Contrasting Structured and Random Permutation Flow-Shop Scheduling Problems: Search-Space Topology and Algorithm Performance

Jean-Paul Watson • Laura Barbulescu • L. Darrell Whitley • Adele E. Howe

Department of Computer Science, Colorado State University, Fort Collins, Colorado 80523, USA

watsonj@cs.colostate.edu • laura@cs.colostate.edu • whitley@cs.colostate.edu • howe@cs.colostate.edu

---

The use of random test problems to evaluate algorithm performance raises an important, and generally unanswered, question: Are the results generalizable to more realistic problems? Researchers generally assume that algorithms with superior performance on difficult, random test problems will also perform well on more realistic, structured problems. Our research explores this assumption for the permutation flow-shop scheduling problem. We introduce a method for generating structured flow-shop problems, which are modeled after features found in some real-world manufacturing environments. We perform experiments that indicate significant differences exist between the search-space topologies of random and structured flow-shop problems, and demonstrate that these differences *can* affect the performance of certain algorithms. Yet despite these differences, and in contrast to difficult random problems, the majority of structured flow-shop problems were easily solved to optimality by most algorithms. For the problems not optimally solved, differences in performance were minor. We conclude that more realistic, structured permutation flow-shop problems are actually relatively easy to solve. Our results also raise doubts as to whether superior performance on difficult random scheduling problems translates into superior performance on more realistic kinds of scheduling problems.

(*Flow Shop; Sequencing; Local Search; Problem Difficulty; Fitness Landscapes*)

---

1. Introduction

To compare the performance of two algorithms, researchers typically run both algorithms on a set of test problems, record both the best solutions found and the computational costs

required to attain them, and analyze the results. Test problems are generally selected from well-known and widely-used benchmark suites, such as those available from the OR Library (http://www.ms.ic.ac.uk/info.html). Most often, benchmark suites contain synthetic test problems that are generated by randomly selecting values for problem features. For example, typical test problems for both flow-shop and job-shop scheduling are generated by sampling operation processing times from a single uniform distribution. Test problems are classified as difficult if state-of-the-art algorithms fail to find good solutions consistently, measured relative to either a lower bound or a best-known solution, in reasonable running times.

One widely acknowledged problem with using benchmarks to evaluate algorithm performance is that researchers run the risk of over-fitting their algorithms to the kinds of test problems found in benchmark suites. One way to avert this risk is to develop benchmarks containing a diverse array of test problems. This approach is exemplified in the benchmark suite for the Quadratic Assignment Problem (QAP), which consists of a variety of random, structured, and real-world test problems; here, both random and structured problems are synthetic, but the features of the structured test problems are modeled after features found in real-world problems (Taillard 1995). In addition to mitigating the risk of over-fitting, a diverse benchmark suite also enables researchers to investigate the relationship between problem structure and algorithm performance. For example, algorithm performance in the QAP depends heavily on the type of test problem considered, with no single algorithm providing superior performance over the entire benchmark suite (Taillard 1995). Further, real-world instances of the QAP are generally much easier to solve than either random or structured test problems.

Yet, diverse benchmarks are not available for many important optimization problems. For example, structured test problems for flow-shop and job-shop scheduling are extremely rare, and those few that exist are rarely considered in comparative studies of algorithm performance. This is despite the fact that both flow-shop and job-shop scheduling are simple abstractions of problems found in some real-world scheduling environments, a situation that should promote the development of structured test problems. For scheduling, the lack of diverse and realistic test problems is a particularly severe shortcoming, as there is an assumption, although usually implicit, underlying most comparative studies of scheduling algorithms: if an algorithm performs well on difficult, random test problems, then it will also perform well on more realistic problems. Given the lack of structured (or real-world) scheduling benchmarks, we are currently unable to test this hypothesis explicitly. Further,

similar questions such as "Can one algorithm provide superior performance on different kinds of problems?" remain unanswered.

The availability of structured scheduling benchmarks would enable us to answer these questions by simply analyzing differences in algorithm performance. However, this approach would provide little understanding as to *why* one particular set of observations occurred, instead of another. Hooker (1995) argues that this dichotomy is inherent in the benchmark-driven approach to algorithm development, in which more effort is expended on the algorithmic fine-tuning required to achieve competitive results than on performing the type of hypothesis-driven research required to advance our understanding of algorithm behavior. In relation to scheduling, this criticism is well-deserved: researchers have developed many effective, high-performance scheduling algorithms, but a deep understanding of *why* these algorithms work well remains elusive.

We consider the aforementioned issues in the context of the well-known permutation flow-shop scheduling problem (PFSP). We first introduce a problem generator for creating structured PFSPs. Our generator produces test problems with three different types of structure, and allows us to control for the expected degree of randomness in the problem features. Given this problem generator, we then explore how both search-space topology and algorithm performance change as we transition from random PFSPs to more structured PFSPs.

All state-of-the-art algorithms for the PFSP, in terms of both solution quality and execution speed, are based on local search. Consequently, we analyze search-space topology from the perspective of two well-known local search move operators for the PFSP. The performance of these move operators depends heavily on their ability to re-order the jobs on the critical path. Our experiments identify substantial differences in the critical path topologies of random and structured PFSPs, and we show that these differences impact the performance of both move operators, albeit for different reasons. The distribution of local optima is also known to influence the performance of local search algorithms, and we further demonstrate significant differences between the local optima distributions of random and structured PFSPs.

Given differences in the search-space topologies of random and structured PFSPs, we anticipated differences in algorithm performance as well (e.g., as was found in the QAP). To test this hypothesis, we ran both state-of-the-art local search and simple heuristic constructive algorithms on a wide range of random and structured PFSPs. While differences

in search-space topology often translated into differences in algorithm performance, we observed a more important and unexpected result: the majority of structured PFSPs were easily solved to optimality by most algorithms. For the problems not solved to optimality, any differences in relative algorithm performance were minor. In other words, structured PFSPs tend to be very easy, in that high-quality (and in many cases optimal) solutions can be easily found by both simple and complex algorithms. This is in stark contrast to random PFSPs, for which there are significant performance differences among even state-of-the-art algorithms.

2. Generating Structured Test Problems for the PFSP

We consider the n by m permutation flow-shop scheduling problem (PFSP). Here, each of the n jobs is processed on each of the m machines in sequential order (starting at machine 1 and finishing at machine m), such that machine $j+1$ cannot begin processing a job until machine j has completed processing of the same job. Concurrency and pre-emption are not allowed: a machine can only process a single job at a time, and processing must be allowed to complete once initiated. An operation o_{ij} denotes the processing of job i on machine j, and requires a processing duration of d_{ij}. A solution to the PFSP specifies a common job processing order π for all of the machines, with the ith job in π denoted by $\pi(i)$. A solution π implicitly specifies an earliest completion time ect_{ij} for each operation o_{ij} (Nowicki and Smutnicki 1996). The makespan of π, denoted $C_{max}(\pi)$, is defined to be the earliest completion time of job $\pi(n)$ on machine m: e.g., $ect_{\pi(n)m}$. The objective is to find a solution π such that $C_{max}(\pi)$ is minimized. For $m \geq 3$, the PFSP with the makespan minimization objective is NP-complete (Garey et al. 1976).

The most widely-used benchmark suite for the PFSP was introduced by Taillard (1993) and is available from the OR Library. Taillard's benchmark plays a central role in the PFSP literature, as nearly all state-of-the-art PFSP algorithms (e.g., Nowicki and Smutnicki's 1996 tabu search algorithm and Reeves and Yamada's 1998 path relinking algorithm) are judged *strictly* on how well they perform on this benchmark. The test problems in Taillard's benchmark suite were generated by sampling the operation durations d_{ij} uniformly from the interval $[1, 99]$ and selecting a subset of problems using several criteria, including problem difficulty as measured by the variance in solution quality over multiple runs of a tabu search algorithm. Because the d_{ij} are independent, Taillard's test problems are, in expectation,

completely unstructured. Consequently, we refer to PFSPs generated in this manner as *random* PFSPs. We refer to PFSPs with non-uniform d_{ij} as *structured* PFSPs.

Rinnooy Kan (1976) introduced methods for creating two types of structure in synthetic PFSPs: job correlation and time gradients. In job-correlated PFSPs, a small number ($< n$) of non-overlapping intervals are defined, and all processing times for a given job are uniformly sampled from one of these intervals. In contrast, PFSPs with time gradients exhibit a trend in processing times as a function of machine index; in problems with positive (negative) time gradients, jobs require less (more) time on early machines than on later machines. Rinnooy Kan developed these structured PFSPs to complement some random PFSPs that were generated using a procedure similar to Taillard's, and studied the aggregate in the context of branch-and-bound algorithms for the PFSP. For each problem type (random, job-correlated, or with time gradients), the number of PFSPs that were solved (in that optimality was *proved*) was recorded. Rinnooy Kan found that while PFSPs with positive time gradients were easily solved, PFSPs with either negative time gradients or job-correlation were actually *more* difficult to solve than random PFSPs.

Reeves (1995) used Rinnooy Kan's structured PFSPs to compare the performance of a genetic algorithm, simulated annealing, and simple neighborhood search. Reeves reported that PFSPs with positive time gradients were particularly easy to solve, by observing that the makespan of solutions produced by the genetic algorithm were always equivalent to a computed lower bound. For job-correlated PFSPs, Reeves reported that simulated annealing and the genetic algorithm often produced solutions with equivalent makespans, a result he interpreted as suggesting an overall smoothness in the search landscape. Optimality was not established (the makespans obtained were different from computed lower bounds), so no claim was made regarding the difficulty of job-correlated PFSPs.

Some surveys of real-world manufacturing facilities (e.g., Panwalker et al. 1973) suggest that a mixture of job-correlation and machine-correlation (in the latter, the processing times of all jobs on a particular machine are sampled from a distribution specific to that machine) is often encountered in practice. These mixed-correlated problems are similar to machine-correlated problems, with the exception that the relative ranks of job processing times are generally consistent across the machines; for example, if a job has the largest processing time on machine 1, then it is likely to have the largest processing time on machines 2 through m. In Figure 1, we provide illustrative examples for each of the three types of correlation. In job-correlated PFSPs, job processing times are largely independent of the machine index.

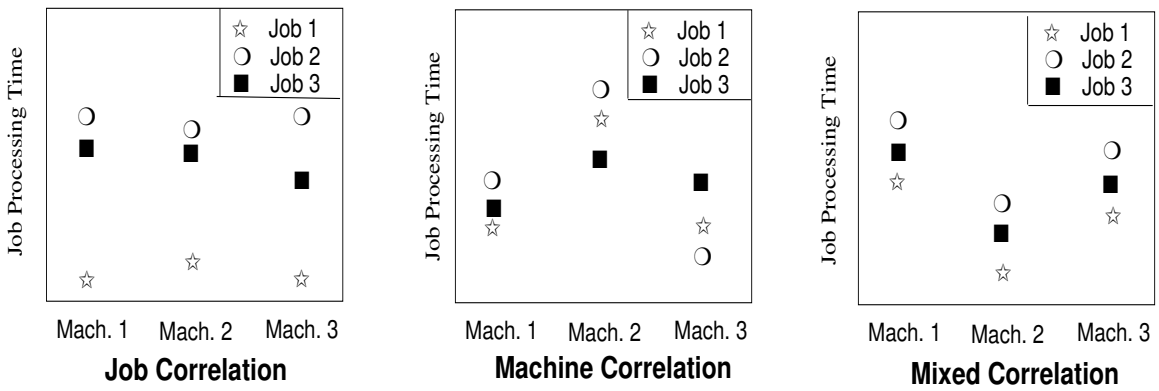

Figure 1: Examples of 3×3 PFSPs With Three Different Types of Structure

In machine-correlated PFSPs, job processing times are almost exclusively dependent on the machine index. Mixed-correlated PFSPs are a form of machine-correlated PFSP in which the relative *ranks* of job processing times are largely independent of machine index.

We now introduce a method for generating structured PFSPs with each of the three forms of correlation. In doing so, we are specifically *not* claiming that our structured PFSPs are directly modeled after real-world flow-shops. Indeed, Dudek et al. (1992) have noted that the static, deterministic, makespan-minimization form of the PFSP is rarely, if ever, encountered in industry. Rather, we are examining the influence of problem structure in the context of a very basic scheduling problem (the PFSP) by considering various forms of structure found in more general industrial scheduling problems (Panwalker et al. 1973).

Producing PFSPs with each of the forms of correlation shown in Figure 1 is conceptually simple; in each case, the processing times are uniformly sampled from one of a number of distributions. For job-correlated PFSPs, we use n distributions, one for each job. For machine-correlated and mixed-correlated PFSPs, we use m distributions, one for each machine. If processing times are restricted to a fixed-width interval I (e.g., $I = [1, 99]$), then the distribution means can be sampled from this interval, with the variances subject to the boundary conditions imposed by the endpoints of I.

We are interested in studying not only the differences in search-space topology and algorithm performance between structured and random PFSPs, but the nature of those differences as we *transition* from purely random PFSPs to highly structured PFSPs. If the distribution means of our structured PFSPs are very different, then we expect relatively few interacting bottleneck elements (jobs or machines). In contrast, if the distribution means are very similar (as they are in random PFSPs), then we can expect to have a large number

of interacting bottleneck elements. Using the intuition that the number of interacting bottleneck elements influences problem difficulty, we control for the expected problem difficulty by randomly sampling the distribution means from a fraction α of I, $0 \leq \alpha \leq 1$. By varying α from 0.0 to 1.0, we can then gradually transition from random PFSPs to more structured PFSPs.

This approach is slightly different from a method we previously reported for generating the distribution means (Watson et al. 1999). There, we controlled for problem difficulty by varying the degree to which the various distributions overlapped. However, for problems with large numbers of jobs or machines, maintaining separated distributions resulted in unrealistically large d_{ij}. Follow-up experiments indicated that problem difficulty was influenced primarily by the number of interacting bottleneck distributions, leading to our current approach for sampling distribution means.

2.1 Problem Generator Description

In the experiments discussed in Sections 3 through 5, we analyze three types of structured PFSPs: job-correlated, machine-correlated, and mixed-correlated. We did not consider structured PFSPs with time gradients due to the experimental results of Reeves (1995). Before providing pseudo-code for our problem generator, we first define the generator parameters and specify the parameter settings used to produce all of our test problems.

All operation durations d_{ij} are restricted to the interval $I = [Dur_{LB}, Dur_{UB}]$. Following Taillard (1993), we let $Dur_{LB} = 1$ and $Dur_{UB} = 99$. For each of the x distributions (either $x = n$ or $x = m$, depending on the type of correlation), a distribution half-width d_i^{hw}, $1 \leq i \leq x$, is uniformly sampled from the interval $[HW_{LB}, HW_{UB}]$; here, we let $HW_{LB} = 1$ and $HW_{UB} = 5$. In correlated problems, distributions are often pair-wise non-overlapping (i.e., consider a very large and a very small job in a job-correlated problem); large values of HW_{LB} and HW_{UB}, relative to $Dur_{UB} - Dur_{LB}$, would prevent us from generating this type of problem. Finally, as discussed above, $\alpha \in [0, 1]$ dictates the proportion of the interval I from which the distribution means μ_i, $1 \leq i \leq x$, are sampled.

Given this set of parameters, we generate structured PFSPs using the following three-step process. All variables are integers unless otherwise noted, and the function $rint(x)$ rounds the real number x to the nearest integer (if the fractional portion of x equals 0.5, *rint* returns the largest integer that is a lower bound of x).

Step 1: Determine Distribution Means

Let $Width_{eff} = rint(\alpha \cdot (Dur_{UB} - Dur_{LB}))$. Assign $Interval_{st}$ by drawing a uniform sample from the interval $[Dur_{LB}, Dur_{UB} - Width_{eff}]$. The distribution means μ_i, $1 \leq i \leq x$, are then uniformly sampled from $[Interval_{st}, Interval_{st} + Width_{eff}]$.

Step 2: Determine Distribution Widths

Sample the distribution half-widths d_i^{hw}, $1 \leq i \leq x$, uniformly from the interval $[HW_{LB}, HW_{UB}]$.

Step 3: Determine processing times d_{ij}

For job-correlated PFSPs, the d_{ij} for each job i are uniformly sampled from the interval $D_i = [d_i^{lb}, d_i^{ub}] = [\mu_i - d_i^{hw}, \mu_i + d_i^{hw}]$. Similarly, for machine-correlated PFSPs, the d_{ij} for each machine j are selected uniformly from the interval $D_j = [d_j^{lb}, d_j^{ub}] = [\mu_j - d_j^{hw}, \mu_j + d_j^{hw}]$.

Determining the d_{ij} for mixed-correlated PFSPs is slightly more complicated. First, the intervals D_j are computed as in the machine-correlated case. Then, for each job i, a real value $rank_i$ is uniformly sampled from the interval $[0, 1]$. A job i with $rank_i$ equal to 1.0 (0.0) biases samples (to produce the d_{ij}) toward the upper (lower) ends of the intervals D_j, $1 \leq j \leq m$, respectively. Letting $width(D_j) = d_j^{ub} - d_j^{lb}$, we then let $d_{ij} = rint(rank_i \cdot width(D_j)) + d_j^{lb}$. In reality, the relative ranks are not always identical across the machines. Thus, we introduce a noise factor η, which serves to modify the d_{ij} by adding a value uniformly sampled from the interval $[-\eta, \eta]$. In our experiments, we take $\eta = 2$.

All d_{ij} are post-processed to ensure that they lie within the interval I. If $d_{ij} < Dur_{LB}$, then $d_{ij} = Dur_{LB}$; similarly, if $d_{ij} > Dur_{UB}$, then $d_{ij} = Dur_{UB}$. When $\alpha = 0.0$ in the job-correlated and machine-correlated modes, we produce test problems that are in expectation identical to random PFSPs in which the d_{ij} are uniformly sampled from the interval $[45, 55]$; in the mixed-correlated mode, we must additionally increase the magnitude of the noise factor η to force a similar correspondence.

2.2 Structured and Random Test Problems

We consider four different sizes of PFSPs in our experiments: 20 machines with 20, 50, 100, and 200 jobs. The most difficult test problems in Taillard's benchmark suite possess these

dimensions; further, all of Taillard's test problems with $m < 20$ have been solved and are generally considered easy, and problems with $m > 20$ have not been previously considered in the PFSP literature. For each problem size, we generated 100 test problems for each type of correlation (job, machine, and mixed) and for each level of $\alpha \in \{0.0, 0.1, ..., 1.0\}$ (for a total of 33 structured problem groups at each problem size). Finally, for each problem size, we also generated two problem groups consisting of 100 random PFSPs apiece. In the first group, we follow Taillard and sample the d_{ij} from the interval $[1, 99]$, while in the second group we sample from the interval $[45, 55]$. As discussed earlier, the second group of random PFSPs closely corresponds to our structured PFSPs with $\alpha = 0.0$. The total number of test problems considered is $14,000$. In contrast to Taillard (1993), we do *not* filter for difficult problems. Instead, we are concerned with analyzing search-space structure and algorithm performance on various classes of test problem, and the analysis would be biased by any such filtering.

Any problem generator has certain biases in the types of problems it produces. For instance, our generator is unlikely to produce test problems with a bi-modal distribution of means, irregardless of α, while Taillard's generator is unlikely to produce any of our structured PFSPs with large α. As mentioned earlier, we hypothesize that the number of interacting bottleneck elements is the primary factor in determining the difficulty of structured PFSPs, and our problem generator was produced in part to specifically test that hypothesis.

While other researchers such as Rinnooy Kan and Reeves have introduced structured PFSPs, our structured problem generator provides two important contributions: 1) the ability to produce both machine-correlated and mixed-correlated PFSPs and 2) a mechanism for varying the expected number of bottleneck elements in structured PFSPs. Both the code for our structured PFSP generator (which is written in `C++`) and the test problems used in our experiments can be found at: http://www.cs.colostate.edu/sched.

3. Critical Path Topology and PFSP Move Operators

Since the early 1990s, the most effective algorithms for the PFSP, in terms of both execution speed and solution quality, have been based on local search. A key component of any local search algorithm is the move operator. "Knowledge-poor" move operators for the PFSP syntactically manipulate a solution π to produce a neighboring solution π'. The primary drawback to knowledge-poor move operators is that they typically generate a large number

of non-improving neighboring solutions. The most successful knowledge-poor PFSP move operator is the shift operator, denoted SH_{op}, and is defined as follows. Consider all pairs of job positions (x, y) in a solution π, subject to the restriction $y \neq x - 1$. Under SH_{op}, neighbors π' of π are produced by *shifting* the job at position x into the position y, while leaving all other relative job orders unchanged. If $x < y$, then $\pi' = (\pi(1), ..., \pi(x-1), \pi(x+1), ..., \pi(y), \pi(x), \pi(y+1), ..., \pi(n))$. If $x > y$, then $\pi' = (\pi(1), ..., \pi(y-1), \pi(x), \pi(y), ..., \pi(x-1), \pi(x+1), ..., \pi(n))$. Taillard (1990) reports SH_{op} as out-performing the *adjacent-swap* move operator (which swaps the $n-1$ pairs of adjacent jobs in π) in terms of both solution quality and computation time, and the *exchange* move operator (which exchanges the $n(n-1)/2$ distinct pairs of jobs in π) in terms of computation time.

In contrast to knowledge-poor PFSP move operators, "critical-path" PFSP move operators use problem-specific knowledge to generate only a subset of moves from π that can potentially lead to neighbors π' with $C_{max}(\pi') < C_{max}(\pi)$. The most widely studied critical-path PFSP move operator was introduced by Nowicki and Smutnicki (1996), which we denote NS_{op}. Most state-of-the-art PFSP algorithms are based on NS_{op}, including Nowicki and Smutnicki's (1996) own tabu search algorithm and Reeves and Yamada's (1998) path relinking algorithm. An exception is Stützle's (1999) iterated local search algorithm, which uses SH_{op} to achieve competitive results on Taillard's test problems. Together, SH_{op} and NS_{op} form the basis of all state-of-the-art PFSP algorithms. In the remainder of this section, we contrast the critical path topologies of random and structured PFSPs, and examine how the different topologies affect the performance of both SH_{op} and NS_{op}.

3.1 Characterizing Critical Path Topology

We now briefly review the concept of a critical path in the PFSP. A solution π implicitly specifies an earliest start time and earliest completion time for each operation o_{ij}, which we respectively denote est_{ij} and ect_{ij}. Recall that $C_{max}(\pi) = ect_{\pi(n)m}$. Next, we denote the amount of time from the earliest possible start time of o_{ij} to $ect_{\pi(n)m}$ by $tail_{ij}$. An operation o_{ij} is a *critical* operation if and only if $est_{ij} + tail_{ij} = C_{max}$: any delay in the start time of o_{ij} results in an increase of C_{max}. A *critical path* consists of a sequence of critical operations, starting with $o_{\pi(1)1}$ and ending with $o_{\pi(n)m}$. Note that a solution π may contain many critical paths.

Figure 2 shows a critical path extracted from a 10×10 random PFSP. Contiguous subsequences of critical operations of size ≥ 2 are present on machines 1, 3, 5, and 8. These

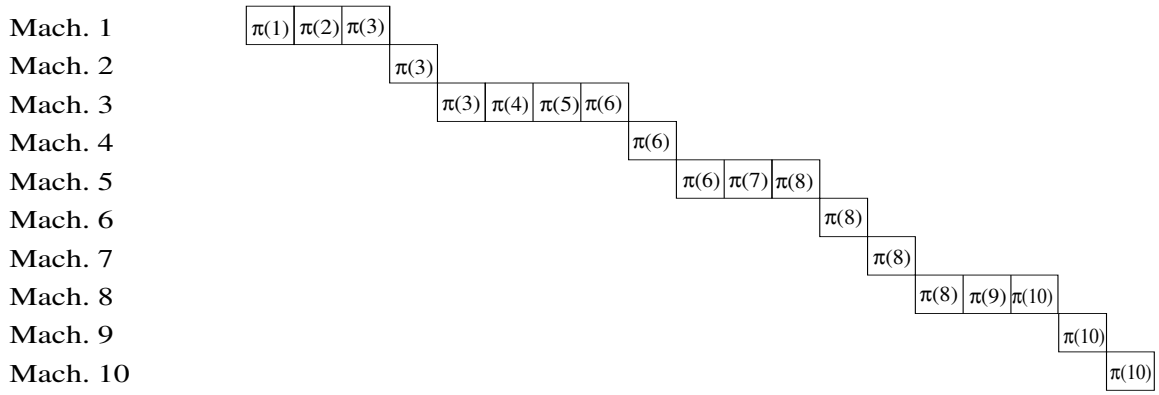

Figure 2: A Critical Path Extracted From a 10×10 Random PFSP, With Critical Blocks on Machines 1, 3, 5, and 8

sub-sequences are known as *critical blocks* (sub-sequences of size 1 are technically also critical blocks, but for reasons discussed later in this section, they are typically ignored). Critical blocks play an important role in the effectiveness of any move operator, because of the following (Nowicki and Smutnicki 1996):

Property 1 *Consider a critical path in a solution π with L critical blocks, and let X_i be the set of jobs internal (i.e., excluding the first and last jobs) to a critical block B_i, $1 \leq i \leq L$. If $i = 1$ and B_1 resides on machine 1, then $\pi(1)$ is additionally included in X_1. If $i = L$ and B_L resides on machine m, then $\pi(n)$ is additionally included in X_L. Then any re-ordering of the jobs in any X_i cannot reduce $\mathrm{C}_{\max}(\pi)$.*

For example, swapping the jobs in either positions 4 and 5 or 1 and 2 in Figure 2 cannot reduce C_{max}. The effectiveness of the shift, exchange, and adjacent-swap move operators is heavily influenced by the size of the blocks in the critical path, as Property 1 applies to a greater number of moves when large critical blocks are present. And although critical-path move operators such as NS_{op} explicitly avoid these moves, the total number of moves under these operators is still influenced by the critical path topology.

Despite the influence of critical path topology on move operator performance, no studies have attempted to characterize the critical path topologies of either random or structured PFSPs. For certain problem types, it is relatively easy to predict the expected topology in limiting cases. For example, the critical path topology of "prototypical" (as $\alpha \to \infty$) job-correlated and machine-correlated PFSPs are shown in Figures 3 and 4, respectively. In the prototypical job-correlated topology, there are two critical blocks, appearing on machines 1

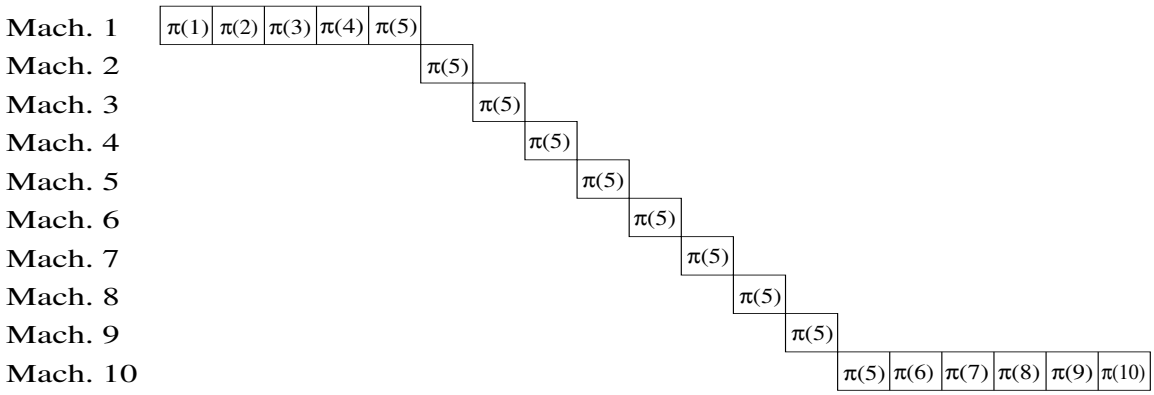

Figure 3: Prototypical Critical Path for a 10×10 Job-Correlated PFSP. The Bottleneck Job is $\pi(5)$.

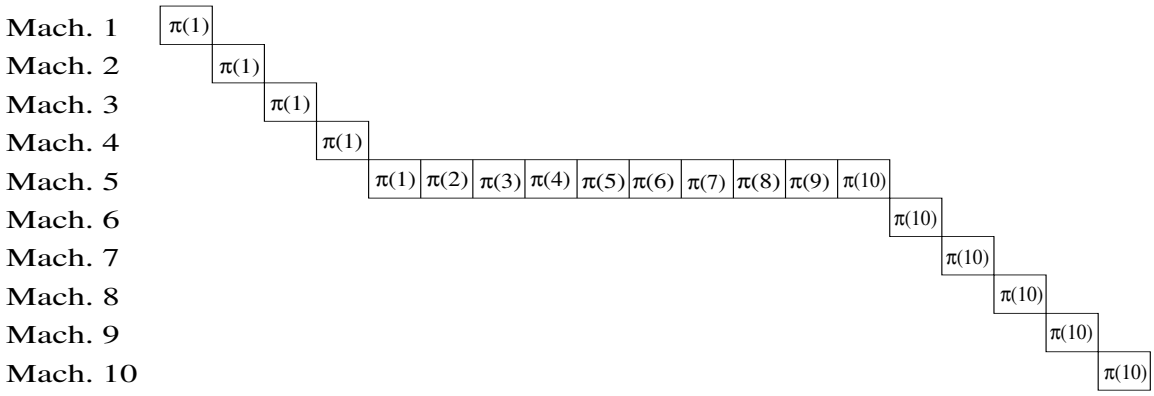

Figure 4: Prototypical Critical Path for a 10×10 Machine-Correlated PFSP. The Bottleneck Machine is 5.

and m, and a single bottleneck job ($\pi(5)$ in Figure 3). For modeling purposes, we assume that the two critical blocks contain $n/2$ and $n/2 + 1$ jobs, respectively, with a single shared job. In the prototypical machine-correlated topology, there is a single critical block (on $m = 5$ in Figure 4). For modeling purposes, we assume that the critical block does not reside on either machine 1 or machine m.

However, the expected form of a critical path is generally far from obvious, as is the case with random and mixed-correlated PFSPs, and for job-correlated and machine-correlated PFSPs with small values of α. Consequently, we turn to empirical methods to characterize the critical path topologies of our random and structured PFSPs, comparing the results to the prototypical topologies where available. For each of the problem groups defined in Section 2.2, we used independent runs of the *NEH-RS* algorithm (documented in Section 5.1) to produce 100 solutions to each test problem in the group, and for each solution we extracted a single (random) critical path and recorded the size of each critical block in the path.

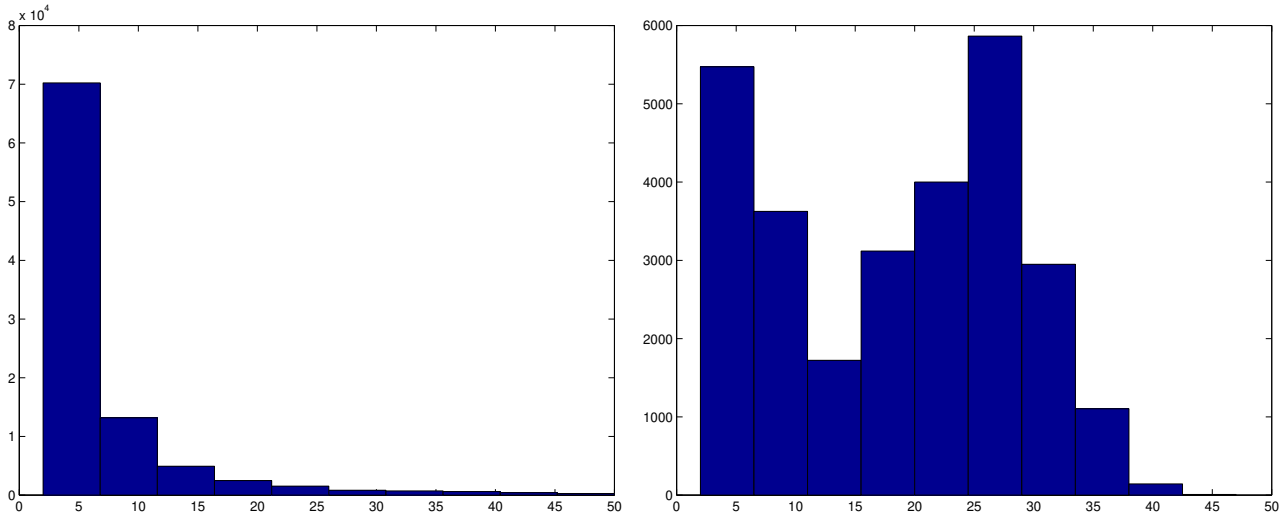

Figure 5: Histograms of Critical Block Sizes for 50×20 Random (Left Figure) and Job-Correlated (Right Figure) PFSPs. For the Job-Correlated PFSPs, $\alpha = 1.0$.

We used *NEH-RS* to generate representative solutions because 1) it produces solutions of reasonable quality in a short period of time and 2) we found little evidence that the critical path topology varies significantly with solution quality.

Selected histograms of the critical block sizes for the 50×20 problem groups are shown in Figures 5 and 6; the corresponding histograms for other problem sizes were very similar, except where noted below. Each histogram records the relative frequency of critical block sizes in 100 solutions to each of 100 test problems in the group. The critical paths of both random PFSP problem groups appear to be possess large numbers of small critical blocks (as shown in the left side of Figure 5), although the right-tail density is non-negligible and becomes more prevalent as problem size is increased. Upon closer examination, we found two dominant topologies: 1) a critical path strictly composed of very small critical blocks and 2) a critical path composed of a very large critical block, in addition to a few smaller ones.

The critical block histogram for job-correlated PFSPs at $\alpha = 0.1$ is similar to the histogram for random PFSPs. However, as α is increased, the histogram tends toward a bi-modal distribution, although the bi-modality did not appear until roughly $\alpha = 0.5$ for the 50×20 problems. We show the histogram for 50×20 job-correlated PFSPs with $\alpha = 1.0$ in the right side of Figure 5. Here, the right-most peak corresponds to critical block sizes of approximately $n/2 \approx 25$ for $n = 50$, as we would expect given our prototypical job-correlated critical path topology. However, there are large number of very small critical blocks (≤ 5),

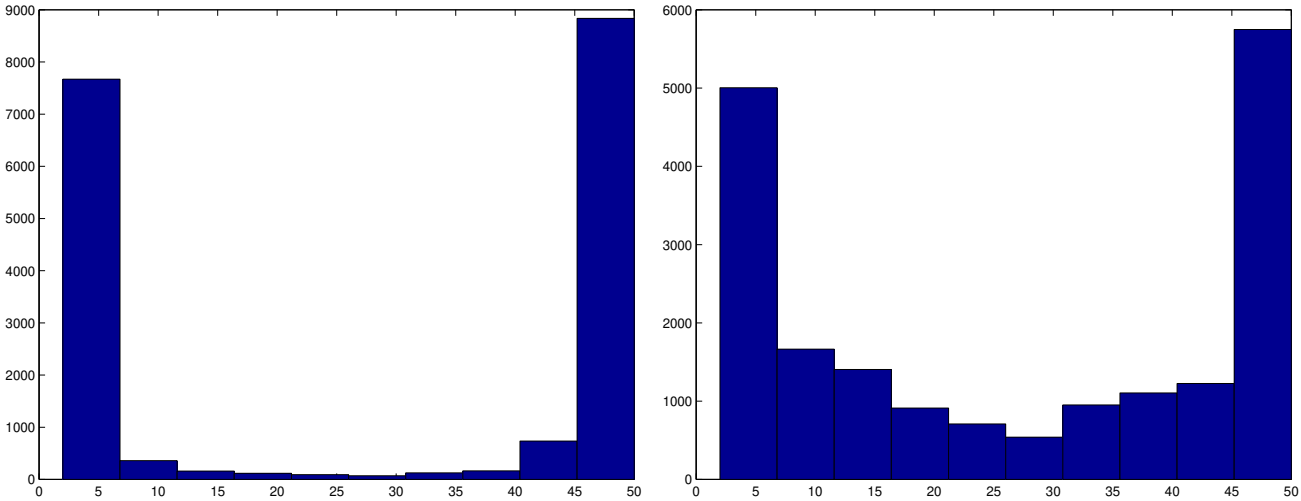

Figure 6: Histograms of Critical Block Sizes for 50×20 Machine-Correlated (Left Figure) and Mixed-Correlated (Right Figure) PFSPs. For Both Problem Groups, $\alpha = 0.3$.

in addition to a number of critical blocks of size 15 to 20 and 30 to 35. In contrast to our prototypical job-correlated critical path topology, the latter indicates that critical block sizes are rarely of equal size in our job-correlated PFSPs. The bi-modality for the 100×20 problems appears much later, at $\alpha = 0.9$, and never appears in the 200×20 problems (it appears at roughly $\alpha = 0.2$ for the 20×20 problems). The similarity between the distributions of critical block sizes in random and job-correlated PFSPs raises the possibility that both the local optima distributions (Section 4) and algorithm performance (Section 5) are similar for these two problem types.

In contrast to the results for job-correlated PFSPs, the critical path topologies of our 50×20 machine-correlated PFSPs (as shown in the left side of Figure 6 for $\alpha = 0.3$) are generally quite similar to the prototypical machine-correlated topology. The sole discrepancy is the large number of critical blocks of size ≤ 5, the proportion of which decreases (but never vanishes) as $\alpha \to 1.0$, indicating that the critical paths of machine-correlated PFSPs generally consist of a single dominant critical block in addition to a few very small critical blocks. At larger problem sizes, the bi-modality appears at $\alpha = 0.1$, and at all problem sizes the proportion of small critical blocks decays more rapidly as $\alpha \to 1.0$.

Finally, the critical path topologies of the 50×20 mixed-correlated PFSPs (as shown in the right side of Figure 6 for $\alpha = 0.3$) tend to mirror the prototypical machine-correlated topology. The influence of job-correlation in these problems is evident in increased numbers of critical blocks of size 5 through 45, although the frequency of these blocks decreases as

$\alpha \to 1.0$, and is negligible for larger problem sizes. As with machine-correlated PFSPs, the proportion of critical blocks of size ≤ 5 decreases as $\alpha \to 1.0$ (but again never vanishes). As was the case for random and job-correlated PFSPs, these observations suggest potential similarities in both local optima distributions (Section 4) and algorithm performance (Section 5) for machine-correlated and mixed-correlated PFSPs.

3.2 Critical Path Topology and SH_{op}

Given a PFSP with n jobs, the number of moves under SH_{op} is $(n-1)^2$, and is independent of the critical path topology. However, the proportion of these moves for which Property 1 is applicable does heavily depend on the critical path topology. We now examine the number of moves internal to a critical block for a solution π, which we denote $|CBINT(\pi)|$. These values are easily computed for both of our prototypical critical path topologies; for any critical block of size x, the number of non-improving internal moves is simply $(x-3)^2$. We also compute $|CBINT(\pi)|$ for a "uniform" prototypical critical path topology with b equal-sized critical blocks, in order to study the influence of the number of critical blocks on $|CBINT(\pi)|$. The $|CBINT(\pi)|$ for each of these topologies is given by:

$$\begin{array}{ll} b(n/b-3)^2 & \text{if uniform} \\ n^2/2-3n+5 & \text{if job-correlated} \\ (n-3)^2 & \text{if machine-correlated} \end{array}$$

We denote the fraction of total moves under SH_{op} that are internal to a critical block by f_{int}. In Figure 7, we plot f_{int} as a function of the total number of jobs n for each of our prototypical topologies ($m = 20$); for the uniform topology, we show the results for $b = 3$, $b = 4$, and $b = 5$. Of particular interest is the strong sensitivity of f_{int} to the number of critical blocks b. The implications are clear: if our prototypical critical path topologies approximately depict the critical path structure found in our structured PFSPs, then SH_{op} can waste anywhere between 45% and 95% of its time evaluating provably non-improving moves.

We now examine f_{int} for both our random and structured PFSPs, when possible comparing the results obtained for the corresponding prototypical critical path topology. For each of our problem groups, we used independent runs of *NEH-RS* to produce 100 solutions to each test problem in the group, and then computed f_{int} for each solution using a single random critical path. We computed the mean f_{int} for each problem group, denoted $\overline{f_{int}}$, by

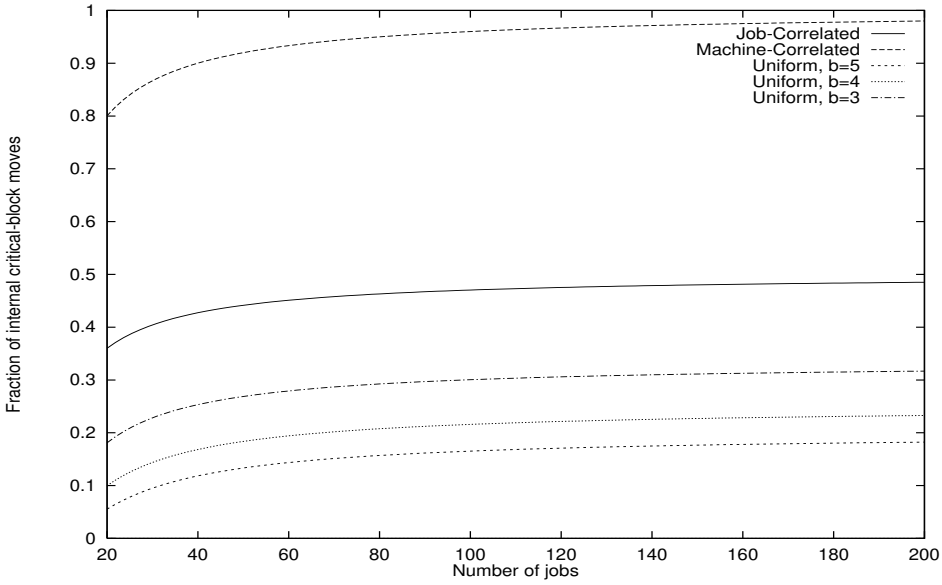

Figure 7: The Fraction of Moves Under SH_{op} That are Internal to a Critical Block for the Prototypical Job-Correlated, Machine-Correlated, and Uniform Critical Path Topologies

averaging over the 100 critical paths from each of the 100 test problems. The results are shown in Tables 1 and 2 for our random and structured PFSPs, respectively.

First, we consider the $\overline{f_{int}}$ observed for random PFSPs, shown in Table 1. Clearly, the width of the sampling interval has little influence on $\overline{f_{int}}$ for random PFSPs. Increases in $\overline{f_{int}}$ and n are correlated, which is consistent with the fact that the relative frequency of large critical blocks in random PFSPS grows $n \to \infty$ (the latter is also the cause of the large standard deviations). This correlation is also consistent with the general observation that local search algorithms based on SH_{op} tend to scale worse on random PFSPs than those based on NS_{op}; for $n \geq 200$, any algorithm using SH_{op} is likely to waste over 30% of its time evaluating provably non-improving moves.

Next, we examine the $\overline{f_{int}}$ for our structured PFSPs, shown in Table 2. The standard deviations (not reported) are roughly equivalent to $\overline{f_{int}}$ for small α, but decrease rapidly as $\alpha \to 1.0$. At $\alpha = 1.0$ the standard deviations for our 100×20 job-correlated, machine-correlated, and mixed-correlated problem groups are 0.09, 0.26, and 0.21, respectively. As expected, the $\overline{f_{int}}$ at $\alpha = 0.0$ are very similar to those of random PFSPs. For machine-correlated PFSPs, we often see a relatively strong agreement between the observed $\overline{f_{int}}$ and the value predicted by the prototypical topology; the agreement generally strengthens with increases in either α or n. Any discrepancy is primarily due to the existence of a number of very small critical blocks in addition to the large dominant critical block, as shown in the

left-hand side of Figure 6. In contrast, the $\overline{f_{int}}$ for job-correlated PFSPs were substantially less than those computed for the prototypical topology. Similarly, these discrepancies are due to the existence of unequally-sized critical blocks, in addition to the existence of several small critical blocks. Finally, as intuition suggests, the $\overline{f_{int}}$ for mixed-correlated PFSPs falls somewhere in-between those for job-correlated and machine-correlated PFSPs. However, increases in either n or α clearly drive the mixed-correlated PFSPs closer to their machine-correlated counterparts (e.g., the entries for $n = 100$ and $n = 200$, $0.5 \leq \alpha \leq 1.0$).

Table 1: The Mean Fraction of SH_{op} Moves ($\overline{f_{int}}$) that are Internal to a Critical Block for Two Groups of Random PFSPs, with Standard Deviations Shown in Parenthesis

	Problem Size			
Type	20×20	50×20	100×20	200×20
Random $[1, 99]$	0.11 (0.11)	0.19 (0.17)	0.21 (0.17)	0.31 (0.24)
Random $[45, 55]$	0.11 (0.13)	0.17 (0.15)	0.21 (0.17)	0.25 (0.19)

Table 2: The Mean Fraction of SH_{op} Moves ($\overline{f_{int}}$) that are Internal to a Critical Block for Structured PFSPs. The Last Column Indicates f_{int} for the Corresponding Prototypical Critical Path Topology, Where Available.

	α											
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	Proto.
						20×20						
Job	0.11	0.17	0.20	0.20	0.22	0.21	0.22	0.23	0.23	0.23	0.22	0.40
Machine	0.11	0.45	0.57	0.60	0.64	0.65	0.63	0.67	0.69	0.73	0.68	0.80
Mixed	0.18	0.28	0.35	0.38	0.45	0.46	0.50	0.59	0.57	0.61	0.60	N/A
						50×20						
Job	0.16	0.21	0.25	0.27	0.27	0.27	0.30	0.31	0.30	0.29	0.30	0.46
Machine	0.15	0.69	0.76	0.82	0.81	0.83	0.84	0.81	0.87	0.87	0.84	0.92
Mixed	0.18	0.51	0.65	0.70	0.72	0.71	0.74	0.77	0.79	0.83	0.81	N/A
						100×20						
Job	0.21	0.20	0.25	0.28	0.31	0.31	0.32	0.33	0.33	0.33	0.34	0.48
Machine	0.19	0.77	0.88	0.84	0.89	0.90	0.88	0.90	0.89	0.88	0.87	0.96
Mixed	0.23	0.63	0.78	0.78	0.83	0.81	0.80	0.87	0.87	0.89	0.87	N/A
						200×20						
Job	0.26	0.24	0.28	0.30	0.32	0.35	0.35	0.35	0.36	0.36	0.38	0.49
Machine	0.24	0.87	0.88	0.91	0.89	0.90	0.90	0.93	0.92	0.93	0.94	0.98
Mixed	0.25	0.75	0.84	0.87	0.80	0.91	0.90	0.90	0.91	0.92	0.93	N/A

The $\overline{f_{int}}$ of structured PFSPs are generally much larger than those of random PFSPs; for many machine-correlated and mixed-correlated PFSPs, over 80% of the moves are provably non-improving. A relatively recent trend in research on the PFSP has focused on using simple move operators such as SH_{op} in conjunction with more complex local search algorithms

to obtain performance equivalent to algorithms using more expensive and powerful move operators such as NS_{op}: e.g., see Ben-Daya and Al-Fawzan (1998) or Stützle (1999). Our results indicate that this trend has been successful in part because these algorithms are evaluated using random PFSPs, in which the fraction of non-improving moves is much smaller — between 10% and 31%.

3.3 Critical Path Topology and NS_{op}

The NS_{op} move operator does not consider moves internal to a critical block, and therefore is not affected by critical path topology in the same manner as SH_{op}. Instead, the influence of critical path topology on NS_{op} is seen through changes in the total number of possible moves.

3.3.1 Definition of NS_{op}

NS_{op} also uses shifts to produce neighbors, but only considers a subset of shift moves that have the potential to immediately improve C_{max}. Consider a job $\pi(i)$ residing in a critical block B_x. Let B_{x-1} and B_{x+1} denote the critical blocks immediately preceding and succeeding B_x, respectively (we assume for now that they exist). For each i, $1 \leq i \leq n$, NS_{op} shifts the job $\pi(i)$ into each possible position of B_{x-1} and B_{x+1}. For example, the job at position 7 in Figure 2 is shifted into positions 3 through 6 and positions 8 through 10.

While conceptually simple, the details of NS_{op} are complex, making it much harder to implement than SH_{op}. No researchers other than ourselves and Reeves and Yamada (1998) have reported independent implementations of this neighborhood operator (although we were able to replicate the results of Nowicki and Smutnicki 1996).

We now precisely define the set of moves considered by NS_{op}, as certain details are required to understand how the performance of NS_{op} is affected by critical path topology. Suppose a random critical path of a solution π consists of l critical blocks B_i, $1 \leq i \leq l$. In addition, we define the dummy critical blocks B_0 and B_{l+1}. Let $B_i(\mathit{left})$ and $B_i(\mathit{right})$ denote the positions of the first and last jobs in the block B_i, respectively. By convention, $B_0(\mathit{left}) = B_0(\mathit{right}) = 1$ and $B_{l+1}(\mathit{left}) = B_{l+1}(\mathit{right}) = n$. Note that $\forall i$, $1 \leq i \leq l$, $B_i(\mathit{right}) = B_{i+1}(\mathit{left})$ and $B_i(\mathit{left}) = B_{i-1}(\mathit{right})$. Finally, let $\mathit{mach}(B_i)$ denote the machine on which the critical block B_i resides.

In computing the set of shifts for a particular job $\pi(j)$, we distinguish two cases: (1) $\pi(j)$ is internal to a critical block and (2) $\pi(j)$ resides at the endpoint of a critical block. Both

cases require the specification of a parameter ϵ, $0.0 \leq \epsilon \leq 1.0$; as shown below, ϵ influences the neighborhood size by restricting attention to a fraction ϵ of the possible left and right shifts for each $\pi(j)$.

We consider case (1) first. Let B_i, B_{i-1}, and B_{i+1} denote the critical blocks containing, immediately preceding, and immediately succeeding $\pi(j)$, respectively. Define $\Delta_{right} = \lfloor \epsilon(B_{i+1}(right) - B_{i+1}(left)) \rfloor$ and $\Delta_{left} = \lfloor \epsilon(B_i(left) - B_{i-1}(left)) \rfloor$. $\forall j \neq n$ the set of right-shifts $RS(j, \epsilon)$ is given by $RS(j, \epsilon) = \{(j, x) : B_i(right) \leq x \leq B_i(right) + \Delta_{right}\}$ if $mach(B_i) \neq m$, and $RS(j, \epsilon) = \emptyset$ otherwise. Similarly, $\forall j \neq 1$, the set of left-moves $LS(j, \epsilon)$ is given by $LS(j, \epsilon) = \{(j, x) : B_i(left) - \Delta_{left} \leq x \leq B_i(left)\}$ if $mach(B_i) \neq 1$, and $LS(j, \epsilon) = \emptyset$ otherwise.

Next, we consider case (2), in which two critical blocks share the job $\pi(j)$. The computation of $RS(j, \epsilon)$ is identical with that of case (1), except that we let B_i denote the *second* critical block containing $\pi(j)$. In computing $LS(j, \epsilon)$, we let B_i denote the *first* critical block containing $\pi(j)$. Nowicki and Smutnicki (1996) first define a utility function $\omega(x)$ where $\omega(x) = 0$ if $x > 2$ and $\omega(x) = 1$ otherwise. Borrowing the definition of Δ_{left} from case (1), $\forall j \neq 1$, the set of left-moves $LS(j, \epsilon)$ is given by $LS(j, \epsilon) = \{(j, x) : B_i(left) - \Delta_{left} \leq x \leq B_i(left) - \omega(size(B_i))\}$ if $mach(B_i) \neq 1$, and $LS(j, \epsilon) = \emptyset$ otherwise. The introduction of the ω function serves to prevent duplication of the single shift-move within a critical block of size 2.

Nowicki and Smutnicki (1996) use the parameter ϵ to reduce the neighborhood size in random PFSPs with a large number of jobs. Given an $n \times m$ test problem, they use $\epsilon = 0.0$ if $n/m > 3$, $\epsilon = 0.5$ if $2 < n/m \leq 3$, and $\epsilon = 1.0$ otherwise ($n/m \leq 2$). These heuristic rules were arrived at experimentally "in order to ensure a compromise between the running time and solution quality" (Nowicki and Smutnicki 1996, page 170). These rules were derived using Taillard's PFSPs; in Section 5 we examine the applicability and effectiveness of these rules for our structured PFSPs.

3.3.2 Effect of Critical Path Topology on NS_{op}

We denote the number of moves under NS_{op} from a solution π by $|NS_{op}(\pi)|$. In the prototypical job-correlated critical path topology, the two critical blocks are on machines 1 and m and contain $n/2 - 1$ and $n/2$ internal jobs, respectively. If $\epsilon = 0.0$, each job internal to a critical block can be shifted into only one position in the other critical block. Similarly, if $\epsilon = 0.5$ and $\epsilon = 1.0$, each job internal to a critical block can be shifted into either one-half or

all of the positions in the other critical block, respectively. Independent of ϵ, the job shared by the two critical blocks can only be shifted into positions 1 and n. Clearly, the number of moves $|NS_{op}(\pi)|$ depends heavily on ϵ, and is given by:

$$|NS_{op}|_{JC} = \begin{cases} n+1 & \text{if } \epsilon = 0.0 \\ n^2/4 + 3/2 & \text{if } \epsilon = 0.5 \\ n^2/2 + 1 & \text{if } \epsilon = 1.0 \end{cases}$$

In the prototypical machine-correlated critical path topology, there is a single critical block containing $n-2$ internal jobs, each of which can only be shifted to the front and end of the critical block (positions 1 and n, respectively). Further, the jobs $\pi(1)$ and $\pi(n)$ can only be shifted to the other end of the critical block. Thus, independent of ϵ, the number of moves $|NS_{op}|$ for the machine-correlated critical path topology given by:

$$|NS_{op}|_{MC} = 2(n-1).$$

Given a prototypical machine-correlated critical path topology, a full neighborhood evaluation ($\epsilon = 1.0$) is relatively efficient, as $|NS_{op}|_{MC}$ grows linearly in n. In contrast, the growth of $|NS_{op}|_{JC}$ is quadratic in n when $\epsilon = 1.0$, making a full neighborhood evaluation relatively expensive. When $n > 40$, the full neighborhood is only sampled ($\epsilon = 0.5$ or $\epsilon = 0.0$). Here, the issue is not the total number of moves, but whether or not sampling eliminates critical moves, thereby reducing the performance of any local search algorithm using NS_{op}. The question of whether neighborhood sampling impacts performance is an empirical one, and is addressed in Section 5.

For each of our problem groups, we used *NEH-RS* to produce 100 solutions to each test problem in the group, and then computed $|NS_{op}|$ for each test problem using a single random critical path. We computed the mean $|NS_{op}|$, denoted $\overline{|NS_{op}|}$, by averaging over the 100 critical paths from each of the 100 test problems. The results are shown in Tables 3 and 4 for our random and structured PFSPs, respectively.

Table 3: The Mean Number of Moves under NS_{op} ($\overline{|NS_{op}|}$) for Random PFSPs, with Standard Deviations Shown in Parenthesis

	Problem Size			
Type	20×20	50×20	100×20	200×20
Random $[1, 99]$	110.0 (29.7)	264.8 (101.9)	179.6 (24.3)	365.2 (54.2)
Random $[45, 55]$	107.5 (29.9)	264.7 (96.1)	180.3 (23.3)	368.2 (48.3)

Table 4: The Mean Number of Moves Under NS_{op} ($\overline{|NS_{op}|}$) for Structured PFSPs. The Last Column Indicates $|NS_{op}|$ for the Corresponding Prototypical Critical Path Topology, Where Available.

	α											
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	Proto.
						20×20						
Job	110.1	128.6	146.3	151.9	157.8	157.7	162.2	162.1	167.5	168.7	172.1	201.0
Machine	108.0	107.0	94.0	88.5	75.6	71.4	68.7	62.7	64.0	55.0	56.6	38.0
Mixed	115.8	148.2	147.6	153.9	138.2	129.4	130.6	96.1	107.7	100.2	97.4	N/A
						50×20						
Job	267.3	271.9	300.2	307.8	356.3	394.4	400.0	412.6	419.9	433.8	456.8	626.5
Machine	259.5	196.3	173.1	140.1	155.4	145.7	116.0	123.9	115.7	111.9	122.7	98.0
Mixed	267.9	359.2	308.8	265.6	250.4	256.5	219.6	221.4	174.8	161.8	180.3	N/A
						100×20						
Job	179.5	175.0	166.7	155.7	142.0	133.9	130.4	125.6	123.5	118.6	117.9	101.0
Machine	176.2	183.1	190.2	187.2	187.4	188.5	183.2	189.9	185.3	183.3	187.0	198.0
Mixed	164.9	183.7	186.0	183.8	185.5	186.1	184.8	187.4	188.5	190.0	191.7	N/A
						200×20						
Job	369.9	360.8	343.3	333.1	321.4	316.2	300.0	291.4	284.0	271.6	267.4	201.0
Machine	362.7	379.3	379.2	374.1	368.9	379.8	373.2	382.4	378.6	379.9	383.1	398.0
Mixed	341.2	379.0	379.0	383.5	376.5	384.1	383.4	379.7	377.0	381.8	380.1	N/A

First, we consider the $\overline{|NS_{op}|}$ of random PFSPs, as shown in Table 3. The relatively large standard deviations are due to the infrequent occurrence of critical paths with very large critical blocks. As with $\overline{f_{int}}$, the width of the sampling interval has no discernible influence on $\overline{|NS_{op}|}$. The discontinuity in the growth of $\overline{|NS_{op}|}$ between $n = 50$ and $n = 100$ is due to the transition of ϵ from 0.5 to 0.0. As expected, there is little difference between the $\overline{|NS_{op}|}$ of random PFSPs and structured PFSPs with $\alpha = 0.0$ (see Table 4).

Next, we consider the results for our structured PFSPs, as shown in Table 4. The standard deviations (not reported) for all of the 100×20 and 200×20 problem groups, and for the 20×20 and 50×20 at small α, are similar to those of the corresponding random PFSP group. At large α, the standard deviations for the 20×20 and 50×20 are significantly larger than for random PFSPs. For example, at $\alpha = 1.0$ the standard deviations for our 20×20 job-correlated, machine-correlated, and mixed-correlated problem groups are 39.4, 38.6, and 65.2, respectively.

For the 100×20 and 200×20 machine-correlated PFSPs, independent of α, the observed $\overline{|NS_{op}|}$ only slightly under-cut the value predicted by the prototypical topology. In contrast, for the 20×20 and 50×20 problems, the $\overline{|NS_{op}|}$ are much larger than the predicted values. This discrepancy, which is due to the presence of several small critical blocks in addition to

the large dominant critical block, diminishes as $\alpha \rightarrow 1.0$, although it never disappears.

For 100×20 and 200×20 job-correlated PFSPs, the $\overline{|NS_{op}|}$ are larger than the value predicted by the prototypical topology, while for 20×20 and 50×20 job-correlated PFSPs the observed $\overline{|NS_{op}|}$ are less than the predicted value. In both cases, the discrepancy diminishes as $\alpha \rightarrow 1.0$, and is due to both the unequal size of the two large critical blocks and the presence of several small critical blocks. The improved accuracy of the predicted values as $\alpha \rightarrow 1.0$ is due to the gradual appearance of the bi-modal critical block distribution in job-correlated PFSPs as $\alpha \rightarrow 1.0$.

For 100×20 and 200×20 mixed-correlated PFSPs, the $\overline{|NS_{op}|}$ closely mirror the values for machine-correlated PFSPs, as expected given the similarities in their critical path topologies. However, for the 20×20 and 50×20 problems, the values for mixed-correlated PFSPs are much larger than the corresponding values for machine-correlated PFSPs. This discrepancy is due to the non-negligible number of mid-sized critical blocks in our mixed-correlated PFSPs.

Critical path topology influences the performance of NS_{op} through changes in $\overline{|NS_{op}|}$. Consider our 20×20 and 50×20 structured PFSPs. There are generally far fewer moves in machine-correlated PFSPs than in random PFSPs. For local search algorithms based on NS_{op}, the question is whether or not the reduction is too substantial, in that it may prevent the algorithm from exploring good regions of the search-space. In contrast, there are generally far more moves under NS_{op} in job-correlated PFSPs than in random PFSPs, and the question is whether or not the cost associated with evaluating the additional moves significantly impacts algorithm performance. Similar questions arise in an analysis of our 100×20 and 200×20 structured PFSPs. We empirically address the issue of the impact of changes in $\overline{|NS_{op}|}$ on algorithm performance in Section 5.

4. The Influence of Structure on the Distribution of Local Optima

While important, factors such as neighborhood size (for NS_{op}) and the fraction of non-improving neighbors (for SH_{op}) are not the only search-space features that affect algorithm performance. Another critical factor is the distribution of local optima, which we now analyze for both our random and structured PFSPs.

4.1 Local Optima and the "Big Valley" Distribution

Reeves (1995) and Reeves and Yamada (1998) demonstrated that the local optima in random PFSPs are distributed in a "big-valley" under both SH_{op} and NS_{op}. In a big-valley distribution: (1) local optima are clustered more tightly than randomly generated solutions, (2) better local optima tend to be closer to global optima, and (3) local optima in close proximity to one another tend to to be of similar quality. Boese et al. (1994) popularized the concept of a big-valley distribution through their analysis of local optima in the Traveling Salesman and Graph Bi-Partitioning problems. Subsequently, big-valley distributions have been used to motivate the design of several new algorithms. For example, Reeves and Yamada's (1998) path relinking algorithm is explicitly designed to exploit the big-valley distribution of local optima observed in random PFSPs, while Boese et al. (1994) used the big-valley distribution to motivate a new local search algorithm for the Traveling Salesman Problem.

To test for the presence of a big-valley distribution in a particular problem, a number of local optima are generated using a particular move operator and a relatively simple local search algorithm. Typically, either next-descent or steepest-descent are employed, although Reeves and Yamada (1998) use a more complex Monte-Carlo procedure to produce higher-quality local optima. In the experiments discussed in Sections 4.2 and 4.3, we generate 1000 local optima using a standard next-descent search algorithm, which is initiated from randomly generated solutions. In each iteration of next-descent, the neighbors are visited in a random order until a neighbor with a makespan better than the makespan of the current solution is identified. If no such neighbor exists, the procedure is terminated.

Because the optimal makespans of our PFSPs are unknown, we measure the average distance between each local optimum and all other local optima ($dist_{avg}$), instead of the distance between each local optimum and a global optimum ($dist_{opt}$) (Boese et al. 1994). Then, we produce a scatter-plot of $dist_{avg}$ vs. C_{max} for the 1000 local optima. Significant positive correlation (as measured either by Pearson's correlation coefficient or through a randomization test Reeves and Yamada 1998) is taken as evidence of a big-valley distribution.

Given a particular move operator, the distance between two local optima is ideally defined as the number of moves required to transform one of the local optima into the other. However, computation of this measure is generally intractable, and operator-independent measures are typically used in their place. In an n-job PFSP, solutions can be represented as permutations π of the integers 1 through n, and a straightforward operator-independent precedence-based

measure can be used to compute pairwise distances (Reeves and Yamada 1998). Given two local optima π and π', the distance $d(\pi, \pi')$ between π and π' is given by:

$$\frac{n(n-1)}{2} - \sum_{i,j,i \neq j} precedes(i, j, \pi) \wedge precedes(i, j, \pi')$$

where the function $precedes(i, j, \pi)$ returns 1 if job i occurs before job j in π and 0 otherwise.

4.2 Local Optima Distributions in Random PFSPs

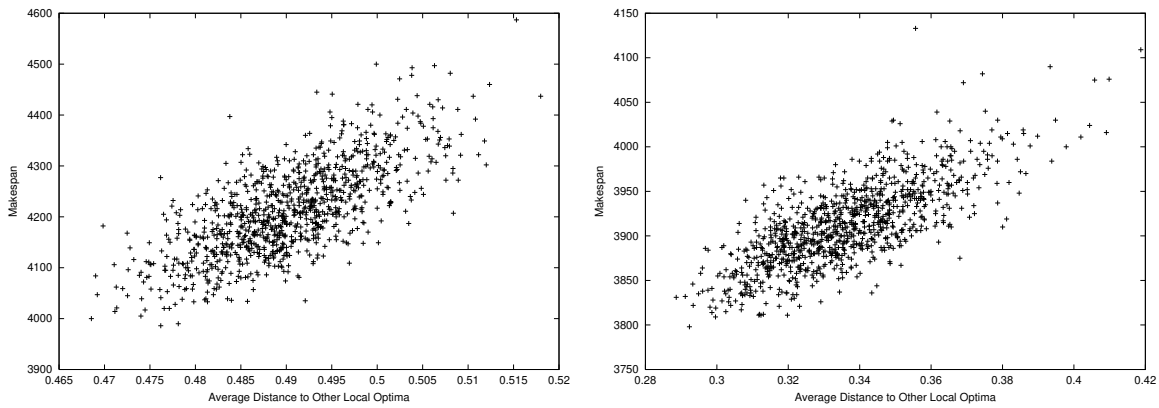

Figure 8: Distribution of 1000 Local Optima for a 50×20 Random PFSP. The Local Optima in the Left and Right Figures were Generated Using Next-Descent in Conjunction with NS_{op} and SH_{op}, Respectively.

In Figure 8, we show scatter-plots of $dist_{avg}$ vs. C_{max} for the local optima of a 50×20 random PFSP, using both the NS_{op} and SH_{op} move operators. In both cases, we see prototypical examples of a big-valley local optima distribution; positive correlation was confirmed using the randomization test described by Reeves and Yamada (1998). The local optima are clearly clustered, becoming more compact as better makespans are achieved. Similar results are obtained for all of the 50×20 PFSPs in Taillard's benchmark suite, in addition to nine other 50×20 random PFSPs we generated by sampling the d_{ij} from the interval $[1, 99]$. Further, we found no differences in the distributions for random PFSPs in which the d_{ij} were sampled from the interval $[45, 55]$.

The results in Figure 8 also exhibit an apparent paradox: the local optima produced using SH_{op} are significantly (on average) better than those produced using NS_{op}, although NS_{op} is found in most state-of-the-art local search algorithms for the PFSP. We have observed this phenomenon in all random and structured PFSPs, and explain the paradox by noting that NS_{op} considers only a *subset* of the shift moves that can reduce C_{max}, while SH_{op} considers

them all. Yet, the cost to produce local optima using SH_{op} is considerably higher than the cost using NS_{op}. For example, the mean CPU time required to produce 1000 local optima under SH_{op} for 50×20 random PFSPs was roughly 20 minutes on a Sun Ultra 10/400, compared to roughly 2 minutes under NS_{op}.

4.3 Local Optima Distributions in Structured PFSPs

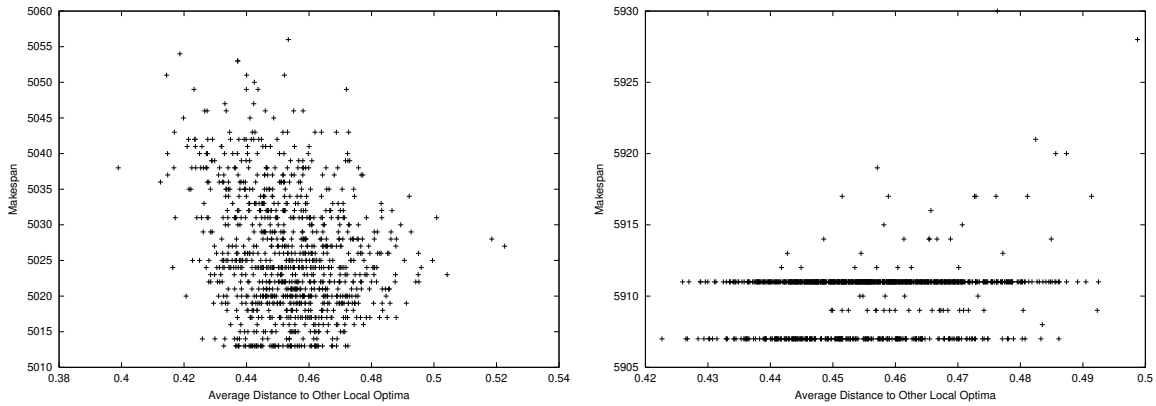

Figure 9: Distribution of 1000 NS_{op} Local Optima for a 50×20 Mixed-Correlated PFSP with $\alpha = 0.7$ (Left Figure) and a 50×20 Machine-Correlated PFSP with $\alpha = 0.3$ (Right Figure)

As with random PFSPs, we found strong evidence for the big-valley distribution in job-correlated PFSPs using both SH_{op} and NS_{op}, and independently of α. At very small α (≤ 0.2), the big-valley distribution was evident in both machine-correlated and mixed-correlated PFSPs, using either SH_{op} or NS_{op}. However, as α was increased, two very different kinds of distributions emerged. Examples are shown in Figure 9 for a 50×20 mixed-correlated PFSP with $\alpha = 0.7$ (left figure), and a 50×20 machine-correlated PFSP with $\alpha = 0.3$ (right figure); both examples were generated using NS_{op}. In the left side of Figure 9 we see no correlation between $dist_{avg}$ and C_{max}, while in the right side of Figure 9 another dominant structural feature emerges: plateaus of local optima at specific values of C_{max}. Additionally, as α was increased we found decreases in both the number of plateaus and the number of local optima not belonging to a plateau. The first type of distribution was more common under NS_{op}, with correlation rapidly degrading as $\alpha \rightarrow 1.0$. In contrast, the plateau-like distribution was more common under SH_{op}.

Following Boese et al. (1994) and Reeves (1995), we have defined a local optimum as a solution from which no improving move is available. Here, there is an implicit assumption

that a local search algorithm must accept a sequence of dis-improving moves in order to "escape" the attractor basin of a local optimum. While generally true for random PFSPs, we have found this assumption does not always hold in our structured PFSPs. We define a *plateau* as a set of solutions $\mathcal{P}$ such that (1) $\forall \pi \in \mathcal{P}$, $C_{max}(\pi) = C$ for some constant C and (2) for any distinct pair of solutions $\pi_1, \pi_2 \in \mathcal{P}$, there exists a sequence of solutions $\pi_1 = a_1, ..., a_n = \pi_2$ such that $\forall i$, $C_{max}(a_i) = C$ and for $1 \leq i \leq n-1$, π_{i+1} is a neighbor of π_i. A plateau is said to have an *exit* if there is an improving move from some solution on the plateau (Frank et al. 1997). A plateau without an exit is a locally optimal plateau, in that the only way to escape the plateau is by accepting a sequence of dis-improving moves.

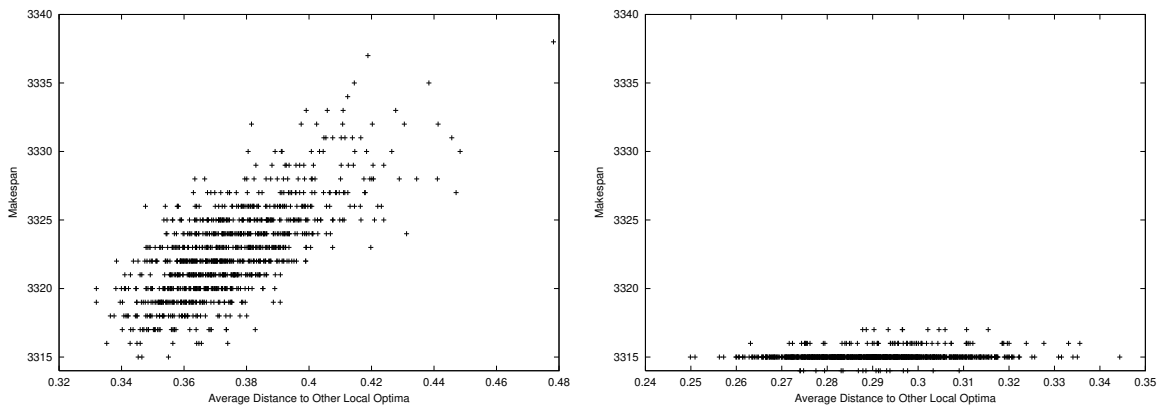

Figure 10: Distribution of 1000 SH_{op} Local Optima for a 50×20 machine-correlated PFSP with $\alpha = 0.1$. The Local Optima in the Left and Right Figures were Produced using Next-Descent, and a Combination of Next-Descent and Plateau-Escape, Respectively.

The existence of plateaus raises an immediate question: Are there exits from the plateaus we observed in machine-correlated and mixed-correlated PFSPs? To answer this question, we modified our next-descent procedure such that once a local optimum is identified, search enters a phase in which a random equal-makespan move is accepted until an improving move is found, at which point normal next-descent is initiated until another local optimum is identified. Because this algorithm has no well-defined termination point, we allocated 5000 iterations to each run, where an iteration consists of either finding a neighbor with an improving makespan (in the next-descent phase) or selecting a random equal-makespan neighbor (in the exit-finding phase); the last local optimum found by the procedure is returned.

We first consider the issue of escaping the plateaus induced by the SH_{op} move operator. In Figure 10, we show two local optima distributions for a particular 50×20 machine-correlated PFSP with $\alpha = 0.1$. The local optima in the left figure were produced using next-descent,

and exhibit a typical big-valley distribution. The local optima in the right figure were produced using our combined next-descent/plateau-escape algorithm, and exhibit a strong plateau-like distribution. By contrasting these two distributions, we see that the plateaus containing relatively poor solutions are easily escaped (i.e., they are not locally optimal), while the plateaus containing high-quality solutions are either very large, such that it is difficult to find an exit, or they do not contain any exits (i.e., they are locally optimal). Similar results were obtained for all of the machine-correlated and mixed-correlated PFSPs that we analyzed, independently of the type of local optima distribution under next-descent (e.g., see Figure 9); in all cases, poor-quality solutions were easily escaped via an exit from the plateau (although it is not obvious, the poor-quality solutions observed in the type of distribution shown in the left-hand side of Figure 9 are actually members of relatively small plateaus, which typically possess exits). We also performed a similar analysis on our job-correlated and random PFSPs. Although the local optima of random PFSPs often resided on small plateaus, we rarely found exits from these plateaus. In contrast, the local optima of job-correlated PFSPs often resided on large plateaus (which was not apparent in our analysis of next-descent local optima in these problems), and in cases where the plateau contained poor-quality solutions, exits were quite common; in other words, we found results similar to those shown in Figure 10.

Next, we perform a similar analysis of the plateaus induced by NS_{op}. Under next-descent, the local optima of random and job-correlated PFSPs typically belong to very small plateaus containing no exits. Consequently, the big-valley distributions persist in these problems even considering local optima produced using our combined next-descent/plateau-escape algorithm. This observation leads us to conjecture that good performance on random PFSPs will translate into good performance on job-correlated PFSPs, a conjecture that we test in Section 5. In contrast, we saw similar results to those shown in Figure 10 for machine-correlated and mixed-correlated PFSPs, although we often observed a number of poor-quality solutions residing on plateaus with no exits.

The existence of plateaus clearly has the *potential* to significantly impact the performance of any local search algorithm. In random PFSPs, local search generally proceeds by identifying and escaping a series of local optima, and the escape is generally performed through a sequence of dis-improving moves (e.g., in tabu search and simulated annealing). However, local optima in structured PFSPs are often members of plateaus containing a number of equally good solutions. Poor-quality plateaus can often be escaped by "wandering"

the plateau until an exit is found, primarily because these plateaus contain relatively few solutions. In contrast, high-quality plateaus contain many millions of solutions, which has prevented us from ascertaining whether or not exits are common for these plateaus. However, we have determined that high-quality plateaus are strong attractors; if a small sequence of non-improving moves from such a plateau is accepted, local search quickly returns to the same plateau. On the other hand, if there are exits from such plateaus, the challenge for local search is to quickly move toward the boundaries of the plateau in search of exits. We conjecture that memory-based algorithms such as tabu search may perform better in these situations than memoryless algorithms such as simulated annealing; the former are forced to move away from the current solution, while the latter are likely to meander in the vicinity of the current solution in the absence of a gradient.

To summarize, the distribution of local optima in random and structured PFSPs is often markedly different; the local optima of random PFSPs follow a big-valley distribution, while plateaus of equally-fit local optima are generally found in structured PFSPs. The existence of plateaus under SH_{op} was somewhat predictable, given that moves internal to a critical block are not prevented, although the topology of exits from these plateaus was unexpected (and parallels in many respects the plateaus found in MAX-SAT, Frank et al. 1997). In contrast, the existence of plateaus under NS_{op} was entirely unexpected, given that NS_{op} explicitly avoids moves internal to a critical block. In either case, the plateau structures clearly have the potential to impact the performance of any local search algorithm, and specifically those designed to exploit the presence of a big-valley local optima distribution.

5. Algorithm Performance on Random and Structured PFSPs

The analyses of Sections 3 and 4 demonstrated strong differences in both the critical path topologies and local optima distributions of random and structured PFSPs, and we argued that these differences have the potential to influence algorithm performance. In this section, we test this hypothesis by analyzing the performance of a number of well-known PFSP algorithms on our structured PFSPs, and contrast these results with the performance of these same algorithms on random PFSPs.

5.1 The Algorithms

We consider two constructive and two local search algorithms in our analysis. The first constructive algorithm is *NEH*, which is generally accepted as the best heuristic constructive algorithm for the PFSP. Individual runs of *NEH* are inexpensive (requiring less than 1 second for any of our test problems), and generally provide reasonable solutions. The second constructive algorithm, *NEH-RS*, is simply an iterated version of *NEH*. Both of the local search algorithms are implementations of tabu search, and differ only in the selection of move operator (either SH_{op} or NS_{op}). We selected tabu search algorithms because they have provided state-of-the-art performance on Taillard's PFSP benchmark suite for approximately the last 10 years.

5.2 *NEH* and *NEH* with Random Re-starts

The *NEH* algorithm (Nawaz et al. 1983) is a simple greedy procedure, summarized as follows (also see Taillard 1990):

(1) Sort the n jobs in decreasing order of total processing time (i.e., $\sum_{j=1}^{m} d_{ij}$ for job i).

(2) Schedule the largest two jobs in the order that minimizes the makespan of the partial schedule containing only these two jobs.

(3) For k=3 to n do

Insert the kth job into the location in the partial schedule, among the k possible, that minimizes the makespan of the resulting partial schedule.

Although the time complexity of *NEH* is relatively low ($\mathcal{O}(n^2m)$), it produces reasonably good (although by no means excellent) solutions to the problems in Taillard's (1993) benchmark suite.

NEH can be extended in a variety of ways. We consider an iterative version in which two *random* jobs are selected in Step **2**, and denote the resulting algorithm by *NEH-RS* (*NEH* with Random Re-starts). Despite the simplicity of this extension, we found that *NEH-RS* produces significantly better solutions than *NEH* on both random and structured PFSPs. Finally, we note that *NEH* can also serve as the basis of a branch-and-bound procedure, an alternative that we did not pursue.

5.3 Tabu Search for the PFSP: $Tabu_{SH}$ and $Tabu_{NS}$

Our first tabu search algorithm is the state-of-the-art algorithm introduced by Nowicki and Smutnicki (1996) (which they denote *TSAB*), and employs the NS_{op} move operator. We independently implemented their algorithm, and were able to replicate the performance results they reported for Taillard's test problems. We use the parameter settings documented in Nowicki and Smutnicki (1996), and denote the resulting algorithm by $Tabu_{NS}$.

Prior to $Tabu_{NS}$, the best algorithms for the PFSP were tabu search algorithms based on SH_{op}. First introduced by Taillard (1990), subsequent improvements were proposed by several researchers, including Reeves (1993) and Ben-Daya and Al-Fawzan (1998). Given the success of these algorithms, our second tabu search algorithm consists of the SH_{op} move operator in conjunction with the tabu search component of $Tabu_{NS}$ (with a single exception, noted below). We denote this algorithm by $Tabu_{SH}$. We decided not to consider existing tabu search algorithms based on SH_{op} for two reasons; first, we wanted to control for any performance differences that were attributable to discrepancies in the tabu search mechanisms, and second, many of the ideas in these algorithms were incorporated into $Tabu_{NS}$.

We did perform one modification to the tabu search mechanism of $Tabu_{NS}$ in our implementation of $Tabu_{SH}$. This modification is based on a simple performance enhancement reported by Stützle (1999), introduced because of the large neighborhood sizes (relative to NS_{op}) induced by SH_{op}. Normally, $Tabu_{NS}$ generates all neighbors of a solution π, categorizes them as either non-tabu, tabu-but-aspired, or tabu. Then, the best non-tabu or tabu-but-aspired move is taken. Instead, we generate the neighbors of a solution π by iteratively considering the possible moves for each job in the solution (i.e., all moves for job $\pi(1)$, followed by all moves for job $\pi(2)$, etc.). After generating the moves for job $\pi(i)$, if a non-tabu or tabu-but-aspired move with a makespan better than $C_{max}(\pi)$ is found, then that move is immediately accepted, and no other moves are evaluated.

5.4 Methodology

To study the influence of structure on algorithm performance, we performed a simple factorial experiment. The independent variables were the problem size (20×20, 50×20, 100×20, or 200×20), structure type (job-correlated, machine-correlated, or mixed-correlated), structure level ($\alpha \in [0.1, 0.2, \cdots, 1.0]$), and algorithm (*NEH*, *NEH-RS*, $Tabu_{SH}$, or $Tabu_{NS}$). The sole dependent variable was the best value of C_{max} obtained by any run of each algorithm on

each of our structured PFSPs. For purposes of comparison, we also measured the value of the dependent variable for each of our algorithms on each problem in our random PFSP problem groups.

For both *NEH* and *NEH-RS*, we only performed a single run of each algorithm on each test problem; *NEH* produces a single solution, and we observed negligible variance over 10 independent runs of *NEH-RS* in a wide range of random and structured PFSPs. We performed 5 runs of both $Tabu_{SH}$ and $Tabu_{NS}$ on each test problem, and recorded the best solution found during any of the 5 runs. As reported by Reeves (1993) and others, heuristic initialization often improves the performance of tabu search algorithms. Consequently, the first run of each of our tabu search algorithms initiates from the *NEH* solution, while the remaining 4 runs are initiated from random solutions.

All runs were conducted using SUN Ultra 10/400 workstations, each with 512 MB of RAM and running the Solaris 2.8 operating system. Each algorithm was implemented in `C++`, and compiled using the egcs compiler (v2.95) with level 3 optimization. With the exception of *NEH* (which runs in sub-second time for all of our test problems), we allocated 5 minutes of CPU time to each of the 20×20 and 50×20 runs, and 10 minutes of CPU time to each of the 100×20 and 200×20 runs; due to hardware advances, such a limit allows for more than five times the number of $Tabu_{NS}$ iterations for random 200×20 PFSPs than were performed in Nowicki and Smutnicki (1996). Normally, we do not advocate comparing algorithm performance by allocating a fixed CPU time to all runs, as differences in performance can often be attributed to implementation details. However, all of the algorithms we analyze share the following characteristics: 1) run-times are dominated ($> 99\%$) by either neighborhood evaluations (in the case of $Tabu_{NS}$ and $Tabu_{SH}$) or look-ahead evaluations (in the case of *NEH* and *NEH-RS*), and 2) both the neighborhood and look-ahead evaluations are produced using the same algorithmic principles, and in our case, the same implementation.

For each of our test problems, we also computed a lower bound on C_{max}. For random PFSPs, the straightforward lower bound used by Taillard (1993) is computationally inexpensive and relatively tight. The Taillard lower bound (which is based on a lower bound defined by Rinnooy Kan 1976) is also relatively tight for both machine-correlated and mixed-correlated problems, primarily because the derivation is machine-oriented. For the same reason, we found the Taillard lower bound to be exceptionally poor for job-correlated PFSPs. In a proportionate flow-shop, all processing times d_{ij} for a given job i are identical, and the optimal makespan can be computed in polynomial time (Pinedo 1995, page 103). To compute lower

bounds for job-correlated PFSPs, we translated each problem into a proportionate flow-shop problem by letting the processing time for each job i equal $min_{1 \leq j \leq m}\{d_{ij}\}$. The lower bound for the job-correlated PFSP is then taken as the optimal makespan for the corresponding proportionate flow-shop instance. Empirically, this approximation yields relatively tight lower bounds for our job-correlated PFSPs.

5.5 Results

Table 5: For Structured PFSPs, the Number of Problems (out of the 100 Possible) for Which the Best Solution Found by any Algorithm was Equal to the Computed Lower Bound

	α										
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20×20											
Job	0	13	47	63	68	67	79	83	86	90	87
Machine	0	12	34	34	48	57	57	69	70	68	72
Mixed	6	1	4	11	15	25	20	37	37	45	45
50×20											
Job	0	1	11	30	49	65	70	83	79	89	91
Machine	1	26	47	57	67	66	78	69	83	84	79
Mixed	1	11	33	48	47	50	55	58	63	75	63
100×20											
Job	0	0	0	0	9	28	31	59	66	68	80
Machine	1	32	55	56	68	74	85	90	87	90	85
Mixed	0	16	42	52	66	60	64	73	76	81	79
200×20											
Job	0	0	0	0	0	0	0	0	1	4	7
Machine	2	35	57	67	71	79	86	80	89	86	87
Mixed	0	31	51	62	63	77	75	77	84	83	87

An instance of the PFSP is often labeled as "difficult" if the makespans of solutions obtained by good heuristic algorithms are far from computed lower bounds on the optimal makespan (e.g., Taillard 1993). This definition clearly assumes that the lower bounds are relatively tight, which is the case for the lower bound computations we have defined for our random and structured PFSPs. Consequently, we first consider the relative difficulty of random and structured PFSPs by comparing the computed lower bounds and the makespan of the best solutions found by *any* of our 4 algorithms. For each of our structured problem classes, we measured the number of problems, out of the 100 possible, in which the makespan of the best solution was identical to the computed lower bound (and consequently optimal); the results are reported in Table 5.

In Section 2, we hypothesized that as α increased, the difficulty of structured PFSPs would decrease, due to a smaller number of bottleneck jobs or machines. The results in Table 5 confirm this hypothesis. However, we did not anticipate the rapid, non-linear decrease in difficulty seen in much of Table 5. For machine-correlated PFSPs, the proportion of optimal solutions is typically larger than half once $\alpha \approx 0.2$. Further, the proportion of optimal solutions grows with increases in both α (approaching roughly 80% at $\alpha = 1.0$) and problem size. The effect is slightly less marked for mixed-correlated PFSPs, although we observe the same general trends in difficulty. For job-correlated PFSPs, we see similar trends for the 20×20 problems. However, the number of optimal solutions in job-correlated PFSPs actually *decreases* with increases in problem size. Interestingly, this contradicts Taillard's (1993, p. 281) conjecture that for a fixed m, $\lim_{n\to\infty} Prob(C^*_{max} = LowerBound) = 1.0$, although our results do support the conjecture for both machine-correlated and mixed-correlated PFSPs. The lower bound was *never* reached for any of our random PFSPs. Finally, we note that the measures reported in Table 5 represent *only* the proportion of best solutions proved optimal by comparing the makespan to the computed lower bound; it is possible, or even likely, that other best solutions are optimal (which could be determined by either tighter lower bound computations, or via branch-and-bound algorithms).

To summarize, our results indicate that a large number of structured PFSPs are actually very easy to solve to optimality. In contrast, for random PFSPs the best solutions were generally very far from the computed lower bounds. For both machine-correlated and mixed-correlated PFSPs, the majority of problems are relatively easy once $\alpha \approx 0.2$, and became easier with increases in both α and problem size. We saw a similar dependency on α for small job-correlated PFSPs, but the effect dissipated with increases in problem size. However, we cannot yet conclude that larger job-correlated PFSPs are difficult, as the effect may be due to either problem difficulty or to inaccuracies in the computed lower bound.

In the preceding analysis, we defined problem difficulty in terms of both the computed lower bound and the makespan of the best solution obtained by *any* of our algorithms. In doing so, we necessarily avoided the issue of *relative* algorithm performance on structured PFSPs, which we now consider. In Tables 6 through 10, we report both (1) the number of problems (out of the 100 possible) for which each algorithm found a solution equal to the best solution found in any run and (2) the mean percent error above the best solution in the cases where an algorithm failed to find such a solution.

First, we consider the performance of the algorithms on random PFSPs, as shown in

Table 6: Algorithm Performance on Random 20×20 PFSPs. X(Y) Denotes the Algorithm Found the Best Overall Solution X Times, out of the 100 Possible, with a Mean Relative Error of Y Percent on the Remaining Instances.

	NEH	*NEH-RS*	$Tabu_{SH}$	$Tabu_{NS}$
		20×20		
Random $[1, 99]$	0(3.87)	1(1.55)	99(.09)	96(.04)
Random $[45, 55]$	0(.49)	1(.21)	99(.05)	98(.05)
		50×20		
Random $[1, 99]$	0(5.09)	0(2.94)	16(.72)	87(.20)
Random $[45, 55]$	0(.65)	0(.35)	36(.09)	86(.05)
		100×20		
Random $[1, 99]$	0(4.28)	0(2.63)	0(1.42)	100(0)
Random $[45, 55]$	0(.51)	0(.29)	2(.12)	98(.05)
		200×20		
Random $[1, 99]$	0(2.94)	0(1.80)	0(1.38)	100(0)
Random $[45, 55]$	0(.34)	0(.20)	0(.16)	100(0)

Table 6. Relative performance differences were independent of the random PFSP group, although the relative errors were smaller in the $[45, 55]$ problem group. *NEH-RS* outperforms *NEH*, but both are significantly worse than $Tabu_{NS}$, and combined only found the best solution in 1 instance. Both tabu search algorithms outperform *NEH* and *NEH-RS*, but $Tabu_{NS}$ is the clear winner, confirming the results reported in the PFSP literature: $Tabu_{SH}$ completely fails to scale with increases in problem size. We see very similar trends when considering relative errors. $Tabu_{NS}$ still performs well even when it fails to find the best solution on the smaller problems. $Tabu_{SH}$ again scales poorly at large problem sizes, although it still out-performs *NEH-RS*. The poorest performers are *NEH* and *NEH-RS*, although in contrast to $Tabu_{SH}$ the relative error actually *decreases* with grow in the problem size. The inability of $Tabu_{SH}$ to scale was anticipated (see Section 3.2), as very large critical blocks occur frequently in the larger random PFSPs, substantially increasing the effort expended in evaluating provably non-improving moves.

Next, we consider the results for machine-correlated and mixed-correlated PFSPs. The first salient feature is the strong relative performance of $Tabu_{NS}$, which for $\alpha \neq 0.0$ achieves the best solution 91% of the time in the *worst* case. Clearly, the state-of-the-art performance of $Tabu_{NS}$ on random PFSP instances transfers to machine and mixed-correlated PFSPs. In contrast, we see that $Tabu_{SH}$ generally under-performs $Tabu_{NS}$, and even *NEH-RS* at larger problem sizes. The exception is with 20×20 problems, in which $Tabu_{SH}$ is competitive with $Tabu_{NS}$. Again, the latter can be explained in terms of the fraction of non-improving SH_{op} moves for structured 20×20 problems, which is significantly less than that for the

Table 7: Algorithm Performance on Structured 20×20 PFSPs. X(Y) Denotes the Algorithm Found the Best Overall Solution X Times, out of the 100 Possible, with a Mean Relative Error of Y Percent on the Remaining Instances.

	α										
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
					20×20 - *NEH*						
Job	0(.66)	8(.28)	38(.15)	49(.10)	72(.10)	71(.09)	77(.07)	84(.06)	78(.06)	86(.07)	94(.08)
Machine	0(.73)	9(.49)	18(.30)	21(.23)	37(.26)	44(.20)	40(.21)	59(.20)	63(.21)	63(.16)	62(.18)
Mixed	5(.36)	5(.39)	8(.36)	10(.31)	19(.24)	22(.24)	32(.25)	41(.20)	43(.22)	46(.16)	44(.15)
					20×20 - *NEH-RS*						
Job	2(.27)	40(.11)	76(.08)	83(.06)	95(.07)	95(.05)	96(.04)	99(.03)	95(.04)	98(.04)	99(.04)
Machine	1(.31)	32(.22)	54(.15)	55(.11)	72(.12)	79(.09)	71(.08)	84(.10)	88(.14)	75(.10)	84(.12)
Mixed	25(.15)	28(.18)	30(.14)	44(.13)	55(.10)	61(.09)	61(.09)	76(.09)	72(.08)	78(.07)	84(.07)
					20×20 - $Tabu_{SH}$						
Job	100(0)	98(.03)	100(0)	99(.05)	100(0)	100(0)	99(.04)	100(0)	98(.06)	100(0)	100(0)
Machine	100(0)	99(.06)	99(.17)	99(.03)	100(0)	100(0)	98(.04)	100(0)	99(.03)	100(0)	100(0)
Mixed	99(.04)	100(0)	99(.03)	96(.12)	97(.07)	99(.06)	97(.07)	100(0)	100(0)	100(0)	99(.03)
					20×20 - $Tabu_{NS}$						
Job	99(.05)	100(0)	100(0)	100(0)	100(0)	99(.03)	99(.03)	100(0)	100(0)	98(.04)	100(0)
Machine	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)
Mixed	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)	100(0)

larger problem sizes. Both *NEH* and *NEH-RS* do relatively poorly at small α, but improve significantly with increases in both problem size and α. To our surprise, the performance of *NEH-RS* is nearly competitive with $Tabu_{NS}$ for large α. The second salient feature is the extremely small relative error of *all* of the algorithms; most relative errors are only a small fraction of 1% percent. Further, the relative errors for all algorithms decrease as $\alpha \rightarrow 1.0$.

We see different results for job-correlated PFSPs. For small problem sizes, both $Tabu_{SH}$ and $Tabu_{NS}$ provide equally good performance. However, as problem sizes increase, the performance of $Tabu_{NS}$ does *not* increase relative to $Tabu_{SH}$ - neither algorithm consistently finds the best overall solution, and the performance of *NEH-RS* is not nearly as competitive as it was for machine-correlated and mixed-correlated instances. Despite these qualitative differences, the mean relative errors for all algorithms were still very small, and decreased as $\alpha \rightarrow 1.0$.

It is worth noting the differences in the total number of iterations performed by a single run of $Tabu_{SH}$ and $Tabu_{NS}$, given our CPU limit and architecture. For 20×20 problems, $Tabu_{SH}$ executes on average (across all problem types) 20K iterations, in contrast to 57K iterations for $Tabu_{NS}$. Considering 200×20 problems, $Tabu_{SH}$ executes less than 200 iterations on average, in contrast to 2K for $Tabu_{NS}$. Given these differences, the fact that $Tabu_{NS}$ fails to consistently dominate $Tabu_{SH}$ at all problem sizes is even more surprising. Further analysis indicates that both algorithms spend most of their time attempting to escape large sub-optimal plateaus, a task that can be extremely problematic.

Table 8: Algorithm Performance on Structured 50×20 PFSPs. X(Y) Denotes the Algorithm Found the Best Overall Solution X Times, out of the 100 Possible, with a Mean Relative Error of Y Percent on the Remaining Instances.

	α										
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
						50×20 - *NEH*					
Job	0(.86)	0(.60)	0(.33)	4(.23)	13(.16)	19(.12)	38(.10)	47(.08)	43(.08)	56(.05)	61(.06)
Machine	0(.95)	17(.23)	33(.15)	43(.15)	45(.11)	49(.11)	62(.13)	59(.08)	75(.07)	73(.07)	68(.11)
Mixed	0(.64)	6(.36)	17(.20)	40(.17)	41(.14)	40(.13)	47(.11)	55(.10)	62(.11)	73(.08)	65(.08)
						50×20 - *NEH-RS*					
Job	1(.44)	0(.30)	6(.15)	22(.10)	38(.06)	60(.06)	68(.05)	79(.04)	78(.04)	89(.03)	88(.04)
Machine	1(.50)	41(.11)	66(.07)	68(.07)	75(.09)	77(.06)	87(.09)	87(.05)	88(.04)	92(.03)	83(.05)
Mixed	0(.38)	17(.20)	45(.11)	66(.10)	66(.08)	70(.06)	74(.04)	83(.04)	86(.07)	90(.03)	84(.03)
						50×20 - $Tabu_{SH}$					
Job	50(.12)	35(.08)	54(.07)	72(.05)	84(.05)	88(.05)	91(.04)	95(.03)	95(.03)	98(.02)	96(.02)
Machine	60(.11)	75(.06)	92(.04)	95(.11)	97(.06)	95(.06)	98(.06)	98(.02)	99(.04)	98(.04)	99(.02)
Mixed	74(.07)	76(.07)	94(.06)	94(.05)	96(.02)	97(.02)	99(.07)	99(.04)	98(.02)	96(.03)	98(.02)
						50×20 - $Tabu_{NS}$					
Job	82(.09)	92(.08)	93(.07)	96(.03)	93(.03)	96(.04)	99(.03)	98(.04)	97(.05)	99(.02)	97(.03)
Machine	81(.06)	100(0)	100(0)	99(.05)	100(0)	100(0)	99(.02)	100(0)	100(0)	100(0)	100(0)
Mixed	68(.04)	95(.04)	96(.05)	98(.03)	100(0)	98(.03)	100(0)	100(0)	100(0)	100(0)	100(0)

To summarize, we see substantially different algorithmic behavior between random and structured PFSPs. Random PFSPs exhibit a fairly clear distinction in algorithm performance, with a higher mean relative error for the inferior algorithms. In contrast, the mean relative error for all algorithms on structured problems is typically very low. For machine-correlated and mixed-correlated PFSPs, increases in α create a "leveling" effect between all the algorithms, in that performance was nearly identical. For job-correlated PFSPs, the small mean relative errors were observed, but no such leveling occurred. Further, no clear winner emerged - on larger problems, $Tabu_{SH}$ and $Tabu_{NS}$ performed best on different problems.

6. Conclusions

A valid criticism of much AI and OR scheduling research is that existing benchmark problems do not contain features present in real-world problems. For example, both the PFSP and the JSP fail to consider many important aspects of real-world manufacturing problems, such as job-dependent set-up times. In response, AI and OR researchers have developed algorithms for enhanced scheduling problems which incorporate these additional constraints. However, there is an alternative route to realism that has rarely been pursued by the scheduling community: test problems whose features are modeled after features found in real-world manufacturing environments. The primary goal of our research was to study how syntactic

Table 9: Algorithm Performance on Structured 100×20 PFSPs. X(Y) Denotes the Algorithm Found the Best Overall Solution X Times, out of the 100 Possible, with a Mean Relative Error of Y Percent on the Remaining Instances.

	α										
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
						100 × 20 - *NEH*					
Job	0(.62)	0(.56)	0(.44)	0(.35)	0(.28)	0(.21)	3(.16)	6(.12)	17(.10)	9(.09)	26(.08)
Machine	1(.76)	16(.14)	46(.09)	39(.08)	58(.08)	65(.06)	70(.04)	76(.05)	65(.05)	71(.07)	77(.03)
Mixed	0(.66)	12(.26)	34(.11)	47(.09)	63(.06)	58(.09)	60(.06)	74(.05)	74(.04)	73(.05)	73(.04)
						100 × 20 - *NEH-RS*					
Job	1(.31)	0(.30)	0(.22)	0(.16)	3(.13)	5(.10)	18(.08)	59(.03)	44(.05)	45(.05)	54(.04)
Machine	2(.43)	53(.06)	72(.03)	69(.04)	79(.04)	87(.04)	88(.03)	95(.03)	86(.02)	87(.03)	93(.01)
Mixed	3(.38)	23(.16)	60(.07)	67(.05)	81(.04)	77(.06)	89(.03)	89(.04)	89(.02)	88(.03)	94(.03)
						100 × 20 - $Tabu_{SH}$					
Job	24(.09)	58(.06)	48(.05)	40(.06)	46(.05)	65(.04)	60(.03)	73(.03)	86(.02)	84(.03)	87(.02)
Machine	36(.08)	65(.05)	87(.04)	75(.03)	89(.05)	91(.04)	89(.04)	94(.03)	93(.03)	95(.03)	95(.02)
Mixed	83(.05)	53(.04)	80(.03)	87(.02)	95(.02)	95(.02)	97(.02)	93(.02)	98(.03)	98(.03)	99(.01)
						100 × 20 - $Tabu_{NS}$					
Job	86(.07)	63(.05)	71(.05)	78(.03)	79(.03)	77(.03)	84(.03)	85(.03)	92(.02)	95(.02)	90(.03)
Machine	79(.03)	99(.02)	96(.06)	99(.09)	100(0)	99(.10)	100(0)	99(.01)	99(.03)	100(0)	99(.01)
Mixed	30(.10)	91(.05)	97(.04)	97(.04)	95(.04)	98(.02)	98(.02)	99(.04)	100(0)	99(.02)	100(0)

problem structure influences the performance of scheduling algorithms, in particular for the PFSP. To this end, we developed a test problem generator that produces structured PFSPs based on features found in some real-world scheduling problems.

We established significant differences in both the critical path topology and local optima distributions of random and structured PFSPs. Given these differences, we also anticipated differences in algorithm performance. While we did observe some differences, we discovered a more important and unexpected result: the majority of structured PFSPs were easily solved to optimality by most algorithms, and for the problems not solved to optimality, any differences in relative algorithm performance were negligible. In other words, structured PFSPs tend to be very easy: high-quality, and in many cases optimal, solutions can be found by both simple and complex PFSP algorithms. This is in stark contrast to random PFSPs, for which there are significant performance differences among even state-of-the-art PFSP algorithms.

There is no doubt that scheduling problems *can* be difficult. Both the job-shop scheduling problem and the PFSP are known to be among the most difficult NP-complete problems. However, in contrast to random problems, our results indicate that many structured problems are actually relatively easy to solve. Further, structured problems influence algorithm performance in a number of ways, such that superior performance on random problems does not necessarily translate into superior performance on structured problems.

Table 10: Algorithm Performance on Structured 200 × 20 PFSPs. X(Y) Denotes the Algorithm Found the Best Overall Solution X Times, out of the 100 Possible, with a Mean Relative Error of Y Percent on the Remaining Instances.

	α										
Type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
						200 × 20 - *NEH*					
Job	0(.44)	0(.41)	0(.31)	0(.32)	0(.28)	0(.25)	0(.22)	0(.18)	0(.16)	0(.17)	0(.14)
Machine	1(.58)	24(.07)	41(.04)	46(.04)	59(.03)	60(.03)	63(.03)	69(.03)	76(.01)	78(.02)	82(.02)
Mixed	0(.53)	17(.12)	42(.09)	54(.05)	52(.04)	66(.04)	63(.03)	71(.03)	79(.03)	73(.03)	86(.02)
						200 × 20 - *NEH-RS*					
Job	4(.21)	0(.22)	0(.16)	0(.14)	0(.12)	0(.12)	2(.10)	1(.09)	1(.08)	1(.09)	0(.07)
Machine	4(.39)	51(.05)	71(.02)	66(.02)	80(.02)	80(.02)	83(.02)	84(.02)	93(.01)	89(.01)	92(.02)
Mixed	0(.35)	35(.08)	66(.07)	77(.04)	74(.03)	84(.03)	81(.01)	87(.02)	89(.01)	90(.02)	93(.01)
						200 × 20 - $Tabu_{SH}$					
Job	8(.13)	15(.10)	23(.07)	28(.06)	31(.05)	43(.05)	46(.04)	50(.03)	38(.03)	56(.03)	71(.02)
Machine	11(.15)	50(.03)	70(.02)	68(.02)	83(.03)	84(.02)	89(.02)	87(.02)	91(.01)	92(.02)	93(.02)
Mixed	54(.07)	46(.03)	70(.02)	77(.02)	78(.01)	89(.02)	86(.01)	86(.01)	92(.01)	87(.01)	95(.01)
						200 × 20 - $Tabu_{NS}$					
Job	95(.75)	87(.05)	85(.03)	81(.04)	76(.04)	65(.04)	62(.03)	60(.03)	76(.02)	53(.03)	51(.03)
Machine	95(.05)	95(.04)	97(.03)	98(.02)	98(.01)	95(.01)	96(.01)	99(.01)	99(.01)	99(.01)	100(0)
Mixed	55(.11)	96(.02)	95(.04)	96(.02)	97(.01)	98(.02)	99(.01)	97(.01)	99(.01)	98(.01)	100(0)

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant numbers F49620-97-1-0271 and F49620-00-1-0144. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The authors would also like to acknowledge the insightful comments of the anonymous referees.

References

Ben-Daya, M., M. Al-Fawzan. 1998. A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research* **109** 88–95.

Boese, K. D., A. B. Kahng, S. Muddu. 1994. A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters* **16** 101–113.

Dudek, R. A., S. S. Panwalker, M. L. Smith. 1992. The lessons of flowshop scheduling research. *Operations Research* **40** 7–13.

Frank, J., P. Cheeseman, J. Stutz. 1997. When gravity fails: local search topology. *Journal of Artificial Intelligence Research* **7** 249–281.

Garey, M. R., D. S. Johnson, R. Sethi. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* **1** 117–129.

Hooker, J. H. 1995. Testing heuristics: we have it all wrong. *Journal of Heuristics* **1** 33–42.

Nawaz,M., E. Enscore, I. Ham. 1983. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA, The International Journal of Management Science* **11** 91–95.

Nowicki, E., C. Smutnicki. 1996. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research* **91** 160–175.

Panwalker, S. S., R. A. Dudek, M. L. Smith. 1973. Sequencing research and the industrial scheduling problem. S. Elmaghraby, ed. *Proceedings of Symposium on Theory of Scheduling and its Applications*, Springer-Verlag, New York. 29–38.

Pinedo, M. 1995. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs, NJ.

Reeves, C. R. 1993. Improving the efficiency of tabu search in machine sequencing problems. *Journal of the Operational Research Society* **44** 375–382.

Reeves, C. R. 1995. A genetic algorithm for flowshop sequencing. *Computers and Operations Research* **22** 5–13.

Reeves C. R., T. Yamada. 1998. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation* **6** 45–60.

Rinnooy Kan, A. H. G. 1976. *Machine Scheduling Problems: Classification, Complexity and Computations.* Martinus Nijhoff, The Hague, The Netherlands.

Stützle, T. 1999. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA-98-04, Darmstadt University of Technology, Computer Science Department, Intellectics Group, Darmstadt, Germany.

Taillard, E. 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research* **47** 65–74.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* **64** 278–285.

Taillard, E. 1995. Comparison of iterative searches for the quadratic assignment problem. *Location Science* **3** 87–105.

Watson, J-P., L. Barbulescu, A. E. Howe, L. D. Whitley. 1999. Algorithm performance and problem structure for flow-shop scheduling. J. Hendler, H. Kautz, eds. *Proceedings of the*

16th National Conference on Artificial Intelligence (AAAI-99), Orlando, Florida, July 1999, AAAI Press/The MIT Press. 688–695.

Accepted by Michel Gendreau; received November 2000; revised June 2001; accepted October 2001.