depth is slightly above 1, as hypothesized above in Section 9.4. Higher temperatures reduce the likelihood of search descending deep enough into attractor basins to detect the corresponding local optima, while lower temperatures prevent search from accepting even 1 or 2 dis-improving moves with any significant probability.

Together, these results demonstrate that *MCMC*, when used in isolation, is capable of locating optimal and near-optimal solutions to a number of well-known benchmark instances. Further, the absolute performance of *MCMC* on these instances is competitive with that of various implementations of simulated annealing reported for the JSP [vLAL88] [YRN94] [YN95] [Kol99] [AvLLU94]. Even taking into consideration relative hardware speed, the performance is still competitive. For example, van Laarhoven

		UAG					
Instance	Optimal Makespan	0.20 $T \approx 38$	0.15 $T \approx 24$	0.10 $T \approx 15$	0.05 $T \approx 10$	*I-JAR*$(k = 2)$	TS$_{\textit{Taillard}}$
ta01	1231	1268.2	1247.7	**1231**	***1231***	1242	1237.4
ta02	1244	1278.3	1257.7	1246.6	1244.9	**1244.7**	1250.6
ta03	1218	1247.5	1223.6	1221.2	**1220.8**	1221.5	1222.3
ta04	1175	1207.1	1184	1179.5	***1175***	1180.4	1180.8
ta05	1224	1265.4	1246.3	1232.7	**1228.6**	1232.4	1232.8
ta06	1238	1273.2	1250.7	1243.4	**1240.3**	1243.2	1243.3
ta07	1227	1251.4	1228.7	**1228**	**1228**	**1228**	**1228**
ta08	1217	1248.2	1229.1	1217.8	**1217.1**	1218.1	1220.1
ta09	1274	1322.1	1292	1280.6	**1280.3**	1281.6	1282.3
ta10	1241	1293.8	1267.6	1244.5	1244	**1243.6**	1250.2

Table 9.4: The mean makespans of the best solutions obtained by *MCMC*, *I-JAR*$(k = 2)$, and TS$_{\textit{Taillard}}$ to Taillard's 15×15 benchmark instances. Statistics are taken over 10 independent trials. Bold-faced entries indicate the mean makespan was less than or equal to that of any competing algorithm. Italicized entries indicate the mean makespan was equal to the optimal makespan.

et al. [vLAL92] obtain a mean best makespan of 933.4 on `ft10` for trials consuming 57,772 seconds of CPU time on a VAX-785; an independent implementation of their algorithm (based on our implementation of *MCMC*) consumes significantly more CPU time than 10 million iterations of *MCMC*.

To assess the performance of *MCMC* relative to the other meta-heuristics analyzed in this thesis, the methodology introduced above is applied to Taillard's 15×15 random JSP benchmark instances. Both *I-JAR*$(k = 2)$ and TS$_{\textit{Taillard}}$ fail to consistently locate optimal solutions to most of these instances, providing at least the potential for *MCMC* to exhibit superior performance. *MCMC* is executed for temperatures ranging from *UAG* $= 0.05$ to *UAG* $= 0.20$ in increments of 0.05; analogous to the results for the 10×10 instances, higher temperatures always yielded lower-quality solutions. Relative to *I-JAR*$(k = 2)$, the CPU time-per-iteration multipliers for *UAG* $= 0.05, 0.10, 0.15$, and 0.20 are respectively 3.0, 3.4, 3.9, and 5. As expected, each iteration of *MCMC* is significantly cheaper than those of *I-JAR*$(k = 2)$, which at a minimum evaluates the complete neighborhood of at least one solution, in addition to the cost of the ascent and descent. To facilitate comparison with the performance of *I-JAR*$(k = 2)$ and TS$_{\textit{Taillard}}$ (originally reported in Section 8.3.3), individual trials of *MCMC* are executed for 10,000,000 $\cdot X$ iterations, where X is the multiplier for a given temperature. Ten independent trials of *MCMC* are executed on each instance at each temperature.

The mean makespans of the best solution obtained by all three meta-heuristics are shown in Table 9.4; statistics are taken over 10 independent trials. Mirroring the results presented in Table 9.3, solution quality is inversely proportional to *UAG*, and at low temperatures, *MCMC* is capable of locating optimal or near-optimal solutions to all of

the problem instances. However, we do not observe a sudden decrease in performance at *UAG* $= 0.05$. Upon closer examination, this is due to the decrease in attractor basin strength as problem size is increased beyond 10×10 (as reported in Chapter 5), such that the temperature at *UAG* $= 0.05$ is not low enough to prevent search from consistently escaping local optima. Comparing the performance of the three meta-heuristics, *MCMC* at *UAG* $= 0.05$ equals or out-performs TS*Taillard* on all but one instance and *I-JAR*$(k = 2)$ on all but 3 instances. Recall that there were no statistically significant differences between the performance of TS*Taillard* and *I-JAR*$(k = 2)$. However, *MCMC* (at *UAG* $= 0.05$) yields statistically significant improvements in performance over the two competitors (as measured by a Wilcoxon sign-rank test at $p < 0.01$) on `ta01` and `ta04`. In summary, the results presented in this section indicate that the performance of *MCMC* is at least competitive with TS*Taillard* and *I-JAR*$(k = 2)$, given roughly equivalent CPU run-times. The latter is ensured through the use of multipliers and a common implementation of the code for evaluating neighboring solution makespans under the $N1$ move operator, which consumes over 95% of the run-time in all three algorithms.

9.7 A Dynamic Cost Model of *MCMC*

We now develop a dynamic cost model of the behavior of *MCMC*, and contrast the resulting model with those developed for *I-JAR* and TS*Taillard*. The results presented in Section 9.6 indicate that *MCMC* is at least competitive with *I-JAR* and TS*Taillard* on 10×10 and 15×15 benchmark instances; there is less conclusive evidence that *MCMC* can actually outperform *I-JAR* and TS*Taillard*. Given sufficiently accurately dynamic models, a comparative analysis should yield a causal explanation for these observations. The structure of the dynamic cost model of *MCMC* and the associated methodology for computing the transition probabilities is identical to that introduced in Chapter 6 for the random walk meta-heuristic *RW*. Specifically, states represent sets of solutions at a given fixed distance from the nearest optimal solution, and transition probabilities are estimated via on-line sampling. Static and quasi-dynamic cost models are not considered in this chapter, as they are only approximations of the full dynamic cost model, as demonstrated in Chapters 6 through 8.

We limit our analysis to 10×10 random JSPs with *UAG* $= 0.10$, 0.15, and 0.20. Both the computation of $\overline{c}$ and estimation of the transition probabilities is prohibitive at lower and higher temperatures, where search under *MCMC* is largely ineffective. As in previous chapters, we only consider those 42 10×10 instances with $\leq$ 100,000 optimal solutions. The transition probabilities for two typical 10×10 instances are shown in Figure 9.4; here, *UAG* $= 0.15$. As with *RW*, *MCMC* induces a bias toward solutions that are roughly equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. However, the bias is generally much weaker than that observed for either *RW* or *I-JAR*, and is comparable to that observed for TS*Taillard*. Often, this bias is neutral for a wide range of distances to the nearest

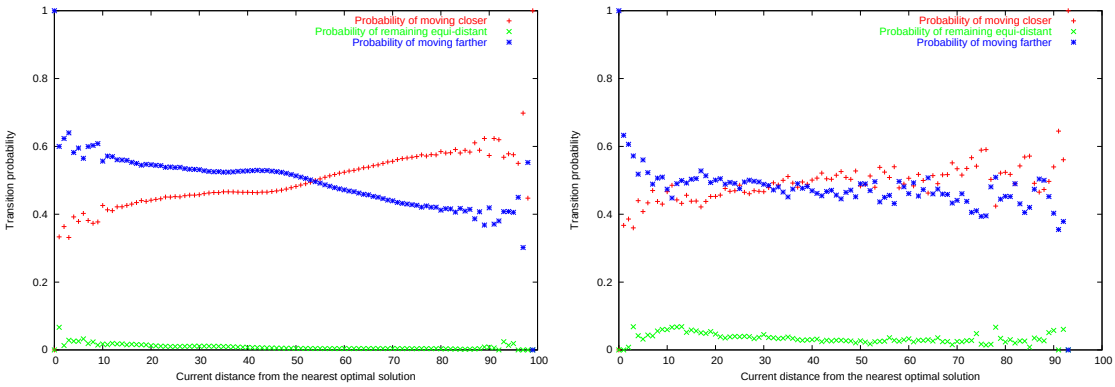

Figure 9.4: Transition probabilities for two 10×10 random JSPs under *MCMC* at *UAG* $= 0.15$ generated via online estimation.

optimal solution, as occurs for the instance shown in the right side of Figure 9.4. Transition probabilities of 10×10 random JSPs are generally more similar to those shown in the left side of Figure 9.4 than to those shown in the right side of Figure 9.4. *MCMC* also appears to induce significantly noisier transition probabilities than any of the other meta-heuristics we have analyzed; upon closer examination, the noise appears to be due to infrequently encountered local optima from which *MCMC* has great difficulty escaping.

Scatter-plots of the $\overline{c}$ predicted by the dynamic cost model versus the actual $\overline{c}$ are shown in Figure 9.5; the predicted $\overline{c}$ are computed using the methodology described in Chapter 6. The r^2 values for the corresponding regression models are 0.9662 (*UAG* $= 0.20$), 0.9833 (*UAG* $= 0.15$), and 0.9682 (*UAG* $= 0.10$), respectively. Results for those 10×10 benchmark instances with $\leq$ 100,000 optimal solutions are included for reference. In all but one case, the actual $\overline{c}$ is within a factor of 3 of the predicted $\overline{c}$; the exceptional case occurs when *UAG* $= 0.20$, with the actual $\overline{c}$ exceeding the predicted $\overline{c}$ by a factor of ≈ 4.8. The dynamic cost model of *MCMC* is roughly 3% to 5% less accurate than the analogous models developed for *RW*, *I-JAR*, and TS$_{Taillard}$. The drop in accuracy is likely due to the existence of noticeable levels of noise in the estimated transition probabilities, which, as indicated above, appears to be due to infrequently encountered local optima with exceptionally strong attractor basins. However, the absolute accuracy of the dynamic cost model indicates that search in *MCMC* can, as with the other meta-heuristics we have analyzed, be viewed as a simple one-dimensional random walk with a bias toward solutions that are roughly equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution.

We explain the roughly equivalent performance of *MCMC*, *I-JAR*, and TS$_{Taillard}$ as follows. First, search in all three meta-heuristics exhibit identical biases, i.e., they are biased toward solutions that are roughly equidistant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. Second, statistical analysis indicates that the maximal distance to the nearest optimal solution in

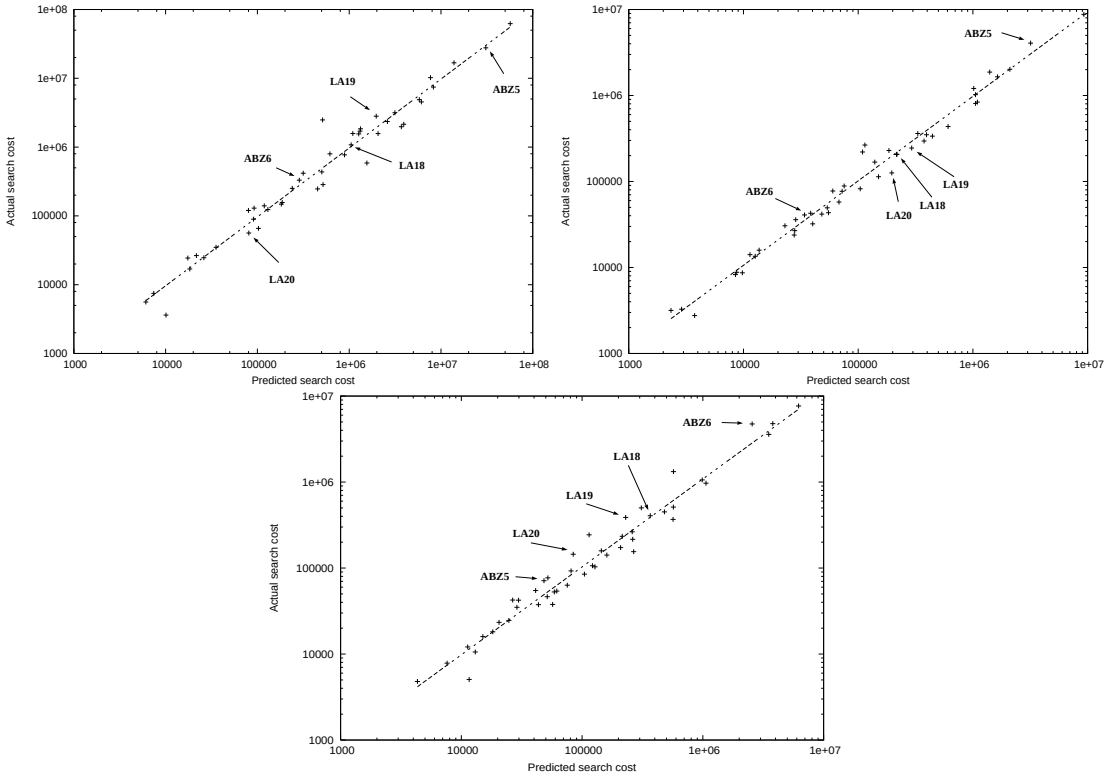

Figure 9.5: Scatter-plots of the observed versus predicted mean cost $\overline{c}$ required to locate an optimal solution under *MCMC* for 10×10 random JSPs at temperatures corresponding to *UAG* $= 0.20$ (upper left figure), *UAG* $= 0.15$ (upper right figure), and *UAG* $= 0.10$ (lower figure); the least-squares fit lines are super-imposed.

all three meta-heuristics is highly correlated. Third, problem difficulty for all three meta-heuristics is a function of both (1) the strength of the search bias and (2) the maximal distance to the nearest optimal solution. Thus, given equivalently strong biases, we would expect no one meta-heuristic to outperform another. Empirically, the bias strength in *MCMC* is typically less than that observed for both *I-JAR* and $\text{TS}_{\textit{Taillard}}$, which is consistent with the slightly inferior performance of *I-JAR* and $\text{TS}_{\textit{Taillard}}$.

9.8 Run-Length Distributions

Mirroring the structure of RLDs for other meta-heuristics (e.g., see Section 6.8), the RLDs for *MCMC* at temperatures corresponding to *UAG* ≥ 0.10 are approximately exponential. For example, at *UAG* $= 0.10$, a two-sample KS test failed to reject the null hypothesis at $p \leq 0.01$ that the RLD is exponentially distributed for 25 of the 42 10×10 random JSPs with $\leq$ 100,000 optimal solutions. For those instances where differences exist, the deviation from the exponential is primarily due to differences in the left tail. As observed in Chapters 6 through 8, easier problem instances are more likely to exhibit

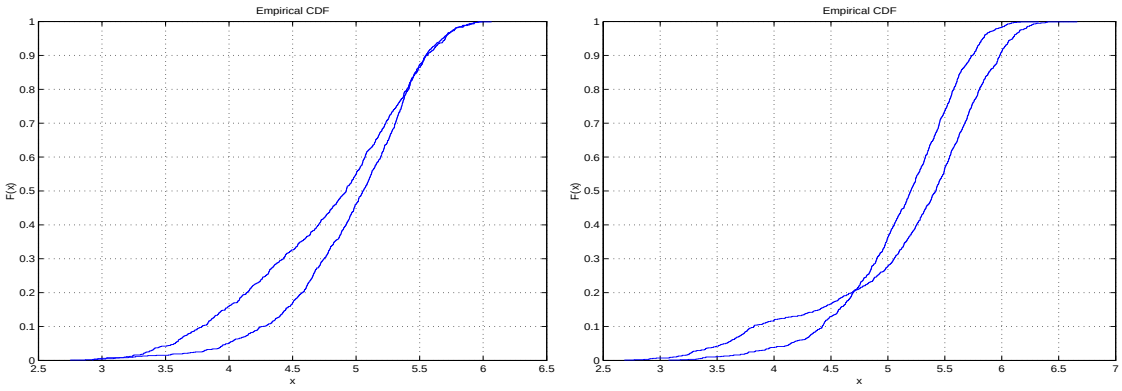

Figure 9.6: CDFs of the predicted and actual RLDs for two 10×10 instances. The p-values for the KS test statistic are respectively $6.34e-6$ and $8.134e-8$.

deviations from an exponential distribution.

As expected given the decreased accuracy of the dynamic cost models, the predicted RLDs often differ substantially from the actual RLDs; using a two-sample KS test, we could reject the null hypothesis that the two distributions were identical at $p \leq 0.01$ for 29 of the 42 random 10×10 JSPs. For the meta-heuristics analyzed in prior chapters, the predicted and actual RLDs were qualitatively identical, such that any differences could be minimized by simply shifting (e.g., through the addition of a constant) one of the two distributions. However, the predicted and actual RLDs under *MCMC* are often qualitatively very different. Examples of two instances for which the two RLDs are dissimilar are shown in Figure 9.6. The source of such differences is currently unclear, although a likely candidate is the presence of infrequently encountered but difficult-to-escape local optima, the same factor that causes significant levels of noise in the estimated transition probabilities.

9.9 Analyzing the Scalability of *MCMC*

As with the prior analyses of tabu search (Chapter 7) and iterated local search (Chapter 8), a key open issue regarding the performance of *MCMC* is scalability. We now consider the scalability of *MCMC*, focusing specifically on the following three aspects:

1. Can *MCMC* locate optimal or near-optimal solutions to larger and more difficult benchmark instances? In other words, is the *absolute* performance of *MCMC* scalable?

2. Can *MCMC* compete with comparable tabu search and iterated local search algorithms on larger and more difficult benchmark instances, as is the case for 15×15 instances. In other words, is the *relative* performance of *MCMC* scalable?

	Problem Size							
	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20
$UAG = 0.05$	7.26	6.48	7.14	7.76	9.58	9.83	10.1	10.15
$UAG = 0.10$	8.14	8.14	8.9	8.9	11.2	11.5	12.2	12.18

Table 9.5: CPU cost-per-iteration multipliers between the baseline *I-JAR*$_{N5}(k = 2)$ and *MCMC*$_{N5}$ at both $UAG = 0.10$ and $UAG = 0.05$ for Taillard's random JSP benchmark instances.

3. How does the performance of *MCMC* compare against state-of-the-art local search algorithms for the JSP? In doing so, our goal is to assess the potential of more complex variations of the Metropolis algorithm for the JSP.

9.9.1 Algorithm and Methodology

Due to both the neighborhood size and the large proportion of neighbors that fail to yield an immediate improvement in the makespan of the current solution, implementations of *MCMC* based on the $N1$ move operator perform poorly on large random JSPs. Significantly stronger performance is obtained using a variant of *MCMC* based on the $N5$ move operator, which we denote *MCMC*$_{N5}$. Other than the difference in the move operator, *MCMC*$_{N5}$ is identical to *MCMC* with the following exception. To avoid becoming trapped in connected sub-components of the fitness landscape, which are due to the fact that the $N5$ operator induces a disconnected search space, *MCMC*$_{N5}$ borrows the monitor-and-escape mechanism employed in both *TS*$_{NS\text{-}A}$ (see Section 8.6.3) and *I-JAR*$_{N5}$ (see Section 8.6.2). Specifically, the mobility M of *MCMC*$_{N5}$ is checked after *TCI* (*T*rap *C*heck *I*nterval) iterations. If M falls below some threshold M_{thrs}, then a random walk from the current solution is initiated. In order to escape the trap, the walk is with respect to the $N1$ operator. If the mobility remains low after another *TCI* iterations, the length of the random walk is increased; the procedure is iterated until super-threshold mobility is achieved. Pseudo-code for the monitor-and-escape mechanism, not repeated here, is shown in Figures 8.11 and 8.12. In the experiments described below, $TCI = 1000$, $M_{thrs} = 10$, $MWL_{nom} = 5$, and $MWL_{inc} = 1$.

As in Chapter 8, the performance of *MCMC*$_{N5}$ is assessed using Taillard's benchmark suite of 80 random JSPs. To achieve approximate equality in the run-times of individual trials, we again turn to cost-per-iteration multipliers relative to the baseline algorithm *I-JAR*$(k = 2)$; the estimated multipliers for *MCMC*$_{N5}$ at $UAG = 0.05$ and $UAG = 0.10$ are shown in Table 9.5. Ten trials of *MCMC*$_{N5}$ are executed on each instance, for the temperatures corresponding to $UAG = 0.05$ and $UAG = 0.10$. For `ta01-ta50`, the best solution located during each trial is recorded. For the large rectangular instances (`ta51`-verb+ta80+), the number of iterations required to locate an optimal solutions is recorded.

9.9.2 Assessing the Relative and Absolute Performance of $MCMC_{N5}$

We first consider the results for Taillard's 15×15 and 20×15 instances, shown in Table 9.6. On the 15×15 instances, the performance of $MCMC_{N5}$ at *UAG=0.05* and *UAG=0.10* is essentially indistinguishable; both variants consistently locate optimal or very near-optimal solutions to all of these instances. In contrast, both absolute (for all 10 instances) and statistically significant (for 2 of the 10 instances) differences in mean performance are observed in the 20×15 instances, with the $UAG = 0.10$ variant providing superior performance. More unexpected is the difference in performance between $MCMC_{N5}$ at $UAG = 0.10$ and the two comparative baselines, $I\text{-}JAR_{N5}(k = 1)$ and $TS_{NS\text{-}A}$. Although indistinguishable for the smaller 15×15 instances, $MCMC_{N5}$ actually *outperforms* $TS_{NS\text{-}A}$, in terms of mean solution quality, on the 20×20 instances; statistically significant differences were detected for 4 of the 10 instances; absolute differences exist for 9 of the 10 instances. Similar results hold when comparing the performance of $MCMC_{N5}$ and $I\text{-}JAR_{N5}(k = 1)$. $MCMC_{N5}$ also consistently locates the best solutions found by any of the algorithms.

The results for Taillard's medium-sized instances, among the most difficult benchmarks available for the JSP, are shown in Table 9.7. As with the smaller instances, the minimum and mean performance of $MCMC_{N5}$ is generally superior at $UAG = 0.10$, although for the 30×20 instances, $MCMC_{N5}$ at $UAG = 0.05$ yields solutions equal to the best located by any of the algorithms tested for 7 of the 10 instances, in contrast to only 4 of the 10 instances under $UAG = 0.10$. Relative to both $I\text{-}JAR_{N5}(k = 1)$ and $TS_{NS\text{-}A}$, $MCMC_{N5}$ at $UAG = 0.10$ yields superior mean solution quality for 23 of the 30 instances. Relative to $TS_{NS\text{-}A}$, the difference is statistically significant for only 5 of the 30 instances. In terms of the the best solution found during any trial, $MCMC_{N5}$ at $UAG = 0.10$ outperforms $TS_{NS\text{-}A}$ on 23 of the 30 instances.

Finally, we contrast the aggregate behavior of $MCMC_{N5}$, $I\text{-}JAR_{N5}$, and $TS_{NS\text{-}A}$ with other state-of-the-art local search algorithms for the JSP. As in Chapter 8, the analysis is based on the mean relative error (MRE) of each algorithm on Taillard's problem instances. The MRE for the 5 sets of Taillard's difficult benchmark instances are shown in Table 9.8. The columns labeled '$MCMC^{*}_{N5}$' and '$MCMC_{N5}$' respectively report the MRE for the best and mean solution quality obtained over 10 trials of $MCMC_{N5}$ at $UAG = 0.10$. The MRE of the best-known makespans is reported in the column labeled 'Best Known'. The remaining column labels are identical to those found in Table 8.7, described in Chapter 8.

As reported in Tables 9.6 and 9.7, it is clear that $MCMC_{N5}$ dominates (albeit not necessarily in a statistical sense) both $I\text{-}JAR_{N5}$ and $TS_{NS\text{-}A}$ in terms of mean solution quality. The mean makespans of solutions obtained by $MCMC_{N5}$ are less than those of both PEZ and BAL, although run-times are not directly comparable. As indicated in Chapter 8, BAL^{*} represents the best solution found in different trials of different variations of Balas and Vazacopoulos' guided local search algorithm [BV98]; consequently, the appropriate comparative baseline is $MCMC^{*}_{N5}$. Mirroring the relative performance

		MCMC$_{N5}$(*UAG=0.05*)			*MCMC*$_{N5}$(*UAG=0.10*)			*I-JAR*$_{N5}$($k=1$)		*TS*$_{NS\text{-}A}$	
Instance	Optimal Makespan	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Min.	Mean
ta01	1231	***1231***	1233.3	1243	***1231***	**1231**	1231	***1231***	1238.2	***1231***	**1231**
ta02	1244	***1244***	1244.1	1245	***1244***	**1244**	1244	***1244***	**1244**	***1244***	1244.4
ta03	1218	*1219*	**1220.8**	1222	1220	1220.8	1223	*1219*	**1220.8**	1221	1221.8
ta04	1175	***1175***	**1175**	1174	***1175***	1175	1175	***1175***	**1175**	***1175***	**1175**
ta05	1224	1230	1230.7	1231	1229	1229.9	1231	1229	1230.9	*1229*	**1229.8**
ta06	1238	1240	1242.7	1244	***1238***	1240.7	1243	***1238***	**1238.8**	1239	1240.7
ta07	1227	*1228*	**1228**	1228	*1228*	**1228**	1228	*1228*	**1228**	*1228*	**1228**
ta08	1217	***1217***	1217.4	1218	***1217***	1217.1	1718	***1217***	**1217**	***1217***	**1217**
ta09	1274	***1274***	1281.3	1283	***1274***	**1277.8**	1282	1281	1282.2	***1274***	1281.8
ta10	1241	***1241***	1243.6	1244	***1241***	1243.6	1244	***1241***	**1242.9**	***1241***	1244.3

		MCMC$_{N5}$(*UAG=0.05*)			*MCMC*$_{N5}$(*UAG=0.10*)			*I-JAR*$_{N5}$($k=1$)		*TS*$_{NS\text{-}A}$	
Ins.	Bounds/ Opt.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Min.	Mean
ta11	1321-1361	1367	1374.2	1382	*1362*	**1371.8**	1377	1373	1377.5	1368	1374.4
ta12	1351-1367	*1374*	1377.4	1382	*1374*	**1376.1**	1377	1377	1379.1	1377	1377.1
ta13	1282-1342	1351	1355	1360	1352	**1353.3**	1354	1352	1354.7	*1350*	1355.4
ta14	1345	***1345***	1346.8	1350	***1345***	**1345**	1345	***1345***	**1345**	***1345***	1345
ta15	1304-1340	1347	1354.1	1360	*1342*	**1346.1**	1353	1346	1355.2	1353	1356
ta16	1302-1360	1365	1368.5	1372	*1362*	**1364**	1367	*1362*	1367.1	1368	1371.4
ta17	1462	1471	1479.3	1494	1470	1475.9	1479	1474	1476.2	*1469*	**1475**
ta18	1369-1396	1408	1411	1415	*1405*	**1409.8**	1412	1409	1411.9	1414	1416.4
ta19	1297-1335	1337	1345.6	1361	*1335*	**1338.8**	1340	1341	1344.9	1343	1346.5
ta20	1318-1351	*1351*	1361.8	1367	1354	**1359.4**	1364	1358	1361.8	1357	1361.8

Table 9.6: Statistics for the makespans of the best solutions obtained by *MCMC*$_{N5}$, *I-JAR*$_{N5}$($k=1$), and *TS*$_{NS\text{-}A}$ to Taillard's small (15×15 – upper portion, 20×15 – lower portion) benchmark instances. Statistics are taken over 10 independent trials. The second column indicates either the optimal makespan, or lower and upper bounds on the optimal makespan. Bold-faced entries in the 'Min' columns indicate equality with the optimal makespan. Italicized entries in the 'Min' columns indicate the best makespan achieved by any algorithm. Bold-faced entries in a 'Mean' column indicates the mean makespan was less than or equal to that of any competing algorithm.

		MCMC$_{N5}$(*UAG=0.05*)			*MCMC*$_{N5}$(*UAG=0.10*)			*I-JAR*$_{N5}$($k=1$)		*TS*$_{NS\text{-}A}$	
Instance	Bounds	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Min.	Mean
ta21	1539-1644	*1647*	1654.7	1662	*1647*	**1649.5**	1655	1658	1661.1	1648	1654.6
ta22	1511-1600	1607	1616.7	1620	*1602*	1615.9	1620	1602	**1612.1**	1613	1620.3
ta23	1472-1557	1561	1565.2	1575	1561	**1562.7**	1566	*1558*	1568.9	1563	1565.9
ta24	1602-1647	1658	1666.2	1674	*1650*	**1654**	1656	1654	1654.2	1654	1656.2
ta25	1504-1595	*1597*	1600.8	1608	*1597*	**1598.6**	1599	1599	1603.6	1599	1601.2
ta26	1539-1645	*1650*	1655.8	1660	1651	**1655**	1656	1655	1661.6	1655	1661
ta27	1616-1680	1691	1695.2	1700	*1689*	**1692.7**	1697	1697	1703	1691	1694.8
ta28	1591-1614	*1617*	1619.5	1622	*1617*	**1618.4**	1621	*1617*	1620.6	1618	1621.4
ta29	1514-1625	*1627*	**1628.4**	1630	*1627*	1628.5	1629	1629	1631.1	1628	1629.2
ta30	1473-1584	*1584*	1595.7	1609	1588	**1590.3**	1595	1592	1600.8	1590	1591.7

		MCMC$_{N5}$(*UAG=0.05*)			*MCMC*$_{N5}$(*UAG=0.10*)			*I-JAR*$_{N5}$($k=1$)		*TS*$_{NS\text{-}A}$	
Instance	Bounds /	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Min.	Mean
	Opt.										
ta31	1764	***1764***	1764.2	1766	***1764***	**1764**	1764	***1764***	**1764**	***1764***	1764.3
ta32	1774-1796	1815	1821.6	1827	*1808*	**1813.9**	1817	1811	1819.7	1811	1816.8
ta33	1778-1793	1803	1814.9	1826	*1802*	**1807.6**	1812	1809	1812.5	1806	1811.6
ta34	1828-1829	1832	1837.2	1872	*1831*	**1832.2**	1837	1833	1834.1	1833	1833.7
ta35	2007	***2007***	**2007**	2007	***2007***	**2007**	2007	***2007***	2007	***2007***	**2007**
ta36	1819	***1819***	1821.3	1831	***1819***	**1819**	1819	***1819***	1822	***1819***	1819.6
ta37	1771-1778	1795	1798.2	1804	*1787*	**1792.5**	1795	1793	1794.2	1793	1794.6
ta38	1673	1676	1682.8	1691	1676	**1679.3**	1684	1683	1684.9	*1675*	1681.5
ta39	1795	1796	1803	1807	1796	1800.6	1806	1797	1798.7	*1795*	**1797.7**
ta40	1631-1674	1686	1691	1697	*1685*	**1689.7**	1693	1695	1696.2	1691	1692.5

		MCMC$_{N5}$(*UAG=0.05*)			*MCMC*$_{N5}$(*UAG=0.10*)			*I-JAR*$_{N5}$($k=1$)		*TS*$_{NS\text{-}A}$	
Instance	Bounds	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Min.	Mean
ta41	1859-2018	*2030*	2039.5	2056	2032	**2037.2**	2044	2042	2045.5	2034	2039.1
ta42	1867-1956	*1966*	1968.9	1974	1967	1972.4	1986	1974	1976	*1966*	**1968.6**
ta43	1809-1859	1879	1887.4	1893	*1875*	1883.3	1889	1889	1890.3	1876	**1882**
ta44	1927-1984	2001	2004.2	2012	*1993*	**2001.5**	2004	2006	2009.2	2002	2005.7
ta45	1997-2000	*2000*	2010.3	2014	*2000*	2009.6	2013	2007	2012.4	2005	**2009.4**
ta46	1940-2021	2033	2041.6	2053	*2031*	2039	2049	2035	**2038.9**	2035	2039.3
ta47	1789-1903	*1920*	1926.6	1938	1922	**1925.7**	1934	1925	1931.3	1926	1929.6
ta48	1912-1952	*1963*	1972.4	1980	1964	**1970.1**	1975	1975	1980.2	1968	1971.9
ta49	1915-1968	*1977*	1986.1	2000	1980	**1983.8**	1989	1991	1992.8	1980	1985.1
ta50	1807-1926	*1939*	1943.5	1954	*1939*	**1943**	1946	1941	1943.9	1940	1945.6

Table 9.7: Statistics for the makespans of the best solutions obtained by *MCMC*$_{N5}$, *I-JAR*$_{N5}$, *TS*$_{NS\text{-}A}$ to Taillard's medium-sized (20×20 – upper portion, 30×15 – middle portion, and 30×20 – lower portion) benchmark instances. Statistics are taken over 10 independent trials. The second column indicates either the optimal makespan, or lower and upper bounds on the optimal makespan. Bold-faced entries in the 'Min' columns indicate equality with the optimal makespan. Italicized entries in the 'Min' columns indicate the best makespan achieved by any algorithm. Bold-faced entries in a 'Mean' column indicates the mean makespan was less than or equal to that of any competing algorithm.

Instance Group	Best Known	$MCMC^{*}_{N5}$	$MCMC_{N5}$	$I\text{-}JAR_{N5}$	$TS_{NS\text{-}A}$	i-TSAB 20M	i-TSAB 50M	PEZ	BAL	BAL*
ta01-10	0.00	0.07	0.15	0.22	0.26	0.11	0.11	0.45	0.25	0.16
ta11-20	2.33	2.66	2.96	3.20	3.40	2.81	2.81	3.47	3.34	2.81
ta21-30	5.45	5.70	5.94	6.27	6.27	5.69	5.68	6.52	6.57	6.10
ta31-40	0.52	0.79	0.96	1.28	1.28	0.85	0.78	1.92	1.13	0.80
ta41-50	4.12	4.74	5.08	5.37	5.24	4.97	4.7	6.04	5.71	5.20

Table 9.8: Mean relative error (MRE) of various algorithms on Taillard's difficult benchmark instances. See text for details.

of $MCMC_{N5}$ and BAL, we observe that $MCMC^{*}_{N5}$ dominates BAL*.

By design, the run-times allocated to each trial of $MCMC_{N5}$ are approximately equivalent to the run-times consumed by 50 million iterations of i-TSAB. Given the fact that $MCMC_{N5}$ does not employ re-intensification, it is unsurprising that its performance *in terms of mean solution quality* is inferior to i-TSAB. However, the best solutions obtained by $MCMC_{N5}$ are nearly equivalent to those obtained by individual runs of i-TSAB on `ta31-ta50`, and are actually *lower* than i-TSAB on `ta01-ta20`.

Chapter 10

The Impact of Problem Structure on Landscapes and Cost Models

Mirroring the majority of prior research on the JSP, the analyses presented in previous chapters are devoted to modeling the behavior of local search algorithms on *random* JSPs. However, the ultimate goal of research on approximation algorithms for the JSP, or for any other optimization problem, is success in solving difficult, real-world problem instances. In contrast to their random counterparts, real-world problem instances often exhibit significant structure. This structure can have a significant impact on search algorithm performance [WBWH99] [WBWH02], raising the question of generalization: Do cost models and related insights obtained from the analysis of local search algorithm behavior on random JSPs extend to more structured JSPs? If so, it should be possible to leverage existing models to explain differences in the difficulty of random and structured JSPs. If not, it is important to identify the boundaries of existing models, and to understand the broader issue of how problem structure impacts algorithm performance.

In the context of the JSP, the term *structure* can refer to two aspects of a problem instance: the job routing orders π_i and the operation durations τ_{ij}. Although the τ_{ij} in real-world scheduling problems are typically non-random [PDS73], the τ_{ij} in all available benchmark instances for the JSP are random, uniformly sampled from a fixed-width interval such as $[1, 99]$. In contrast, workflow and flowshop JSPs possess non-random π_i. In this chapter, structure refers to structure in the job routing orders, and not the operation durations. Although not considered here, it is relatively straightforward to extend previously reported methods for generating permutation flowshop problem instances with non-random operation durations to the JSP [WBWH02].

The analysis presented below contrasts the fitness landscapes and cost models developed for random JSPs with those of workflow and flowshop JSPs. The order-of-magnitude increase in the difficulty of workflow and flowshop JSPs over random JSPs necessitates restricting much of the analysis to sets of 6×4 and 6×6 problem instances. However, it is possible to analyze the impact of structure on the cost models of some local search algorithms using small sets of 10×10 instances. All problem instances

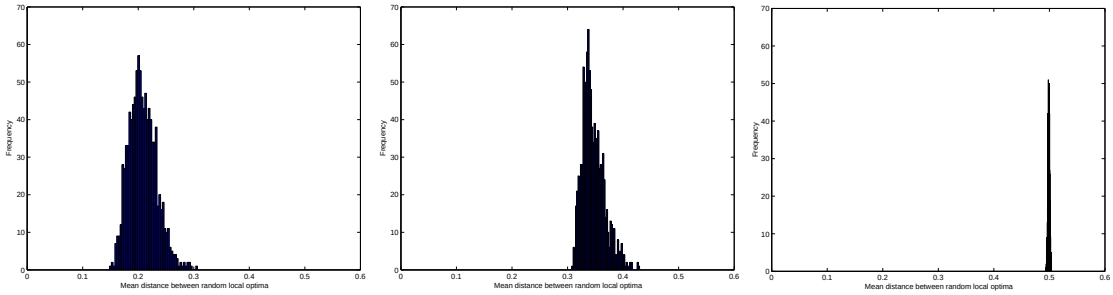

Figure 10.1: Histograms of the mean distance between random local optima ($\overline{d}_{\textit{lopt-lopt}}$) for 6×6 random (left figure), workflow (center figure), and flowshop (right figure) JSPs.

considered in this chapter are fully described in Section 4.6.3.

The rest of this chapter is organized as follows. The fitness landscapes of random and structured JSPs are contrasted in Section 10.1, with a particular focus on measures for quantifying the absolute and effective size of the space of feasible solutions (S) and the sub-space of local optima and near-local optima ($S_{\textit{lopt}}$ and $S_{\textit{lopt+}}$, respectively). The impact of structure on cost models of *RW* and $\textit{TS}_{\textit{Taillard}}$ is analyzed in Sections 10.2 and 10.3, respectively. Qualitatively identical results hold for *I-JAR* and *MCMC*, and are not discussed. The chapter concludes in Section 10.4 with a summary of key results and a discussion of their implications.

10.1 Contrasting the Fitness Landscapes of Random Versus Structured JSPs

Mattfeld et al. [MBK99] argue that differences in the relative difficulty of random versus workflow JSPs are largely due to differences in the size of the search space. Consider the sub-space $S_{\textit{lopt}}$ of local optima. The absolute size of the sub-space $S_{\textit{lopt}}$ (as well as $S_{\textit{lopt+}}$) can be estimated by the mean distance between random local optima, denoted $\overline{d}_{\textit{lopt-lopt}}$. The distributions of $\overline{d}_{\textit{lopt-lopt}}$ for 6×6 random, workflow, and flowshop JSPs are shown in Figure 10.1; each histogram represents data for the 1,000 instances in the corresponding problem set. The distributions indicate that $S_{\textit{lopt}}$ is roughly 75% larger on average in workflow JSPs than in random JSPs. The size of $S_{\textit{lopt}}$ in flowshop JSPs is, ignoring statistical fluctuations, equal to the theoretical maximum of 0.5, and is significantly larger than the values observed in workflow JSPs. Thus, differences in the size of $S_{\textit{lopt}}$ (as measured by $\overline{d}_{\textit{lopt-lopt}}$) are at least consistent with the observation that JSPs become more difficult as the number of workflow partitions is varied from 1 to m (see Section 2.3). Similar shifts in the distribution of $\overline{d}_{\textit{lopt-lopt}}$ are observed in 6×4 problem sets. Qualitatively identical results also hold when considering the absolute size of the space of feasible solutions S, as measured by $\overline{d}_{\textit{rand-rand}}$.

A key deficiency of the $\overline{d}_{\textit{lopt-lopt}}$ measure is that it fails to account for the number

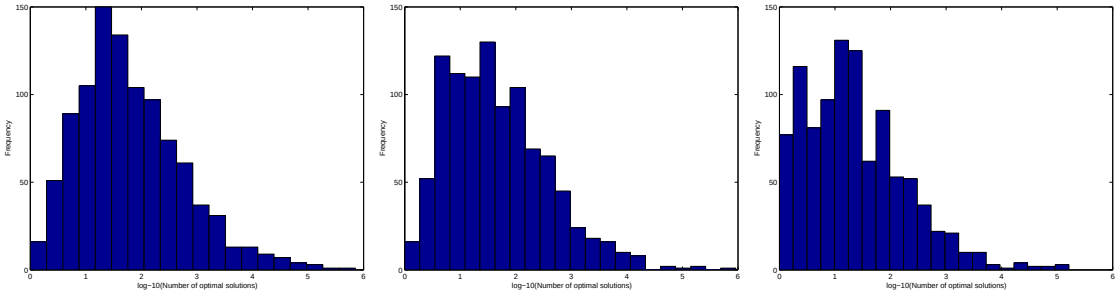

Figure 10.2: log_{10} histograms of the number of optimal solutions ($|optsols|$) for 6×6 random (left figure), workflow (center figure), and flowshop (right figure) JSPs.

and/or distribution of optimal solutions within the S_{lopt} sub-space. In order for differences in $\overline{d}_{lopt\text{-}lopt}$ to account for the relative difficulty of random, workflow, and flowshop JSPs, we would expect the number of optimal solutions $|optsols|$ to either remain constant or drop as the number of workflow partitions *wf* increases; any increase in $|optsols|$ would likely decrease the effective size of the search space, yielding an overall reduction in problem difficulty. This intuition is confirmed in Figure 10.2, which shows the log_{10} distribution of $|optsols|$ for 6×6 random, workflow, and flowshop JSPs. Although the extreme right-tail mass (with $log_{10}(|optsols|) > 4$) of the problem sets is essentially indistinguishable, there are more workflow and flowshop instances with small (e.g., ≤ 10) numbers of optimal solutions than in the random problem set. Similar differences are also found in the mean $|optsols|$ for the various problem sets, computed as 2,374, 2,061, and 894 for the random, workflow, and flowshop JSPs, respectively. Thus, the absolute size of the S_{lopt} sub-space is proportional to the number of workflow partitions *wf*, while at the same time the number of optimal solutions drops as *wf* is increased. This suggests that the effective size of S_{lopt} increases with increases in *wf*. Qualitatively identical results hold for 6×4 JSPs. Finally, there is a slight decrease in the overall correlation between $|optsols|$ and the backbone size in workflow and flowshop JSPs (from ≈ 0.92 to ≈ 0.8), primarily due to poor correlation in instances with very large numbers of optimal solutions.

To test the hypothesis that increases in *wf* are correlated with increases in the effective size of S_{lopt}, we analyze changes in the distribution of the $\overline{d}_{lopt\text{-}opt}$, the mean distance between random local optima and the nearest optimal solution. Recall that the $\overline{d}_{lopt\text{-}opt}$ measure directly quantifies the effective size of the S_{lopt} sub-space. The distribution of $\overline{d}_{lopt\text{-}opt}$ for 6×6 random, workflow, and flowshop JSPs are shown in Figure 10.3. The results clearly confirm the prediction that the effective size of the S_{lopt} sub-space increases, on average, with increases in *wf*. The mean $\overline{d}_{lopt\text{-}opt}$ for random and workflow JSPs is respectively 0.157 and 0.3248. The mean $\overline{d}_{lopt\text{-}opt}$ for flowshop JSPs grows to 0.4467, frequently nearing the theoretical maximum of 0.5. Qualitatively identical results hold for 6×4 JSPs, and for the analogous $\overline{d}_{rand\text{-}opt}$ measure for quantifying the effective size of the space S of feasible solutions.

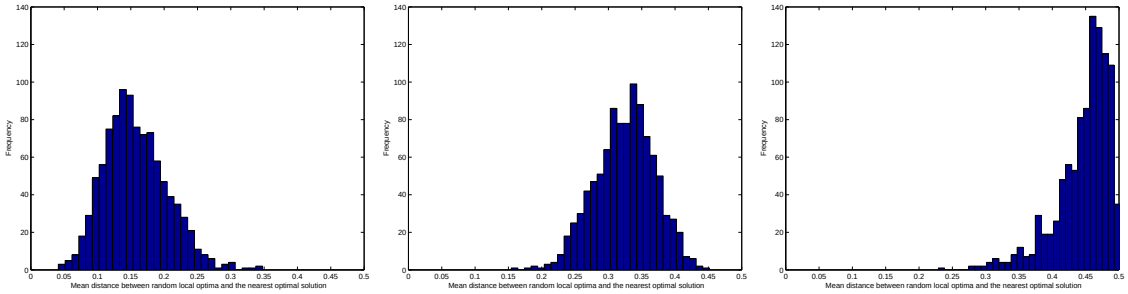

Figure 10.3: Histograms of the mean distance between random local optima and the nearest optimal solution ($\overline{d}_{lopt\text{-}opt}$) for 6×6 random (left figure), workflow (center figure), and flowshop (right figure) JSPs.

10.2 The Impact of Structure on Cost Models of *RW*

In Chapters 6 through 9, we showed that static and quasi-dynamic cost models based on the effective size of the search space ($\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{X\text{-}opt}$, respectively, where X denotes a particular meta-heuristic) accounted for much of the variability in problem difficulty for local search on small random JSPs. In contrast to $\overline{d}_{lopt\text{-}opt}$, the accuracy of the $\overline{d}_{X\text{-}opt}$ model scaled to larger problem instances. Further, $\overline{d}_{X\text{-}opt}$ estimates a key parameter of the dynamic cost model, which in turn accounts for nearly all of the variability in the difficulty of random JSPs for local search. We now investigate whether similar results hold for cost models of local search on structured JSPs. The subsequent analysis considers cost models of the random walk algorithm *RW*; qualitatively identical results hold for the two other memoryless local search algorithms (*I-JAR* and *MCMC*) investigated in prior chapters. *RW* was selected due to its unique status as a baseline. A similar analysis for $TS_{Taillard}$ is described below in Section 10.3.

Before analyzing the impact of structure on cost models of *RW*, it is illustrative to quantify the relative difficulty of random versus structured JSPs. Under *RW*, the mean c_{Q2} for 6×4 random and workflow JSPs is 4,860 and 215,708 iterations, respectively, indicating that the workflow instances are nearly 2 orders of magnitude more difficult than random instances. The increase is less dramatic when moving from workflow to flowshop instances. The mean c_{Q2} for the latter is 461,512, such that flowshop JSPs are roughly twice as difficult as workflow JSPs. Statistics for larger (e.g., 6×6) structured JSPs are unavailable due to the excessive computational requirements. Consequently, the following analysis is restricted to 6×4 random, workflow, and flowshop JSPs. However, qualitatively identical results hold for *I-JAR* and *MCMC* on both 6×4 and 6×6 problem sets.

The accuracy of static cost models is impacted by problem structure. Table 10.1 reports the r^2 values for various static cost models of *RW* for 6×4 random, workflow, and flowshop JSPs. The models are based on various measures related to the size (both absolute and effective) of the space S of feasible solutions, i.e., $log_{10}(|optsols|)$, $\overline{d}_{rand\text{-}rand}$,

	Fitness Landscape Feature		
Problem Set	$log_{10}(\|optsols\|)$	$\overline{d}_{rand\text{-}rand}$	$\overline{d}_{rand\text{-}opt}$
6×4 *wf=1*	0.4981	0.3098	0.8088
6×4 *wf=2*	0.7177	0.0000	0.6636
6×4 *wf=m*	0.8236	0.0005	0.8780

Table 10.1: The r^2 values of static cost models of the cost required by *RW* to locate optimal solutions to 6×4 random, workflow, and flowshop JSPs.

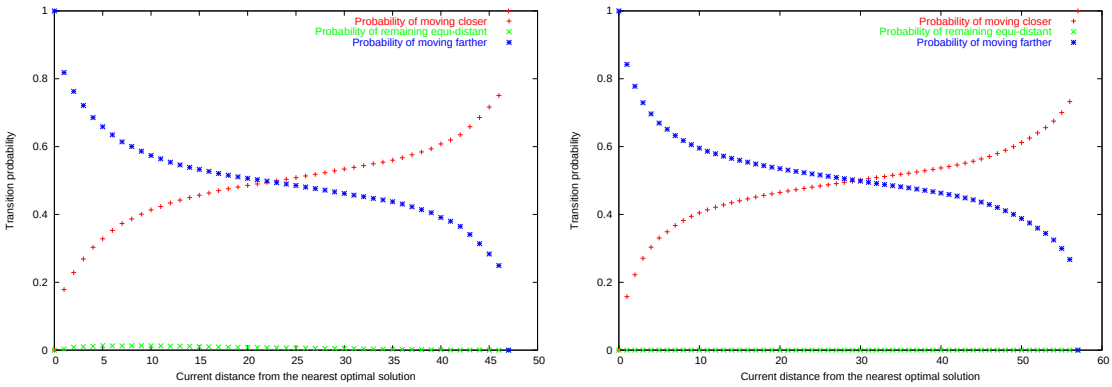

Figure 10.4: Transition probabilities under *RW* for typical 6×4 workflow (left figure) and flowshop (right figure) JSPs.

and $\overline{d}_{rand\text{-}opt}$. The results clearly demonstrate the variable impact of structure on the accuracy of static cost models. For example, the number of optimal solutions accounts for *more* of the variability in search cost as *wf* is increased, while the mean distance between random solutions accounts for none of the variability in search cost observed in workflow and flowshop instances. In retrospect, these two observations are consistent with intuition: as the variability in $\overline{d}_{rand\text{-}rand}$ is decreased (approaching 0 in the case of flowshop JSPs), difficulty is dictated entirely by the number and/or distribution of optimal solutions. The drop in the accuracy of the $\overline{d}_{rand\text{-}opt}$ cost model when $wf = 2$ raises the possibility that the dynamics of *RW* may be qualitatively different on workflow JSPs. However, the accuracy of quasi-dynamic cost models based on $\overline{d}_{rw\text{-}opt}$ indicate that such differences, if they exist, are likely to be minimal; the corresponding r^2 values for 6×4 random, workflow, and flowshop JSPs are 0.9539, 0.8595, and 0.9124, respectively.

As in random JSPs, random solutions to structured JSPs are not necessarily representative of solutions visited by *RW* during search. Consequently, we use the on-line estimation methodology introduced in Section 6.6 to compute the set of transition probabilities under *RW* for the 6×4 workflow and flowshop JSPs. Figure 10.4 shows the resulting probabilities for typical 6×4 workflow and flowshop instances. In contrast to the probabilities observed for random JSPs (e.g., see Figure 6.7), there typically exists more curvature in the probabilities near the minimal and maximal distance from the nearest optimal solution. However, the bias toward solutions that are roughly equi-distant from

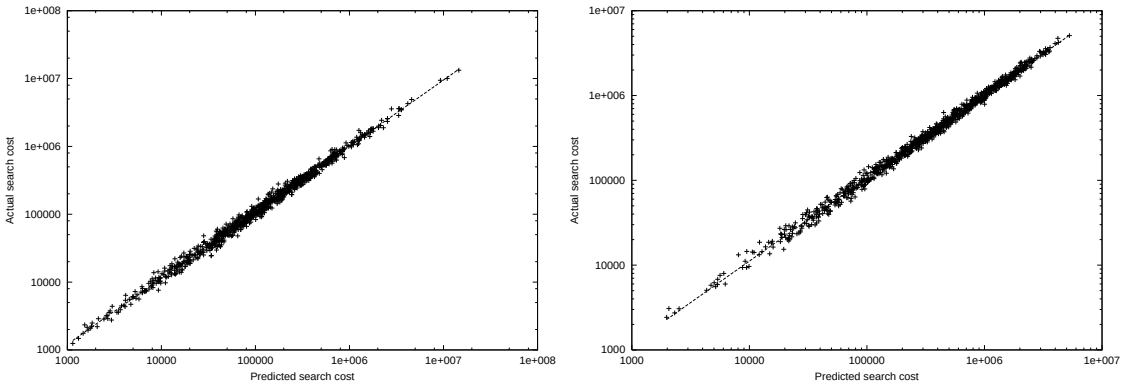

Figure 10.5: Scatter-plots of the observed versus predicted mean cost $\overline{c}$ to locate an optimal solution under *RW* for 6×4 workflow (left figure) and flowshop (right figure) JSPs; the least-squares fit lines are super-imposed.

the nearest optimal solution and solutions maximally distant from the nearest optimal solution persists. Ignoring the curvature at extreme values of d_{opt}, the key difference between the transition probabilities for random and structured JSPs is D, which estimates the maximal distance to the nearest optimal solution; as indicated in Section 6.7, D is another measure of the effective size of the search space S.

To assess the impact of the accentuated curvature on the accuracy of the dynamic cost model of *RW*, we compare the predicted versus actual mean search cost $\overline{c}$; the predicted cost is computed using the methodology described in Section 6.6. log_{10}-log_{10} scatter-plots of the predicted versus actual $\overline{c}$ for 6×4 workflow and flowshop JSPs are shown in Figure 10.5. The r^2 values of the corresponding regression models are 0.9931 and 0.9951, respectively. In all cases, the actual $\overline{c}$ is within a factor of 2 of the predicted $\overline{c}$. For *I-JAR* and *MCMC*, $r^2 \geq 0.95$ for the analogous dynamic cost models on 6×4 and 6×6 workflow and flowshop JSPs.

Clearly, structure in the job routing orders has no impact on the accuracy of the dynamic cost models of *RW*, and at best negligible impact on the dynamic cost models of *I-JAR* and *MCMC*. When executing on random and structured JSPs, the search dynamics of all three algorithms are accurately modeled as a one-dimensional random walk with a bias toward solutions that are roughly equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. This distance, as measured by $\overline{d}_{X\text{-}opt}$, accounts for over 80% of the variability in the cost of locating optimal solutions to both random and structured JSPs. The remaining variability is due to instance-specific fluctuations in the transition probabilities, which a simple summary statistic is unlikely to capture.

	Fitness Landscape Feature		
Problem Set	$log_{10}(\|optsols\|)$	$\overline{d}_{lopt\text{-}lopt}$	$\overline{d}_{lopt\text{-}opt}$
6×4 *wf=1*	0.5365	0.2415	0.8044
6×4 *wf=2*	0.6504	0.0024	0.6241
6×4 *wf=m*	0.6075	0.0000	0.7022
6×6 *wf=1*	0.2223	0.2744	0.6424
6×6 *wf=2*	0.3345	0.0030	0.3004
6×6 *wf=m*	0.3268	0.0007	0.4104

Table 10.2: The r^2 values of static cost models of the cost required by $TS_{Taillard}$ to locate optimal solutions to 6×4 and 6×6 random, workflow, and flowshop JSPs.

10.3 The Impact of Structure On Cost Models of $TS_{Taillard}$

Due to its effectiveness, it is feasible to assess the impact of structure on cost models of $TS_{Taillard}$ using larger JSPs than was possible in the case of *RW*. Specifically, the analysis presented below proceeds primarily in the context of both 6×4 and 6×6 problem sets. As expected, the relative difficulty of the instances increases as *wf* is varied from 1 to m. For 6×6 random, workflow, and flowshop JSPs, $TS_{Taillard}$ yields respective mean c_{Q2} values of 280, 3,137, and 12,127; for these computations, $L_{min} = 6$ and $L_{max} = 12$, with the tabu tenure re-sampled every 15 iterations. The differences represent nearly an order-of-magnitude increase in average difficulty as *wf* is varied from 1 to $m/2$ and again from $m/2$ to m; similar results hold for 6×4 JSPs. For 10×10 random, workflow, and flowshop JSPs, the mean c_{Q2} are respectively 315,413, 4.35×10^7, and 2.62×10^8; for these computations, $L_{min} = 8$ and $L_{max} = 14$, with the tabu tenure re-sampled every 15 iterations. Again, we observe an order-of-magnitude increase in problem difficulty as *wf* is varied from both 1 to $m/2$ and from $m/2$ to m. To further illustrate the extreme difficulty of these instances, we note that the most difficult flowshop instance required more than 900 *million* iterations on average to locate an optimal solution.

We first assess the impact of structure on the accuracy of static cost models of $TS_{Taillard}$. As indicated in Chapter 7, search in $TS_{Taillard}$ is largely restricted to the sub-space $S_{lopt+} \subseteq S$ containing local optima and solutions that are near, in terms of distance, to local optima. Three measures related to the absolute and effective size of S_{lopt+} are $log_{10}(|optsols|)$, $\overline{d}_{lopt\text{-}lopt}$, and $\overline{d}_{lopt\text{-}opt}$. The r^2 values of static cost models of $TS_{Taillard}$ based on these features are shown in Table 10.2 for 6×4 and 6×6 random, workflow, and flowshop JSPs. As expected given decreased variability, the accuracy of the $\overline{d}_{lopt\text{-}lopt}$ model drops to ≈ 0 when $wf = 2$ and $wf = m$; in response, the accuracy of the $|optsols|$ model increases. The drop in the accuracy of the $\overline{d}_{lopt\text{-}opt}$ model at $wf = 2$ observed for *RW* also persists. Analogous to the case for random JSPs, static cost models of $TS_{Taillard}$ are somewhat less accurate than those of *RW* on identical problem sets.

The inaccuracies of the $\overline{d}_{lopt\text{-}opt}$ model on 6×6 workflow and flowshop JSPs again

raises the possibility that structured JSPs induce fundamentally different search dynamics in $TS_{Taillard}$. This conjecture is further supported by the absolute accuracy of quasi-dynamic models based on $\overline{d}_{tabu\text{-}opt}$. For 6×6 workflow and flowshop JSPs, the r^2 values of the $\overline{d}_{tabu\text{-}opt}$ model are 0.5519 and 0.5524, respectively, in contrast to $r^2 = 0.7808$ for 6×6 random JSPs. In other words, the effective size of the search space, as quantified by $\overline{d}_{tabu\text{-}opt}$, only accounts for slightly over half the variability in problem difficulty observed for structured JSPs.

To analyze the potential for qualitative changes in the run-time dynamics of $TS_{Taillard}$ on structured JSPs, we compute sets of estimated transition probabilities for 6×4 and 6×6 workflow and flowshop JSPs using the on-line sampling methodology described in Section 7.9. The probabilities of $TS_{Taillard}$ continuing to move closer to the nearest optimal solution for two 6×6 flowshop JSPs are shown in Figure 10.6. For the instance corresponding to the left-hand figure, the probability of continuing to move closer to the nearest optimal solution decreases as search moves closer to the nearest optimal solution, which is consistent with the results observed for random JSPs (e.g., see Figure 7.16). However, in contrast to the results for random JSPs, the probability of moving closer to the nearest optimal solution frequently drops well below 0.5 when search is very near an optimal solution. For the instance corresponding to the right-hand figure, the probability of continuing to move closer to the nearest optimal solution is nearly constant at ≈ 0.6. In contrast, the probability of inverting the gradient when search is moving away from the nearest optimal solution is proportional to the current distance from the nearest optimal solution, and is generally quite small. The examples shown in Figure 10.6 illustrate a key point regarding the behavior of $TS_{Taillard}$ on workflow and flowshop JSPs: the transition probabilities are significantly more heterogeneous than those observed for random JSPs, often deviating significantly from the prototypical form in *both* qualitative and quantitative aspects. Given such large deviations, it is unsurprising that cost models based on simple summary statistics (e.g., $\overline{d}_{tabu\text{-}opt}$) are unable to account for a significant proportion of the variability in problem difficulty.

To assess the impact of large instance-specific irregularities in the transition probabilities on the accuracy of the dynamic cost model of $TS_{Taillard}$, we again compare the predicted versus actual $\overline{c}$; the predicted $\overline{c}$ are computed using the methodology introduced in Section 7.9. The results for 6×4 and 6×6 workflow JSPs are shown in Figure 10.7; the r^2 values of the corresponding $log_{10} - log_{10}$ regression models are 0.9973 and 0.9806, respectively. The absolute accuracy is only slightly worse for 6×4 and 6×6 flowshop JSPs, with respective r^2 values of 0.9760 and 0.9777; the corresponding scatter-plots are shown Figure 10.8. In all but a few exceptional cases, the actual $\overline{c}$ is within a factor of 3 of the predicted $\overline{c}$.

We also analyze the scalability of the dynamic cost model to structured 10×10 instances. Due to their difficulty, it is computationally prohibitive to estimate transition probabilities for any of our 10×10 flowshop JSPs. For our 10×10 workflow JSPs, it is possible to estimate transition probabilities for those 8 instances with $\leq$ 10,000 optimal

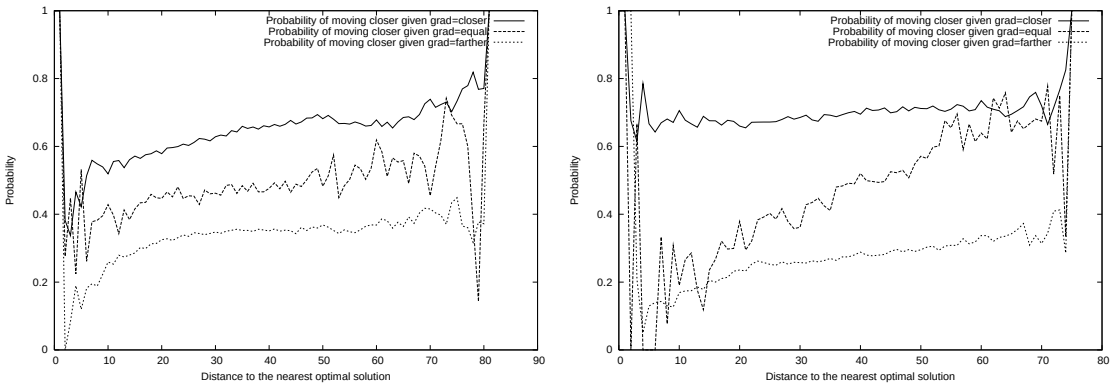

Figure 10.6: The transition probabilities for moving closer to the nearest optimal solution under *TS*$_{Taillard}$ for two different 6×6 flowshop JSPs.

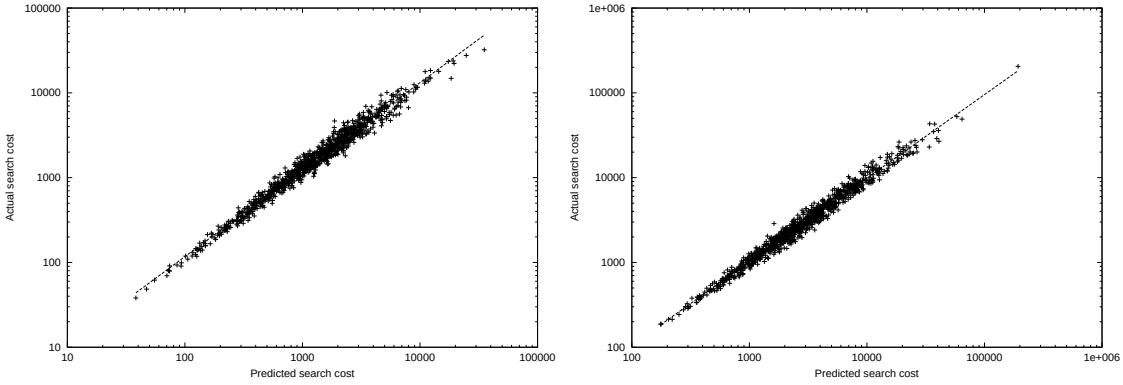

Figure 10.7: Scatter-plots of the predicted versus actual mean cost ($\overline{c}$) required to locate an optimal solution under *TS*$_{Taillard}$ to 6×4 (left figure) and 6×6 (right figure) workflow JSPs; the least-squares fit lines are super-imposed.

solutions with reasonable computational effort. As indicated in Section 4.6.3, the only structured 10×10 benchmark instances are `orb01-orb10`; the job routing orders of these instances are similar to those found in workflow and flowshop JSPs (which subclass depends on the particular instance). Of these, transition probability estimation was feasible for `orb01-orb07`, each of which possess fewer than 10,000 optimal solutions. A $log_{10} - log_{10}$ scatter-plot of the predicted versus actual $\overline{c}$ for these instances is shown in Figure 10.9. The r^2 value of the corresponding regression model is 0.9863, and in no case did the actual $\overline{c}$ exceed the predicted $\overline{c}$ by more than a factor of 2.

The results presented above clearly indicate that simple aggregate Markov models can accurately capture the fundamental dynamics of *TS*$_{Taillard}$ on both random and structured JSPs. The effective size of the S_{lopt+} sub-space is less of a factor in problem difficulty in structured JSPs, due primarily to the more heterogeneous nature of the transition probabilities. At the same time, the decreased accuracy of the dynamic cost model on structured JSPs raises the possibility for further and potentially more signifi-

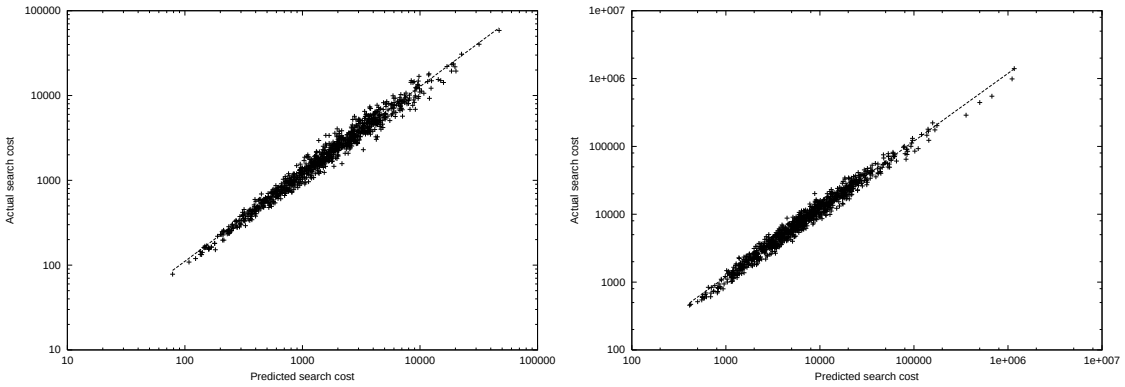

Figure 10.8: Scatter-plots of the predicted versus actual mean cost ($\overline{c}$) required to locate an optimal solution under $TS_{Taillard}$ to 6×4 (left figure) and 6×6 (right figure) flowshop JSPs; the least-squares fit lines are super-imposed.

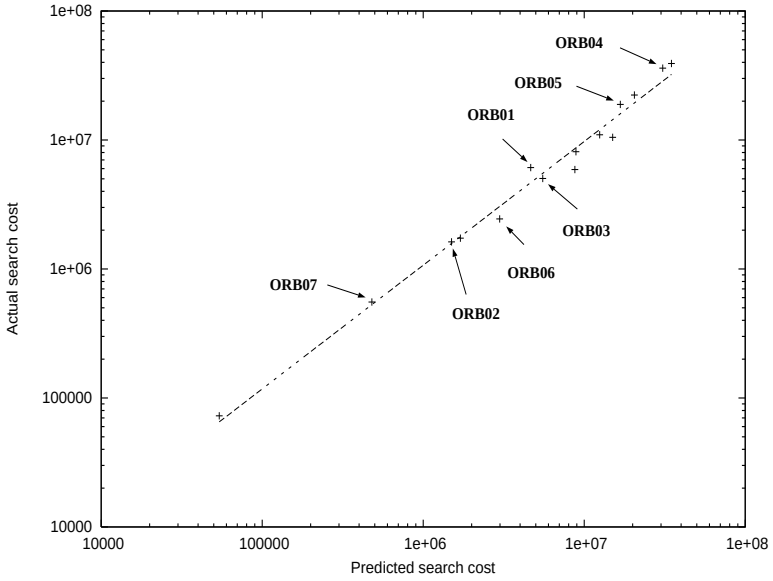

Figure 10.9: Scatter-plots of the predicted versus actual mean cost ($\overline{c}$) required to locate an optimal solution under $TS_{Taillard}$ to 10×10 workflow and structured benchmark JSPs; the least-squares fit lines are super-imposed.

cant degradation on problem instances with fundamentally different types of structure – for example, strong correlation between sub-sets of the operation durations τ_{ij}.

10.4 Re-Assessing the Causal Factor for Differences in the Relative Difficulty of Random and Structured JSPs

The analyses presented in this chapter extends the results of Mattfeld et al., confirming their original hypothesis in spirit, although not detail: differences in the relative difficulty of random, workflow, and flowshop JSPs are primarily due to differences in the size of the search spaces. Mattfeld et al. focused on differences in the *absolute* size of the S_{lopt+} sub-space, as measured by $\overline{d}_{lopt\text{-}lopt}$. However, as shown throughout this thesis, the *effective* size of the search space, as quantified to varying degrees of accuracy by $\overline{d}_{lopt\text{-}opt}$, $\overline{d}_{X\text{-}opt}$, and D, is significantly more correlated with problem difficulty than $\overline{d}_{lopt\text{-}lopt}$. In fact, $\overline{d}_{lopt\text{-}lopt}$ is at best weakly correlated with problem difficulty, and frequently un-correlated. Consequently, the results presented in this chapter serve to clarify Mattfeld et al.'s original assertion, by identifying a search space measure that is both highly correlated with problem difficulty for fixed-sized problem sets *and* and exhibits significant differences between fixed-sized problem sets with different *wf*. The $\overline{d}_{lopt\text{-}lopt}$ only succeeds at the latter.

It is also clear that the variability in problem difficulty observed for structured JSPs is unlikely to be fully captured by simple summary statistics, i.e., static or quasi-dynamic cost models. The transition probabilities induced by structured JSPs can be significantly more heterogeneous than those induced by random JSPs; large instance-specific irregularities are not uncommon. However, these irregularities have little impact on the dynamic cost models developed in Chapters 6 through 9; local search can still be modeled with high fidelity as relatively simple Markov chains.

Chapter 11

Analyzing the State-of-the-Art: Does the Core Meta-Heuristic Really Matter?

Although $MCMC_{N5}$, $IJAR_{N5}$, and $TS_{NS\text{-}A}$ all provide excellent performance on a wide range of benchmark instances, they fall just slightly short of the state-of-the-art, as currently represented by Nowicki and Smutnicki's i*TSAB* algorithm [NS03]. i*-TSAB* is a tabu search algorithm based directly on Nowicki and Smutnicki's own well-known *TSAB* algorithm [NS96]. *TSAB* is largely a straightforward implementation of tabu search based on the $N5$ move operator, albeit with two key differences. First, *TSAB* maintains a pool of *elite solutions* consisting of high-quality solutions previously encountered during search. Second, *TSAB* periodically re-initiates search from an elite solution when the core tabu search algorithm fails to improve on the best-so-far solution after a pre-specified number of iterations; the core tabu search algorithm is equivalent to $TS_{NS\text{-}A}$, with the exception of code devoted to cycle detection and recovery. The strategy of periodically restarting search from previously encountered high-quality solutions is known as reintensification [GL97].

The notation i-TSAB stands for *iterated TSAB*, in which *TSAB* is treated as the core meta-heuristic. Search in i-TSAB proceeds in two phases. In the first phase, multiple runs of *TSAB* are used to generate a pool of elite, high-quality solutions. In the second phase, pairs of elite solutions are selected and *path relinking* [GL97] is used to construct a new solution that is approximately equidistant from each elite solution; *TSAB* is then initiated from the intermediate solution. If the best solution located by *TSAB* improves upon either of the parent elite solutions, it replaces one of the parents in the elite solution pool. The strategy of restarting search from solutions distant from previously encountered high-quality solutions is an example of *diversification* [GL97]. In summary, *TSAB* improves upon straightforward implementations of tabu search, e.g., $TS_{NS\text{-}A}$, through the use of reintensification, while i*-TSAB* improves upon *TSAB* through the use of diversification.

Given the strong similarity between *TS$_{NS\text{-}A}$* and *TSAB*, the state-of-the-art status of i-*TSAB* can likely be attributed to the use of re-intensification and diversification. Further, it is also well-known that both strategies can significantly improve the performance of tabu search algorithms for a wide variety of other NP-hard problems. What is less clear is the degree to which the performance of i-*TSAB* depends on tabu search. A reasonable hypothesis states that re-intensification and diversification can equally improve the performance of other meta-heuristics, specifically iterated local search and Metropolis sampling. This hypothesis is driven by two factors: (1) the relative equity in the performance of *MCMC$_{N5}$*, *I-JAR$_{N5}$*, and *TS$_{NS\text{-}A}$* observed in Chapters 8 and 9 and (2) the strong similarities in the dynamic cost models of these meta-heuristics. In other words, given minimal differences between these core meta-heuristics, it seems unlikely that the performance of i-*TSAB* depends on tabu search.

The remainder of this chapter is devoted to testing this hypothesis. In Section 11.1, through the introduction of a common framework, we define variations of our three meta-heuristics (*MCMC$_{N5}$*, *I-JAR$_{N5}$*, and *TS$_{NS\text{-}A}$*) that employ both re-intensification and diversification. The framework is significantly more straightforward than i-*TSAB* and allows us to test hypotheses related to whether the full complexity of the i-*TSAB* algorithm is integral to its superior performance. The comparative methodology is described in Section 11.2, and the experimental results are documented and analyzed in Section 11.3. The key results of the chapter, along with implications of the experimental results, are discussed in Section 11.4.

11.1 A Generic Framework for Re-Intensification and Diversification

We now introduce a general framework that enhances a given core meta-heuristic (e.g., tabu search or metropolis sampling) with re-intensification and diversification. We denote this framework by i-*RDLS*, for *i*terated *R*eintensification and *D*iversification for *L*ocal *S*earch. Much of the terminology and algorithmic details underlying i-*RDLS* are borrowed from Nowicki and Smutnicki [NS03] and are not reproduced here. i-*RDLS* is a simplified version of Nowicki and Smutnicki's i-*TSAB* algorithm. The primary difference is that search in i-*RDLS* is not hierarchical, in that diversification and re-intensification are applied at the same level. As with i-*TSAB*, search in i-*RDLS* proceeds in two phases. In the first phase, a procedure denoted *RDLS*, which is analogous to *TSAB*, is initiated from random local optima to generate the initial pool of elite solutions E. Search in the second phase proceeds through a series of iterations in which diversification and re-intensification procedures are applied to solutions in E, with equal probability in any given iteration. When re-intensification is performed, *RDLS* is initiated using a random solution $e \in E$; if a solution e' with a lower makespan than e is located, then e' replaces e in E. When diversification is performed, two distinct solutions $a, b \in E$ are selected at random. A path relinking procedure described in [NS03] is

Algorithm	Problem Size	Total Iterations	X_a	X_b
i-*MCMC*$_{N5}$	15×15	111M	44.4K	11.1K
i-*MCMC*$_{N5}$	20×15	127M	101.6K	17.8K
i-*MCMC*$_{N5}$	20×20	143M	114.4K	20.02K
i-*MCMC*$_{N5}$	30×15	151M	120.8K	21.14K
i-*MCMC*$_{N5}$	30×20	188M	150.4K	26.32K
i-*I-JAR*$_{N5}$	15×15	20M	8K	1.6K
i-*I-JAR*$_{N5}$	20×15	22M	17.6K	3.1K
i-*I-JAR*$_{N5}$	20×20	22M	17.6K	3.1K
i-*I-JAR*$_{N5}$	30×15	27.5M	22K	3.9K
i-*I-JAR*$_{N5}$	30×20	27M	21.6K	3.8K

Table 11.1: Parameter settings for trials involving i-*MCMC*$_{N5}$ and i-*I-JAR*$_{N5}$.

applied to a and b to generate a new solution c that is roughly equidistant from both a and b. *RDLS* is then initiated from c; if the best solution located c' has a lower makespan than either a or b, then c' replaces that solution in E. Search proceeds until a user-specified number of iterations of the core *RDLS* algorithm have been performed.

The *RDLS* algorithm initially executes a core meta-heuristic M from a given initial solution s_{init}. At least X_b iterations of M are performed from s_{init}. If a solution s' with $C_{max}(s') < C_{max}(s_{best})$ is located, then at least X_a additional iterations are performed from s'. The idea is to continue to execute M as long as improving solutions are being located. Using i-*RDLS*, we instantiate extensions to the *N5*-based meta-heuristics introduced in Chapters 7 through 9. The resulting algorithms are denoted i-*MCMC*$_{N5}$, i-*I-JAR*$_{N5}$, and i-*TS*$_{NS\text{-}A}$. We only investigate i-*I-JAR*$_{N5}$ with $k = 1$ and i-*MCMC*$_{N5}$ with *UAG* $= 0.10$.

11.2 Comparing the Performance of the Enhanced Meta-Heuristics: Methodology

The performance of all four algorithms is compared using a subset of Taillard's random JSP benchmark instances: `ta01` through `ta50`. Ten trials of each algorithm are executed on each instance. All of the larger rectangular instances (`ta51` through `80`) can be quickly solved to optimality by the core variants of these algorithms (i.e., those not employing re-intensification and diversification) and are therefore not considered in this analysis. Performance is quantified in terms of both (1) the best solution found during any of the 10 trials and (2) the median solution quality obtained over the 10 trials. In both cases, we measure the instance *MRE* (see Section 8.6.5) relative to either the optimal makespan, if known, or the largest known lower bound. The *MRE* of the makespan of the best solution found in any trial by algorithm X is denoted X^b, while the *MRE* of the median makespan over the 10 trials is denoted X^m.

Given the similarities between i-*RDLS* and i-*TSAB*, we use the following parameter settings, borrowed directly from Nowicki and Smutnicki [NS03]. We made no attempt

Instance Group	Best Known	i-MCMC$^b_{N5}$	i-MCMC$^m_{N5}$	i-I-JAR$^b_{N5}$	i-I-JAR$^m_{N5}$	i-TS$^b_{NS\text{-}A}$	i-TS$^m_{NS\text{-}A}$	i-TSAB
ta01-10	0.00	0.03	0.11	0.05	0.07	0.04	0.14	0.11
ta11-20	2.33	2.52	2.55	2.57	2.80	2.58	2.78	2.81
ta21-30	5.45	5.66	5.70	5.64	5.81	5.58	5.82	5.68
ta31-40	0.52	0.69	0.73	0.74	0.85	0.66	0.84	0.78
ta41-50	4.12	4.38	4.6	4.43	4.89	4.35	4.74	4.7

Table 11.2: Mean relative error (MRE) of various algorithms on Taillard's difficult benchmark instances.

to further modify or optimize these values, primarily due to the computational cost of performing additional experiments. In all of our experiments, we let $|E| = 8$. To enable direct comparison with Nowicki and Smutnicki's reported results for i-*TSAB*, we execute each trial for an equivalent of 50 million iterations of *TS*$_{NS\text{-}A}$. The values X_a and X_b are also specified relative to *TS*$_{NS\text{-}A}$, given as follows. For 15×15 problem instances, we let X_a=20,000 and X_b=4,000. In trials involving other instances, X_a=40,000 and X_b=7,000. Although computable from the multipliers presented in Tables 8.3 and 9.5, for completeness, we show the resulting CPU per-iteration multipliers in Table 11.1.

11.3 Comparing the Performance of the Enhanced Meta-Heuristics: Results

To simplify the presentation, only the mean *MRE* on each set of identically-sized instances are described. The performance results for each of our algorithms are shown in Table 11.2, in addition to results for two baselines: i-*TSAB* and the *MRE* for the best-known (or optimal, if available) solutions, i.e., the makespan of the best solution located to date by *any* of the myriad algorithms developed for the JSP. Because the results are derived from a single trial (as reported by Nowicki and Smutnicki [NS03]), the performance of i-*TSAB* is comparable only to the median performance of each of our three algorithms, i.e., X^m where X denotes one of our three algorithms. Comparing the performance of i-*TSAB* with of i-*TS*$_{NS\text{-}A}$, we observe minimal differences: the median solution quality obtained by i-*TS*$_{NS\text{-}A}$ is slightly better than that of the single trial of i-*TSAB* on the 20×20 instances and slightly worse on the remaining instances. In particular, the difference in the *MRE* is only 0.04% and 0.06% for 30×20 and 30×15 problem sets, respectively – which contain some of the most difficult benchmark instances available. The largest difference is only 0.14%, which occurs for the 20×20 problem set. Unfortunately, it is impossible to assess whether the differences are statistically significant, given the results of a single trial of i-*TSAB*. At *worst*, we can conclude that i-*TS*$_{NS\text{-}A}$ only *slightly* underperforms i-*TSAB*. A more realistic assessment attributes the differences to the use of a single trial of i-*TSAB*, such that the performance of i-*TSAB* and i-*TS*$_{NS\text{-}A}$ are likely indistinguishable. In support of this view, we note that, in initial experimentation, individual trials of i-*TS*$_{NS\text{-}A}$ yielded smaller differences in *MRE* than

that observed for the median of multiple trials.

Next, we compare the performance of the remaining two algorithms with that of i-*TSAB*, again using the median solution quality of the former as the basis of comparison. The solutions obtained by i-*TSAB* are slightly better than the median solution quality obtained by i-*I-JAR*$_{N5}$. For all but the largest problem set, the maximum difference is 0.13%. The underperformance grows to 0.19% on the 30×20 problem set, which is consistent with the fact that *I-JAR*$_{N5}$ on average slightly underperforms *TS*$_{NS\text{-}A}$. In contrast, the median solution quality obtained by *MCMC*$^{m}_{N5}$ is generally *superior* to that of i-*TSAB*; the sole exception occurs for the 20×20 instances, where the difference is only 0.02%. As with i-*TS*$_{NS\text{-}A}$, i-*I-JAR*$_{N5}$, at worst, only slightly underperforms i-*TSAB*. Although it is tempting to conclude that *MCMC*$_{N5}$ outperforms i-*TSAB*, the lack of statistics associated with i-*TSAB* prevents this conclusion; however, we can conclude that *MCMC*$_{N5}$ is competitive with i-*TSAB*.

Finally, we analyze the *MRE* of the best solutions located by any of our algorithms. As indicated above, the comparison is not strictly fair: for any given fixed algorithm, the best solution obtained by multiple trials will always be no more than the median solution quality. For all problem sets, *MRE* of the best solutions obtained by all of our algorithms is lower than that of i-*TSAB*. While not surprising, this result is important, as it is consistent with the hypothesis that the median performance of all four algorithms is effectively indistinguishable. Interestingly, although the median performance of i-*TS*$_{NS\text{-}A}$ was slightly inferior to that of i-*MCMC*$_{N5}$, the former locates the best overall solutions. Although our goal was not to establish a new baseline for the state-of-the-art, we do observe that the *MRE* of the best solutions obtained by i-*TS*$_{NS\text{-}A}$ is the lowest reported to date. Finally, we observe that the *MRE* of the best solutions located by any of our algorithms is still as much as 0.23% higher than the that of the best-known solutions; this is despite the fact that four new best-known solutions (which are not included in the computation of the *MRE* reported in the second column of Table 11.2) were located. Clearly, a status of state-of-the-art does *not* guarantee that the algorithm cannot be *occasionally* outperformed on *individual* instances by algorithms that are, on average, judged inferior.

11.4 What Makes the State-of-the-Art the State-of-the-Art?

The results presented above provide strong evidence for two general conclusions regarding the behavior of state-of-the-art local search algorithms for the JSP. First, it does not appear that a core tabu search meta-heuristic is required to achieve state-of-the-art performance on the JSP. Under controlled experimental conditions, we showed that variants of iterated local search and Metropolis sampling can achieve equivalently strong performance. However, two components *do* appear to be required to achieve state-of-the-art performance: the combination of re-intensification with diversification and a restricted

move operator, i.e., $N5$. Since the introduction of Taillard's algorithm in the mid-1990s, the most effective algorithms for the JSP have consistently, with few exceptions, been based on variants of tabu search. The results presented above indicate that the prominent status of tabu search is largely unjustified. In contrast, our results reinforce, via direct experimentation, the idea that the analysis and improvement of long-term memory mechanisms should be emphasized in future research on meta-heuristics.

Second, although of less significance, our results also suggest that Nowicki and Smutnicki's i-*TSAB* algorithm may be simplified somewhat without sacrificing performance. In particular, the algorithmic framework described in Section 11.1 eliminates both the multi-level search employed by i-*TSAB* and various less significant features. Although not pursued here, the modifications simplify the analysis of i-*TSAB*-like algorithms and support comparison of the effect of different relative levels of re-intensification and diversification on performance.

Chapter 12

Summary, Implications, and Future Research Directions

The majority of research on local search algorithms is devoted to the development of newer, better-performing variants. In terms of research priorities, scientific modeling and analysis of existing algorithms runs a distant second. This disparity has had a significant adverse impact on the local search community. In particular, the lack of fundamental models and theories has led to rampant speculation regarding the benefits and/or behavior of particular local search algorithms and has allowed ad-hoc development paradigms to persist, likely impeding advances in the field. The research presented in this thesis serves as an important first step toward rectifying the current situation.

Using the JSP as a test-bed, we developed empirical models of several well-known and widely-studied algorithms, specifically tabu search, iterated local search, and Metropolis sampling. The resulting models explicitly capture the key behavioral characteristics of these algorithms and additionally enable us to disprove many conjectures reported in the literature on local search. Ultimately, scientific models are judged on their ability to both account for observations beyond their original scope and to generate new behavioral hypotheses. The models we develop satisfy both criteria. Our models are both consistent with and provide explanations for a number of previously unexplained observations regarding problem difficulty for local search in the JSP. Additionally, individual models and comparison of different models suggest many novel hypotheses regarding the behavior of local search algorithms, which are confirmed through subsequent experimentation. The broad explanatory and predictive (in the sense of identifying new behavioral characteristics) power of our models reinforces their potential to serve as the basis for a broader theory of local search.

In the remainder of this chapter, we summarize our key contributions and discuss future research directions. Our discussion is intentionally high-level, with a goal of placing the results in a broader context. Various results concerning specific algorithms – despite their importance and relevance – are not recounted and are detailed in Chapters 6 through 9.

12.1 Cost Models of Local Search

For any local search algorithm, the primary behavior of interest is the cost required to locate optimal solutions to problem instances, i.e., problem difficulty. Consequently, our objective in modeling local search algorithms for the JSP is to account for the variability in problem difficulty observed within ensembles of problem instances. The core contribution of this thesis is a set of *cost models* of local search, which account for nearly all of the variability in problem difficulty over a range of different types of problem instance.

With relatively few exceptions, local search algorithms are designed to achieve a single over-riding objective: to either escape or altogether avoid local optima. Little attention is generally paid to the higher-level search strategy. In the absence of an explicit global search strategy, a reasonable assumption is that local search algorithms simply perform variations of random walks, e.g., over the sub-space of local optima. Under the random walk assumption, problem difficulty is essentially a function of the size of the search space. However, the number of optimal solutions also plays a role; the presence of multiple optimal solutions increases the probability that local search need not search the entire space before encountering one. Consequently, we hypothesize that problem difficulty for local search is a function of the *effective* size of the search space, i.e., after taking into account the number and density of optimal solutions.

We confirm this hypothesis by developing a series of cost models, which explicitly capture the run-time dynamics of local search. Our models indicate that the global strategies underlying the local search algorithms we consider are simple variations of a one-dimensional random walk, e.g., of the type commonly associated with the well-known Gambler's Ruin problem. Further, local search is biased toward solutions that are roughly equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. An analysis of the structural characteristics of the resulting cost models indicates that problem difficulty for local search is a function of (1) the effective size of the search space, as originally hypothesized, and (2) the strength of the search bias toward solutions that are distant from optimal solutions. This inference is supported by the near-perfect accuracy of the cost models, which account for over 96% of the variability in problem difficulty in the *worst* case.

The availability of accurate cost models also enables us to dispel many conjectures regarding the behavior and/or benefits of particular local search algorithms. The results may be viewed as disappointing, in that a number of well-known local search algorithms are effectively behaving as simple random walks. However, this result is also intriguing, as these strategies or simple variations of these strategies are able to yield excellent and state-of-the-art performance – even in comparison to a broad range of other algorithmic paradigms.

12.2 The Explanatory Power of Cost Models

The explanatory power of the cost models we developed extends beyond the primary behavior of interest: variability in the cost required to locate optimal solutions. The literature on the JSP documents a wide range of observations relating to problem difficulty and algorithm behavior for local search. To date, few scientific explanations for these phenomena have been put forth. In contrast, our cost models are both entirely consistent with and provide concrete explanations for these phenomena. The ability to account for behaviors beyond their original intent strongly reinforces the correctness and utility of the proposed cost models. The specific observations and the corresponding explanations are as follows:

- *The relative difficulty of square versus rectangular JSPs.* Empirical evidence indicates that 'square' JSPs (i.e., $n/m \approx 1$) are generally harder than 'rectangular' JSPs (i.e., $n/m \gg 1$). Further, JSPs generally become easier with increases in n/m, despite the corresponding explosion in the size of the search space. We observe that as $n/m \rightarrow \infty$, the number of optimal solutions rapidly increases. The increase in the number of optimal solutions offsets the increase in the size of the search space, such that the net result is a *decrease* in the size of the effective search space. Consequently, differences in the relative difficulty of square versus rectangular JSPs are simply due to differences in the effective sizes of the search spaces.

- *The relative difficulty of random versus structured JSPs.* The presence of particular types of structure in the job routing orders of problem instances has a dramatic influence on problem difficulty in the JSP. Specifically, workflow JSPs are known to be significantly more difficult than random JSPs, while flowshop JSPs are generally more difficult than workflow JSPs. As is the case with square versus rectangular JSPs, the differences in difficulty are due to differences in the effective size of the search space; flowshop and workflow JSPs typically possess fewer optimal solutions than random JSPs, while the search spaces are significantly larger.

- *The algorithm-independent nature of problem difficulty.* The relative difficulty of different JSPs benchmark instances tends to be somewhat stable. In other words, instances that are difficult (easy) for a particular local search algorithm tend to be difficult (easy) for most local search algorithms. This observation can be explained by noting that (1) the effective size of the search space is largely independent of the meta-heuristic employed by a local search algorithm (assuming a fixed move operator) and (2) problem difficulty is a function of the effective size of the search space for a wide range of local search algorithms.

- *Dramatic changes in the difficulty of locating sub-optimal solutions with nearly identical fitness.* Consider sub-optimal solutions with makespans $X - 1$ and X

greater than the optimal makespan. Several researchers have noted that the difficulty of locating the respective solutions can differ dramatically, often by an order of magnitude or more, despite nearly identical fitness. Simple extensions of our cost models fully account for the variability in the cost required to locate sub-optimal solutions to problem instances. Our analysis indicates that large differences in the difficulty of locating sub-optimal solutions with makespans $X-1$ and X greater than the optimal makespan are simply due to differences in the effective size of the search spaces containing the sub-optimal solutions.

Our cost models demonstrate that many local search algorithms are simple variations of straightforward random walks and, as a consequence, the cost of locating optimal solutions is simply a function of the effective size of the search space. The results presented above strongly reinforce this view: changes in the effective search space size account for a wide range of phenomena relating to both problem difficulty and local algorithm behavior in the JSP.

12.3 The Predictive Power of Cost Models

Accounting for existing behavioral observations is only the first step in developing and validating powerful models (i.e., a theory) of local search. The ultimate test of a model lies in its ability to predict the existence of new, previously unobserved behavioral phenomena. Such predictions are key as they expose potential deficiencies, whose correction leads to more accurate models, by subjecting the model to falsifiability testing. Analyses of our cost models raise several novel hypotheses regarding the behavior of local search algorithms for the JSP. Subsequent empirical testing confirms each of these hypotheses, providing exceptionally strong evidence as to the validity and generality of our cost models. The specific hypotheses and their implications are as follows:

- *Heuristic initialization is unlikely to reduce the cost required by local search to locate optimal solutions.* Several researchers have argued that heuristically constructed initial solutions can significantly improve the performance of local search. However, our cost models indicate that unless the initial solution is extremely close to the nearest optimal solution, the difficulty of locating optimal solutions is largely invariant. In practice, methods for constructing high-quality initial solutions yield solutions that are distant from optimal solutions, and, consequently, have at best negligible impact on the cost of locating optimal solutions.

- *The performance of different local search algorithms is largely indistinguishable.* The cost models of different local search algorithms exhibit very strong similarities. Specifically, all of the algorithms we considered can be modeled as straightforward variations of a random walk, with qualitatively identical biases and quantitatively similar dimensions. Given such strong similarities, we would *a priori*

expect no algorithm to significantly outperform another. Experimental evidence verifies this prediction, demonstrating minimal to non-existent differences in the performance of tabu search, iterated local search, and Metropolis sampling. This result is extremely surprising given the recent dominance of tabu search in the JSP literature, in particular, in comparative studies of local search algorithm performance.

- *State-of-the-art performance in local search algorithms for the JSP requires intensification and diversification, but not tabu search.* For the last 10 years, tabu search algorithms have consistently provided state-of-the-art performance on the JSP. Specifically, Nowicki and Smutnicki have introduced a series of tabu search algorithms that make extensive use of long-term memory mechanisms in addition to the core tabu search meta-heuristic. The underlying similarity of tabu search, iterated local search, and Metropolis sampling suggest that tabu search is *not* an integral feature of these algorithms. This hypothesis is confirmed through experimentation, which further indicates that long-term memory mechanisms equally improve the performance of all three meta-heuristics.

12.4 Implications and Future Research Directions

Although the research presented in this thesis represents an important first step toward a theory of local search, significant work remains.

We have yet to fully explore the implications of our cost models for the design of new local search algorithms. Specifically, our results provide a clear design objective: to minimize the bias that guides search away from optimal solutions. Without fundamental changes in the move operator and/or the fitness function, the effective size of the search space - and consequently problem difficulty - is unlikely to be changed. However, it may still be possible to reduce this bias and, therefore, improve search efficacy through novel uses of short-term and long-term memory. Further, it now appears feasible to delineate local search algorithms based on the strength and qualitative nature of their search bias, rather than on superficial algorithmic characteristics.

Our analysis of random walk behavior indicates that the bias away from optimal solutions is due in large part to the representation used to encode solutions, i.e., the binary hypercube. The relationship between representation and search efficiency is, in general, poorly understood. However, our results suggest that the most effective representations are those that minimize the bias away from optimal solutions, leading to two key open research questions: (1) Do different representations induce either qualitatively or quantitatively different biases? and (2) How do we design representations to minimize search bias? The identification of the unexpectedly strong relationship between representation and problem difficulty also provides a metric by which different representations can be judged and should ultimately lead to the design of more efficient representations.

From the standpoint of accounting for variability in problem difficulty, the cost models developed in this thesis are clearly descriptive. The reason is straightforward: the information upon which the models are based, e.g., the set of optimal solutions to a problem instance, is intractable to obtain in practice. However, by clearly identifying those factors responsible for problem difficulty, our models provide a basis from which researchers can begin to address the question of whether it is possible to predict problem difficulty, and if so, to characterize the trade-off in accuracy and computational effort.

Moving beyond the JSP, there is the obvious issue of generalization: Do similar cost models and the associated insights hold for other local search algorithms and other combinatorial optimization problems? The mechanics of testing the generalizability of our models is clearly straightforward, if not tedious. However, such experimentation should provide key insights into the behavior of local search algorithms for other well-known NP-hard problems and lay the groundwork for a more general theory of local search. Further, because local search algorithms for other NP-hard problems often employ substantially different representations than those found in local search algorithms for the JSP, such studies should also provide insight into the relationship between representation and search efficiency.

Finally, significant questions remain as to the potential impact of problem structure on the qualitative nature and overall accuracy of our cost models. Initially developed using random JSPs, our models do generalize to specific types of structured JSPs. Even these relatively minor structural changes induced qualitatively different transition probabilities than those observed for random JSPs, although the overall model structure and accuracy were not significantly impacted. However, real-world problems exhibit a much richer range of structural characteristics than the problems we considered. Consequently, it seems at least possible, if not likely, that some real-world problems may induce radically different run-time dynamics for local search and, by inference, qualitatively different cost models.

12.5 Final Thoughts

Despite the high level of research activity in local search over the last decade, comparatively little progress has been made in the theoretical foundations of the field. Most research focuses either on the application of existing algorithms to new problems or the development of new algorithms. Ideas and techniques are routinely re-introduced and re-invented, and it is often difficult to assess the novelty and/or contribution of new research. Recently, particularly within the Artificial Intelligence community, the roots of a theory of local search have begun to emerge. The models developed in this thesis build on this initial research, culminating in a major advance toward a more general theory of local search. Specifically, we now better understand the mechanisms underlying these algorithms and how these mechanisms give rise to various observed phenomena. Model generalization to both other problems and a wider range of local search algorithms is

a significant outstanding challenge. Similarly, the implications of these models for algorithm design are largely unknown. Even with inefficient and ad-hoc development methodologies, researchers have continued to make significant advances in the effectiveness of local search algorithms. By developing a generalized theory of local search, it should be possible to more precisely focus future algorithmic research and, as a consequence, significantly accelerate the rate of advances in the field.

REFERENCES

[ABZ88] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job-shop scheduling. *Management Science*, 34(3):391–401, 1988.

[AC91] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[AGKS00] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In Kenneth Ford, editor, *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pages 256–261. AAAI/MIT Press, 2000.

[AK89] E.H.L. Aarts and J.H.M. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, 1989.

[AvL85] E.H.L. Aarts and P.J.M. van Laarhoven. A new polynomial-time cooling schedule. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 206–208, 1985.

[AvLLU94] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, and N.L.J. Ulder. A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, 6(2):118–125, 1994.

[Bau86] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Unpublished Manuscript, 1986.

[BC95] J.W. Barnes and J.B. Chambers. Solving the job shop scheduling problem with tabu search. *IIE Transactions*, 27:257–263, 1995.

[BDP96] J. Blażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33, 1996.

[Bea90] J.E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[BF00] J.C. Beck and M.S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117:31–81, 2000.

[BGS97] J.R. Beveridge, C.R. Graves, and J. Steinborn. Comparing random starts local search with key feature matching. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1997.

[BK94] K.D. Boese and A.B. Kahng. Best-so-far vs. where-you-are: Implications for optimal finite-time annealing. *Systems and Control Letters*, 22(1):71–80, January 1994.

[Bru01] P. Brucker. *Scheduling Algorithms*. Springer Verlag, 3rd edition, 2001.

[BSE+96] J. Blazewicz, G. Schmidt, K.H. Ecker, E. Pesch, and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer Verlag, 1996.

[BT94] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):128–140, 1994.

[BV95] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job-shop scheduling. Technical Report MSRR-609, Department of Management Science, Carnegie Mellon University, 1995.

[BV98] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job-shop scheduling. *Management Science*, 44(2):262–275, 1998.

[Car82] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 26:42–47, 1982.

[CB96] J.B. Chambers and J.W. Barnes. New tabu search results for the job shop scheduling problem. Technical Report ORP96-10, Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, 1996.

[CFG+96] D.A. Clark, J. Frank, I.P. Gent, E. MacIntyre, N. Tomov, and T. Walsh. Local search and the number of solutions. In Eugene C. Freuder, editor, *Proceedings of the Second International Conference on Principles and Practices of Constraint Programming (CP-96)*, pages 119–133. Springer-Verlag, 1996.

[CKT91] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the *Really* hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[Coh95] P.R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, 1995.

[Cě85] V. Cěrny. A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[DMU98] E. Demirkol, S.V. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109:137–141, 1998.

[DPS92] R.A. Dudek, S.S. Panwalker, and M.L. Smith. The lessons of flowshop scheduling research. *Operations Research*, 40(1):7–13, 1992.

[DT93] M. Dell'Amico and M. Trubian. Applying tabu-search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[Fan94] H. Fang. *Genetic Algorithms in Timetabling and Scheduling*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1994.

[FCS97] J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[Fel68] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley and Sons, third edition, 1968.

[FF94] C. Fleurent and J.A. Ferland. Genetic hybrids for the quadratic assignment problem. *DIMACS Series in Mathematics and Theoretical Computer Science*, 16:173–187, 1994.

[Fre82] S. French. *Sequencing and Scheduling – An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, John-Wiley & Sons, 1982.

[FT63] H. Fisher and G.L. Thompson. *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*, chapter 15, pages 225–251. Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

[FT88] S. Y. Foo and Y. Takefuji. Stochastic neural networks for solving job-shop scheduling: Part 1. problem representation. In Bart Kosko, editor, *IEEE International Conference on Neural Networks*, volume 2, pages 275–282, 1988.

[Gan19] H. L. Gantt. Efficiency and democracy. In *Transactions of the American Society of Mechanical Engineers*, volume 40, pages 799–808, 1919.

[GJ79] M.S. Garey and D.S. Johnson. *Computers And Intractability: A Guide To The Theory Of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GJS76] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[GL97] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5:533–549, 1986.

[Glo89] F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glo90] F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[GSK98] C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437, 1998.

[GT60] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[GW94] I.P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1–2):335–345, 1994.

[HG95] W.D. Harvey and M.L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[HHW96] T. Hogg, B.A. Huberman, and C.P. Williams. Special issue on frontiers in problem soling: Phase transitions and complexity. *Artificial Intelligence*, 81(1–2), 1996.

[Hoo94] J.N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42:201–212, 1994.

[Hoo95] J.N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[Hoo98] H.H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, Darmstadt University of Technology, 1998.

[Hoo02] H.H. Hoos. A mixture-model for the behaviour of sls algorithms for SAT. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 661–667. AAAI Press/The MIT Press, 2002.

[IR92] L. Ingber and B. E. Rosen. Genetic algorithms and very fast simulated reannealing: A comparison. *Mathematical and Computer Modeling*, 16(11):87–100, 1992.

[Jai98] A.S. Jain. *A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem*. PhD thesis, University of Dundee, 1998.

[JAMS89] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; Part 1, graph partitioning. *Operations Research*, 37(6):865–891, 1989.

[JAMS91] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; Part 2, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[JF95] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty. In L.J. Eschelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.

[JM97] D.S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In *Local Search in Optimization*, pages 215–310. John Wiley and Sons, 1997.

[JM99] A.S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present, and future. *European Journal of Operational Research*, 113:390–434, 1999.

[Jon95] T. Jones. *Evolutionary Algorithms, Fitness Landscapes, and Search*. PhD thesis, Department of Computer Science, University of New Mexico, 1995.

[JR93] T. Jones and G. Rawlins. Reverse hillclimbing, genetic algorithms and the busy beaver problem. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 70–75. Morgan Kaufmann, 1993.

[JRM00] A.S. Jain, B. Rangaswamy, and S. Meeran. New and ”stronger” job-shop neighborhoods: A focus on the method of Nowicki and Smutnicki(1996). *Journal of Heuristics*, 6:457–480, 2000.

[Kau93] S.A. Kauffman. *The Origins of Order*. Oxford University Press, 1993.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Kol99] M. Kolonko. Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 113:123–136, 1999.

[KS94] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.

[KT85] S. Kirkpatrick and G. Toulouse. Configuration space analysis of traveling salesman problems. *Journal de Physique*, 46:1277–1292, 1985.

[Law84] S.R. Lawrence. Resource-constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[LB00] D.P. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 2000.

[Lho93] O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings of Thirteen International Joint Conference on Artificial Intelligence (IJCAI-93)*, volume 1, pages 232–238, 1993.

[LK73] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[LMS03] H.R. Lourenço, O. Martin, and T. Stuützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic, 2003.

[Lou93] H.R. Lourenço. *A Computational Study of the Job-Shop and the Flow-Shop Scheduling Problems*. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, 1993.

[Lou95] H.R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.

[LZ96] H. R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu search: Application to the job-shop scheduling problem. In I. H. Osman and J. P. Kelley, editors, *Meta-Heuristics: Theory and Applications*, pages 219–236. Kluwer Academic Publishers, 1996.

[MÖ90] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In *Foundations of Genetic Algorithms - 1*, pages 316–337. Morgan Kaufmann, 1990.

[Mat96] D.C. Mattfeld. *Evolutionary Search and the Job Shop*. Physica-Verlag, Heidelberg, 1996.

[MBK99] D.C. Mattfeld, C. Bierwirth, and H. Kopfer. A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, 86:441–453, 199.

[MGSK88] H. Mühlenbein, M. Georges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[Mit98] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[MOF91] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[MP86] M. Mezard and G. Parisi. A replica analysis of the traveling salesman problem. *Journal de Physique*, 47:1285–1296, 1986.

[MRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[MSB88] K.N. McKay, F.R. Safayeni, and J.A. Buzacott. Job-shop scheduling theory: What is relevant? *Interfaces*, 18(4):84–90, 1988.

[MSS88] H. Matsuo, C.J. Suh, and R.S. Sullivan. A controlled search simulated annealing method for the general job-shop scheduling problem. Working Paper 03-04-88, Graduate School of Business, The University of Texas at Austin, Austin, Texas, USA, 1988.

[MZK+98] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity for characteristic 'phase transitions'. *Nature*, 400:133–137, 1998.

[NS96] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[NS01] E. Nowicki and C. Smutnicki. Some new ideas in TS for job-shop scheduling. Technical Report 50/01, Institute of Engineering Cybernetics, Wroclaw University of Technology, Poland, 2001.

[NS02] E. Nowicki and C. Smutnicki. Some new tools to solve the job shop problem. Technical Report 60/02, Institute of Engineering Cybernetics, Wroclaw University of Technology, Poland, 2002.

[NS03] E. Nowicki and C. Smutnicki. New algorithm for the job shop problem. Technical report, Institute of Engineering Cybernetics, Wroclaw University of Technology, Poland, 2003.

[Nui94] W.P.M. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.

[Ott93] R.L. Ott. *An Introduction to Statistical Methods and Data Analysis*. Duxbury Press, Belmont, California, 1993.

[Par97] A.J. Parkes. Clustering at the phase transition. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 340–345. AAAI/MIT Press, 1997.

[PDS73] S.S. Panwalker, R.A. Dudek, and M.L. Smith. Sequencing research and the industrial scheduling problem. In *Proceedings of Symposium on Theory of Scheduling and its Applications*, pages 29–38. Springer-Verlag, New York, 1973.

[Pin01] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2nd edition, 2001.

[PM00] F. Pezzella and E. Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120:297–310, 2000.

[Pro94] P. Prosser. Binary constraint satisfaction problems: Some are harder than others. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 95–99, 1994.

[PSW91] C.N. Potts, D.B. Shmoys, and D.P. Williamson. Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10:281–284, 1991.

[Ree98] C.R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1998.

[RS01] C.M. Riedys and P.F. Stadler. Combinatorial landscapes. Technical Report 01-03-014, The Santa Fe Institute, 2001.

[RW97] S. Rana and L.D. Whitley. Representation, search, and genetic algorithms. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*. AAAI Press/MIT Press, 1997.

[RY98] C.R. Reeves and T. Yamada. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6:45–60, 1998.

[SC96] R. Schrag and J. M. Crawford. Implicates and prime implications in random 3SAT. *Artificial Intelligence*, 88:199–222, 1996.

[SFM+96] J. Schneider, C. Froschhammer, I. Morgernstern, T. Husslein, and J.M. Singer. Searching for backbones - an efficient parallel algorithms for the traveling salesman problem. *Computational Physics Communications*, 96:173–188, 1996.

[SG95] B. Smith and S. Grant. Sparse constraint graphs and exceptionally hard problems. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[SGS00] J. Singer, I.P. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

[Sha48] C.E. Shannon. A mathematical theory of communication. *Bell Systems Tech. Journal*, 27:379–423,623–656, 1948.

[Sin00] J. Singer. *Why solutions can be hard to find: A featural theory of cost for a local search algorithm on random satisfiability*. PhD thesis, University of Edinburgh, 2000.

[SM90] R.L. Scheaffer and J.T. McClave. *Probability and Statistics for Engineers*. PWS-Kent Publishing Company, Boston, 3rd edition, 1990.

[Sou86] N. Sourlas. Statistical mechanics and the traveling salesman problem. *Europhysics Letters*, 2:919–923, 1986.

[SSF02] P. Salamon, P. Sibani, and R. Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*. Society for Industrial and Application Mathematics, 2002.

[Sta96] P.F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996.

[Stu99] T. Stutzle. *Local Search Algorithms for Combinatorial Problems – Analysis, Improvements, and New Applications*. PhD thesis, Darmstadt University of Technology, 1999.

[Stu01] T. Stuützle. Personal communication, 2001.

[SW01] J. Slaney and T. Walsh. Backbones in optimization and approximation. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 254–259. Morgan Kaufmann, 2001.

[SWV92] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38:1495–1509, 1992.

[Tai89] E.D. Taillard. Parallel taboo search technique for the jobshop scheduling problem. Technical Report ORWP 89/11, DMA, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1989.

[Tai93] E.D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[Tai94] E.D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[Vae95] R.J.M. Vaessens. *Generalized Job Shop Scheduling: Complexity and Local Search*. PhD thesis, Eindhoven University of Technology, 1995.

[VAL96] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996.

[vLA88] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer, 1988.

[vLAL88] P.J.M van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. Technical Report OS-R8809, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1988.

[vLAL92] P.J.M van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[WBHW01] J.P. Watson, J.C. Beck, A.E. Howe, and L.D. Whitley. Toward an understanding of local search cost in job-shop scheduling. In Amadeo Cesta, editor, *Proceedings of the Sixth European Conference on Planning*. Springer-Verlag, 2001.

[WBWH99] J.P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Algorithm performance and problem structure for flow-shop scheduling. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[WBWH02] J.P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123, 2002.

[Wei89] E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the differences. *Biological Cybernetics*, 63:325–336, 1989.

[WHH+97] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevast'yanov, and D.B. Shmoys. Short shop schedules. *Operations Research*, 45:288–294, 1997.

[Wri32] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In D.F. Jones, editor, *International Proceedings of the Sixth International Congress on Genetics*, volume 1, pages 356–366, 1932.

[WS99] D.J. Wales and H.A. Scheraga. Global optimization of clusters, crystals, and biomolecules. *Science*, 285:1368–1372, 1999.

[WW95] F. Werner and A. Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58:191–211, 1995.

[YN95] T. Yamada and R. Nakano. Job-shop scheduling by simulated annealing combined with deterministic local search. In *Proceedings of the First Meta-Heuristics International Conference (MIC'95)*, pages 344–349, 1995. Breckenridge, Colorado, USA.

[Yok97] M. Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Proceedings of the Third International Conference on the Principles and Practice of Constraint Programming (CP-97)*, pages 356–370. Springer-Verlag, 1997.

[YRN94] T. Yamada, B.E. Rosen, and R. Nakano. A simulated annealing approach to job-shop scheduling using critical block transition operators. In *IEEE International Conference on Neural Networks (ICNN '94)*, pages 4687–4692, 1994. Orlando, Florida, USA.