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 benchmark problems to evaluate scheduling algorithm performance raises an important, and unresolved, question: are the results generalizable to more realistic problem instances? 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 scheduling environments. We perform experiments that demonstrate significant differences between the search space topology of random and structured flow-shop problems, and show that these differences *can* affect the performance of certain algorithms. 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 test problems are actually relatively easy to solve, raising concerns about the utility of commonly practiced methodologies for comparing the performance of scheduling algorithms.

(*Flow shop, Scheduling, Artificial Intelligence*)

---

1. Motivation and Introduction

To compare the performance of two scheduling algorithms, researchers typically run both algorithms on some test problems, record both the best solutions found and the computational costs to attain them, and analyze the results. Generally, synthetic test problem instances

are used, such as those found in the benchmark test suites available from the OR Library (http://www.ms.ic.ac.uk/info.html). These test problems are typically generated by selecting problem features at random; in both job-shop and flow-shop scheduling, job processing times are usually sampled uniformly from a single distribution. Problems are accepted as difficult if state-of-the-art algorithms fail to consistently find good solutions in reasonable running times, as measured relative to either a lower bound or a best-known solution.

Such a competitive, benchmark-driven approach to algorithm development is not without its critics, and although the pitfalls are well-known, the paradigm is still followed by a majority of researchers. The criticism generally targets either problems with the application of the paradigm or, at a deeper level, the paradigm itself.

The most common criticism of the benchmark-driven approach is that researchers run the risk of over-fitting their algorithms to specific benchmarks. The risk of over-fitting can be significantly reduced by using benchmarks containing a diverse array of problem types. For instance, benchmarks for the Quadratic Assignment Problem (QAP) consist of a variety of random, real-world, and structured problem instances (Taillard 1995). While both structured and random problems are produced using a problem generator, the former are modeled after real-world problems, while the later are typically unrealistic. Interestingly, although scheduling problems such as the flow-shop (FSP) and job-shop (JSP) are simple abstractions of the kinds of problems faced in real-world scheduling environments, structured synthetic benchmarks for these problems are extremely rare, and those few that exist are very limited in their scope.

For scheduling, the lack of diverse and realistic benchmarks is a particularly severe shortcoming, as there is an assumption, although usually implicit, underlying most comparative performance studies of scheduling algorithms: if an algorithm performs well on difficult, random synthetic problem instances, then it will also perform well on more realistic, structured problem instances. Given the lack of structured benchmarks, explicit testing of this assumption is rarely performed, and answers to related, basic research questions such as "Are structured problems easier than random problems?" remain unanswered.

Hooker (1995) provides a much deeper criticism of the competitive, benchmark-driven approach to algorithm development. He argues that such a paradigm has led to an arms-race mentality among researchers, in which more effort is expended on the algorithmic fine-tuning required to achieve competitive results rather than performing the hypothesis-driven research required to advance the understanding of algorithm behavior. In relation to scheduling, this

criticism is also well-deserved; researchers have developed many effective, high-performance scheduling algorithms, but a deep understanding of *why* these algorithms work well remains elusive.

Our research addresses these deficiencies for the permutation flow-shop scheduling (PFSP). First, we introduce a problem generator for producing structured PFSPs. The generator is not only capable of producing different types of structured PFSPs, but is also capable of varying the degree of randomness present. Second, we explore how search space topology and algorithm performance changes as we transition from random PFSPs to more structured ones.

Our first series of experiments focuses on characterizing the differences between the search space topology of random and structured PFSPs. In effect, we ask the question "Do differences in problem structure translate into differences in search space topology?" Search space topology is clearly algorithm-dependent, so a subset of algorithms must be selected before proceeding. Local search algorithms represent the state-of-the-art for the PFSP, and all of these algorithms are based on two well-known move operators. In the PFSP, the critical path topology plays a central role, as the performance of any move operator depends heavily on its ability to alter the critical path. In a series of experiments, we identify substantial differences between the critical path topologies of random and structured PFSPs, and show that these differences impact the performance of the two key PFSP move operators, albeit in different manners. By examining the influence of critical path topology on move operator performance, we only indirectly analyze the topology of the underlying search space. A more direct form of analysis is based on examination of local optima distributions. Here, we also find differences between the local optima distributions of random and structured PFSPs for both of the key move operators. Apparently, differences in problem structure translate into differences in search space topology.

Given differences in the search space topologies of random and non-random PFSPs, we anticipated differences in algorithm performance as well. This hypothesis is not without precedent. For example, Taillard (1995) found a strong dependency between QAP algorithm performance and the type of problem structure: different algorithms did best on different problems. To test this hypothesis for the PFSP, 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 negligible. 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 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.

2. Generating Structured PFSP Instances

We restrict our attention to the well-known n by m permutation flow-shop scheduling problem (PFSP). Here, n jobs must be processed, in the same order, on each of m machines. Each job progress through the machines in order, starting at machine 1 and ending at machine m. Concurrency is not allowed: a machine can only process a single job at a time, and processing of a job must complete once initiated. Furthermore, machine $j+1$ cannot begin processing a job until machine j has completed processing of the same job. An operation o_{ij} denotes the processing of a job i on a machine j, and requires a duration of d_{ij}. A candidate solution to the PFSP is simply a permutation of the n jobs, denoted π, with the ith job in π denoted by $\pi(i)$. The makespan C_{max} of π, denoted $C_{max}(\pi)$, is defined to be the completion time of job $\pi(n)$ on machine m, or operation $o_{\pi(n)m}$. The objective is to find a permutation π such that C_{max} is minimized. For $m \geq 3$, the makespan minimization form of the PFSP is NP-complete (Garey et al. 1976).

The most commonly used PFSP benchmark problems were introduced by Taillard (1993) and are available through the OR Library. In fact, the performance of most state-of-the-art PFSP algorithms (e.g., the Nowicki and Smutnicki (1996) tabu search algorithm and the Reeves and Yamada (1998) path relinking algorithm) is compared using *only* this benchmark. Taillard's benchmark problems were generated by selecting the operation durations d_{ij} uniformly from the interval $[1, 99]$ and then choosing a subset of problems based on 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 uniformly sampled from a single interval, problems generated using Taillard's procedure are, in expectation, syntactically unstructured.

Kan (1976) introduced methods for creating two types of structured PFSPs: job correlation and time gradients. In job-correlated problems, a small number ($< n$) of non-overlapping

distributions are defined, and all processing times for a given job are sampled from one of these distributions. In contrast, time gradients impose non-random syntactic structure by creating a trend in processing times as a function of machine. For instance, in problems with positive(negative) time gradients, jobs require less(more) time on early machines than on later machines. Kan developed these structured PFSPs to complement some random PFSPs 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 structure type, the number of PFSPs which were solved, or proved to optimality, was recorded. 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 than random PFSPs.

Reeves (1995) used Kan's structured PFSPs to compare the performance of a genetic algorithm, simulated annealing, and neighborhood search. In contrast to Kan, Reeves reported that PFSPs with positive time gradients were particularly easy to solve to optimality; the C_{max} values produced by the genetic algorithm were always equivalent to a computed lower bound. This result is not necessarily inconsistent with Kan's results; Kan was attempting to prove optimality using a branch-and-bound procedure, and it is unclear whether his algorithms terminated once a solution with C_{max} equal to a lower bound was identified (many branch-and-bound procedures do not). For job-correlated PFSPs, Reeves reported that simulated annealing and the genetic algorithm often produced equivalent C_{max} values, a result he interpreted as suggesting an overall smoothness in the search landscape. Optimality was not established (the obtained C_{max} were different from computed lower bounds), so no claim was made regarding the difficulty of Kan's 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 PFSPs are similar to machine-correlated PFSPs, 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 will likely have the largest processing time on the remaining machines. Figure 1 shows illustrative examples for each of the three types of correlation.

Producing PFSPs with each form of correlation is conceptually simple; in each case, processing times are uniformly sampled from a number of relatively narrow distributions. For job-correlated PFSPs, we use n distributions, one for each job. For machine-correlated

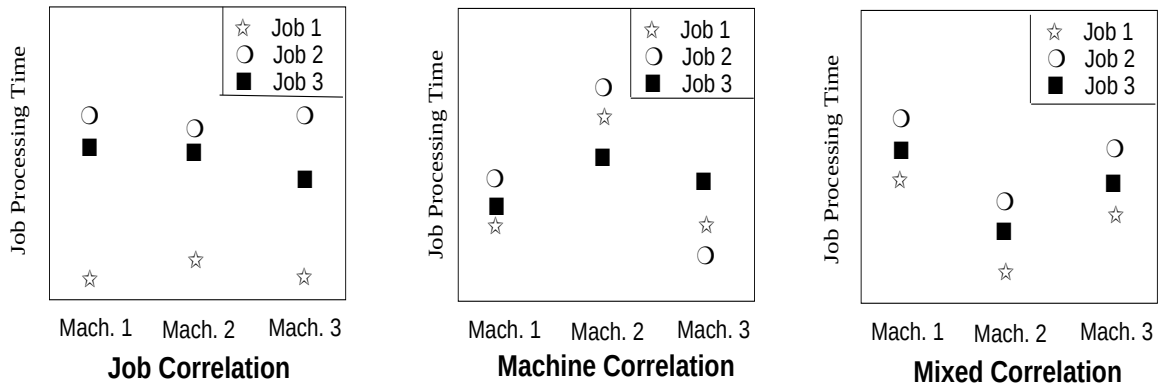

Figure 1: Examples of 3×3 PFSPs showing the three different forms of correlation. In job-correlated PFSPs, job processing times are largely independent of the machine. In machine-correlated PFSPs, job processing times are heavily dependent upon the machine. Mixed-correlated PFSPs are a form of machine-correlated PFSP in which the relative ranks of job processing times are generally consistent across the machines.

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 distribution variance subject to the boundary conditions imposed by the endpoints of I.

In Taillard's problems, the distribution means for all jobs and machines are very similar (≈ 50), especially for instances with large n or m. If we sampled the distribution means for our structured PFSPs uniformly from I, then our problems would in expectation be very different from Taillard's. However, we are interested in studying the differences in search space topology and algorithm performance between structured and random PFSPs, particularly as we transition from purely random PFSPs (e.g., Taillard's) to more structured PFSPs.

In correlated PFSPs, bottleneck jobs and machines are the primary factors defining the optimal makespan, as they dominate the overall schedule. If distribution means are sampled from the full interval I, then we expect relatively few bottleneck elements. In contrast, if the distribution means are very similar (as they are in Taillard's problems), then we have a large number of interacting bottleneck elements, which makes the scheduling problem more difficult. Using this intuition, we control for expected problem difficulty by randomly sampling the distribution means from a fraction α of I, $0.1 \leq \alpha \leq 1.0$. By varying α, we can gradually transition from random, Taillard-like problems to more structured PFSPs.

We note that 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 overlapping *bottleneck* distributions, leading to our current approach for sampling distribution means.

2.1 The Problem Generator

In Sections 3 - 5, we use a variety of structured PFSPs containing three forms of correlation: job, machine, and mixed. PFSPs with time gradients were not considered, based on Reeves (1995) experimental results. Before introducing the pseudo-code of the problem generator used to create these structured PFSPs, we introduce some notation and document the parameter settings used in our experiments. All operation durations d_{ij} are restricted to the interval $I = [Dur_{LB}, Dur_{UB}]$. Following Taillard and many other PFSP researchers, we take $Dur_{LB} = 1$ and $Dur_{UB} = 99$. For each of the x (either $x = n$ or $x = m$) distributions, a distribution half-width d_i^{hw}, $1 \leq i \leq x$, is uniformly sampled from the interval $[HW_{lb}, HW_{ub}]$. In all of our experiments, we take $HW_{lb} = 1$ 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), and selecting large values of HW_{lb} and HW_{ub} relative to I would prevent generation of 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.

Structured PFSPs are then generated using a three-step process:

Step 1: Determine Distribution Means

Let $Width_{eff} = \alpha \cdot (Dur_{UB} - Dur_{LB})$. 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

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

Determining the d_{ij} for mixed-correlated PFSPs is slightly more complicated. For each job i, a real value mag_i is uniformly sampled from the interval $[0, 1]$. A job i with mag_i near 1 or 0 consistently sample d_{ij} near the upper and lower ends of the distribution D_j, respectively (values near 0.5 sample near the mean of D_j). Letting $width(D_j) = d_j^{ub} - d_j^{lb}$, we then assign the d_{ij} using: $d_{ij} = mag_i \cdot width(D_j) + d_j^{lb}$. In reality, the ranks mag_i 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 $d_{ij} = Dur_{ub}$.

We note that random, Taillard-like PFSPs can be produced using our structured PFSP generator, specifically by setting $\alpha = 0.0$, $HW_{lb} = HW_{ub} = 49$, and $Interval_{st} = 50$.

2.2 Structured Benchmark Problems

In the experiments described in Sections 3 through 5, we use a variety of structured PFSPs. We consider 4 different problem sizes: 20 machines with 20, 50, 100, and 200 jobs (these dimensions correspond to the more difficult problems in Taillard's benchmark suite). For each problem size, we generated 100 instances for each type of correlation (job, machine, and mixed), at each level of $\alpha \in \{0.1, 0.2., , , .1.0\}$ (for a total of 30 structured problem groups at each problem size). Finally, we also generated 100 random PFSPs for each problem size using Taillard's methodology. The total number of problem instances considered is 12400.

In contrast to Taillard (1993), we do not advocate filtering for difficult problems. Instead, we are concerned with algorithm performance on *classes* of problems, as any comparison of different algorithms 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 problem instances with a bi-modal distribution of means, irregardless of α, while Taillard's generator is unlikely to produce any of our structured PFSPs. As mentioned earlier, we hypothesize that the number of bottleneck jobs or machines is the primary factor in determining the difficulty of structured PFSP instances, and our problem generator was produced in part to specifically test that hypothesis.

Finally, both the structured PFSP generator (written in C++) and the problems used in our experiments can be obtained by contacting the first author. While other researchers such as Kan have introduced structured PFSPs, our structured problem generator provides two important contributions: 1) the ability to produce both machine-correlated and mixed-correlated PFSPs (the later is important because many industrial scheduling problems show such a form of correlation) and 2) a mechanism for varying the expected number of bottleneck elements in structured problems (a factor we hypothesize to be correlated with problem difficulty).

3. Critical Path Topology and PFSP Move Operators

Since the early 1990's, the most efficient algorithms for Taillard's PFSP benchmark problems, in terms of both execution speed and solution quality, have been based on local search paradigms. The primary component of any local search algorithm is the move operator. "Knowledge-poor" PFSP move operators perform syntactic manipulations of a solution π to produce neighboring solutions π'. Because no problem-specific knowledge is used in the manipulation process, these operators often generate provably non-improving neighboring solutions. The most successful knowledge-poor PFSP move operator is the shift operator, denoted SH_{op}. 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 $(\frac{n(n-1)}{2})$ 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 the subset of moves which 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 Thomas Stützle's (1999) Iterated Local Search algorithm, which uses SH_{op} to achieve competitive results on Taillard's benchmark 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 PFSP instances, and examine how the different topologies affect the performance of SH_{op} and

NS_{op}.

3.1 Prototypical Critical Path Topologies

Before introducing the notion of a prototypical critical path topology, we briefly review the concept of a critical path in the PFSP. Suppose a particular PFSP instance has a solution π. We denote the earliest starting and completion times of the ith job on machine j by EST_{ij} and ECT_{ij}, respectively. Clearly, $C_{max}(\pi) = ECT_{nm}$. Next, we denote the amount of time from the start of the ith job on the jth machine to ECT_{nm} by $TAIL_{ij}$. An operation o_{ij} is a critical operation (i.e., is critical to the makespan) if and only if $EST_{\pi(i)j}+TAIL_{\pi(i)j} = C_{max}$. 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.

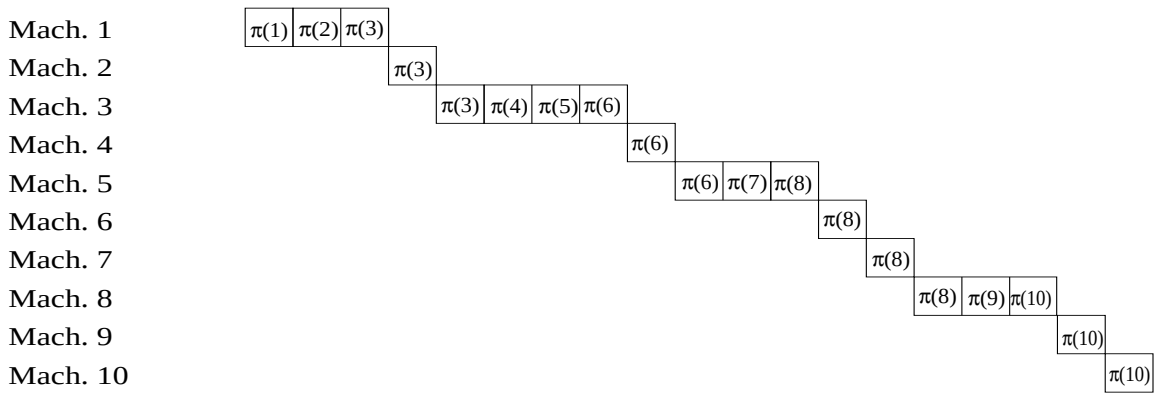

Figure 2: An example critical path extracted from a 10×10 random PFSP, with critical blocks on machines 1, 3, 5, and 8.

Figure 2 shows a critical path extracted from a 10×10 random PFSP. Contiguous sub-sequences of critical operations of size ≥ 2 are present on machines 1, 3, 5, and 8. These sub-sequences are known as *critical blocks* (technically, sub-sequences of size 1 are also critical blocks, but because NS_{op} ignores them and Property 1, below, is not applicable to these critical blocks, we restrict attention to blocks of size ≥ 2). 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 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 the first job in π is additionally included in X.*

If $i = L$ and B_L resides on machine m, then the last job in π is additionally included in X. Then any re-ordering of the jobs in X cannot reduce $C_{max}(\pi)$.

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 PFSP move operators is primarily influenced by the size of the blocks in the critical path; 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 is still heavily influenced by the critical path topology.

Despite the large influence of critical path topology on move operator performance, no studies have characterized the critical path topologies of random, let alone structured, PFSPs. Toward this goal, we define models of 'prototypical' critical path topologies for our structured PFSPs. In the simplest form of job-correlated and machine-correlated PFSP, the critical path is induced by a single bottleneck job or machine, respectively. We were unsure whether a single bottleneck job or a single bottle machine would induce the critical path for mixed-correlated PFSPs, so we did not define a mixed-correlated prototypical topology. Finally, for reasons discussed below, we were unable to develop a prototypical critical path topology for random PFSPs.

Figure 3 shows the prototypical critical path topology for a job-correlated PFSP. There are two critical blocks, appearing on machines 1 and m, and a single bottleneck job: $\pi(5)$. For modeling purposes, we assume the two blocks contain $(\frac{n}{2})$ and $(\frac{n}{2})+1$ jobs, respectively; a single job is shared between the two critical blocks. Figure 4 shows the prototypical critical path topology for a machine-correlated PFSP. There is a single bottleneck machine: $m = 5$. For modeling purposes, we assume the bottleneck machine is not equal to either 1 or m.

Next, we studied the critical path topologies of real random and structured PFSPs. For each of the problem groups described in Section 2.2, we used independent runs of the NEH-RS algorithm (described in Section 5.1) to produce 100 solutions to each problem instance in the group, and for each solution we extracted a random critical path and recorded the size of each critical block. We used NEH-RS to generate representative solutions because 1) it produces reasonable solutions in a short period of time and 2) we found no evidence that the critical path topology of a particular group varies significantly with solution quality.

Figures 5 and 6 show histograms of the critical block sizes (for all 100 samples of all 100 instances) for the 50×20 problem groups; the corresponding histograms from different-sized

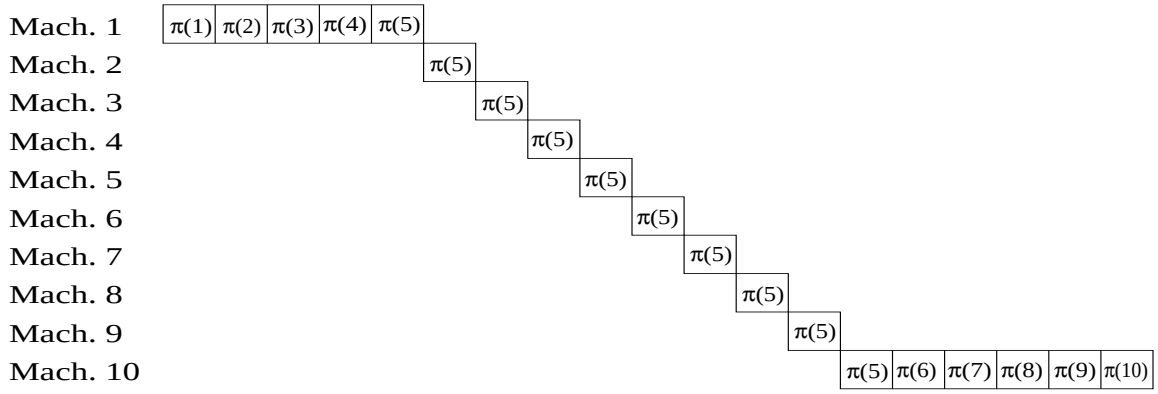

Figure 3: Prototypical critical path for a 10×10 job-correlated PFSP. The bottleneck job is $\pi(5)$.

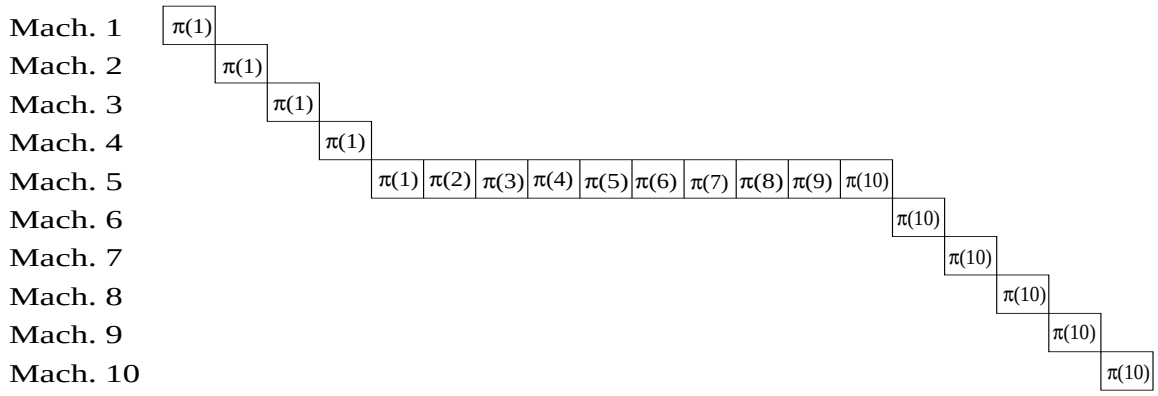

Figure 4: Prototypical critical path for a 10×10 machine-correlated PFSP. The bottleneck machine is 5.

problem groups were very similar, except where noted below. The critical paths of random PFSPs appear to be dominated by large numbers of small critical blocks, although the right-tail density is non-negligible, and becomes more prevalent as problem size is increased. Upon closer examination, we found that two types of critical path topologies were present: 1) a critical path composed strictly of very small critical blocks and 2) a critical path composed of a very large critical block, in addition to a few smaller ones. However, consider variance in the number and sizes of critical blocks for each type prevented us from accurately modeling the critical path topology of random PFSPs.

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 (and at $\alpha = 0.2$ for the 20×20 problems). The right-most peak corresponds to block sizes of approximately $\frac{n}{2} \approx 25$ for $n = 50$, matching the critical blocks found in

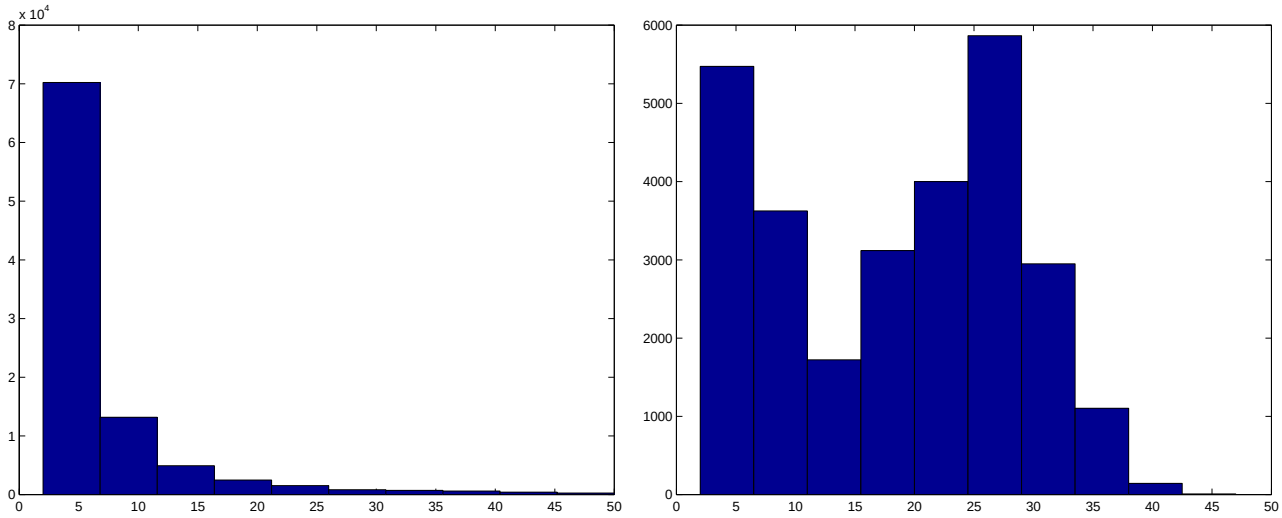

Figure 5: Histogram of critical block sizes for 50×20 random (left figure) and job-correlated (right figure) problem groups. For the job-correlated problem group, $\alpha = 1.0$. Entries represent the critical block sizes of 100 solutions to each of 100 problem instances.

our prototypical job-correlated critical path topology. However, there are a large number of both very small critical blocks (≤ 5), in addition to a number of critical blocks of size 15-20 and $25 - 35$ The latter suggests that the equal-size critical block assumption is frequently violated in real job-correlated PFSPs. Both factors conspire against accurate modeling of the critical path topology of job-correlated PFSPs. Further, we note that the bi-modality for the 100×20 problems appears much later, at $\alpha = 0.9$, and never appears in the 200×20 problems; because the critical block histograms for these problems are very similar to those for random PFSPs, we conjecture that these job-correlated PFSPs will induce similar algorithm behaviors as random PFSPs.

In contrast to results for job-correlated PFSPs, the critical path topologies of the 50×20 machine-correlated problem group are very similar to our prototypical machine-correlated topology, even at $\alpha = 0.3$. The sole discrepancy is the large number of critical blocks of size ≤ 5, the proportion of which decreases (but never vanishes) as $\alpha \to 1.0$. As suggested by our prototypical machine-correlated topology, the critical paths of machine-correlated PFSPs 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 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 problem group tend to mirror the prototypical machine-correlated topology. The influence of job-correlation in

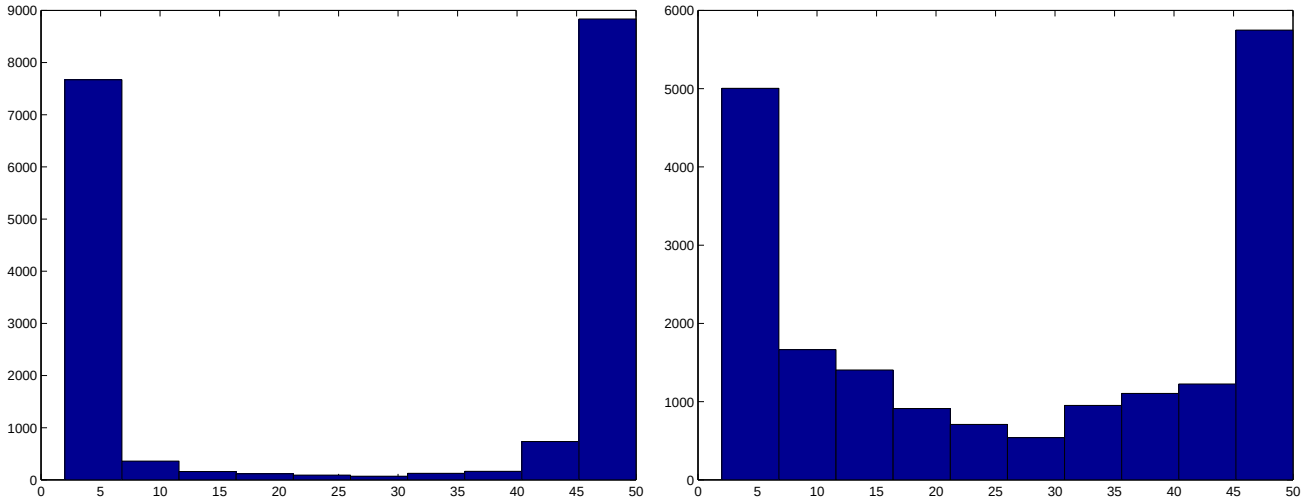

Figure 6: Histogram of critical block sizes for 50×20 machine-correlated (left figure) and mixed-correlated (right figure) problem groups. For both problem groups, $\alpha = 0.3$. Entries represent the critical block sizes of 100 solutions to each of 100 problem instances.

these problems is evident in increased numbers of critical blocks of size 6 through 44, 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). Based on these observations, we conjecture that mixed-correlated and machine-correlated PFSPs will induce similar algorithm behaviors.

3.2 Critical Path Topology and SH_{op}

Next, we examine the influence of critical path topology on the number of ineffective moves (those for which Property 1 is applicable) under SH_{op}.

3.2.1 Definition of SH_{op}

Consider all pairs of distinct job positions (x, y) in a solution π, subject to the restriction $y \neq x - 1$. Under the SH_{op} move operator, 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 $\pi' = (\pi(1), ..., \pi(y-1), \pi(x), \pi(y), ..., \pi(x-1), \pi(x+1), ..., \pi(n))$. Given a problem with n jobs, the number of moves under the SH_{op} operator is $(n-1)^2$, and is independent of the critical path topology.

3.2.2 Effect of Critical Path Topology on SH_{op}

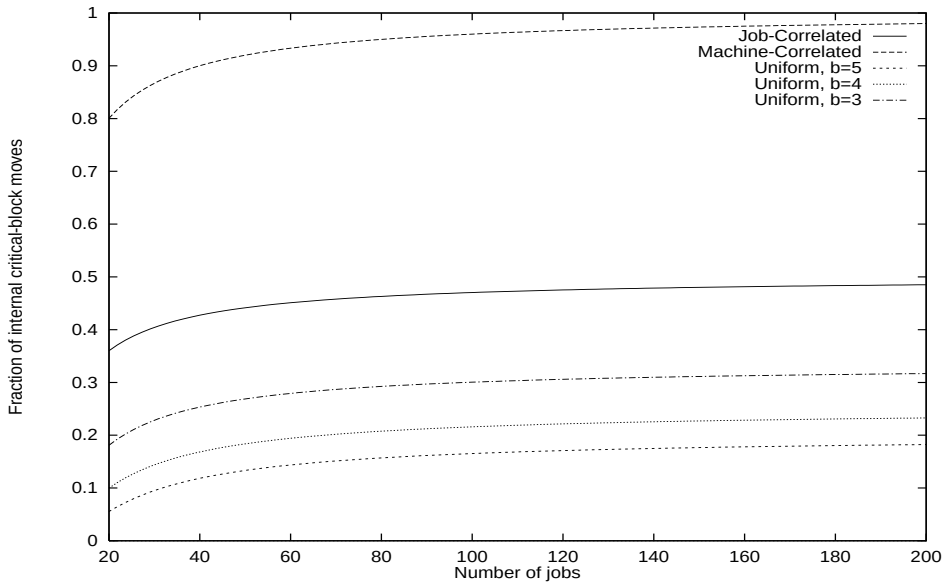

Figure 7: The fraction of moves ($f_{int} = \frac{|CBINT(\pi)|}{(n-1)^2}$) under SH_{op} that are internal to a critical block, under the prototypical critical path topologies. For the uniform critical path topology, f_{int} are shown for $b = 3$, $b = 4$, and $b = 5$.

Now, we 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 each of the 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" critical path topology with b equal-sized critical blocks. As discussed in Section 2, this does not reflect the topology of random PFSPs - we consider this topology to study the influence of the number of critical blocks on $|CBINT(\pi)|$. For each of these topologies, $|CBINT(\pi)|$ is given by:

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

Based on these equations, Figure 7 shows the fraction of total moves under SH_{op}, denoted f_{int}, that are internal to a critical block (and are therefore non-improving) for each prototypical critical path topology. Of particular interest is the strong sensitivity of f_{int} to the number of critical blocks b (note that the prototypical machine-correlated and job-correlated critical path topologies have 1 and 2 critical blocks, respectively). The implications are clear: if our prototypical critical path topologies are representative of the critical paths

found in real-world PFSPs, then SH_{op} can waste anywhere between 45% and 95% of its time evaluating useless moves.

The more interesting question is "How well do these estimations reflect the critical path topologies encountered in real structured PFSPs?". For each of our problem groups, we used independent runs of NEH-RS to produce 100 solutions to each problem instance in the group, and then computed f_{int} for each problem instance using a randomly extracted critical path. The mean f_{int} values, averaged over all 100 critical paths from all 100 instances, are shown in Tables 1 and 2 for random and structured PFSPs, respectively.

Table 1: The fraction of SH_{op} moves that are non-improving (internal to a critical block) for random PFSPs (f_{int}).

	Problem Size			
	20×20	50×20	100×20	200×20
f_{int}	0.11	0.19	0.21	0.31

Table 2: The fraction of SH_{op} moves that are non-improving (internal to a critical block) for structured PFSPs (f_{int}). The mean observed values are reported for $\alpha = 0.1$ through $\alpha = 1.0$. The last column shows the f_{int} values for the corresponding prototypical critical path topology, where available.

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

First, we consider the f_{int} observed for random PFSPs (Table 1). Clearly, f_{int} increases with increases in problem size, which is consistent with the observation that in random PFSPs, critical path topologies often consist of relatively large critical blocks. The correlation

between the increase in the number of jobs and f_{int} is also consistent with the general observation that local search algorithms based on SH_{op} tend to scale worse than those based on NS_{op}; for $n = 200$, any algorithm using SH_{op} is likely to waste over 30% of its computation time evaluating provably non-improving moves.

Next, we compare the observed f_{int} for our structured PFSPs with the f_{int} computed for the prototypical critical path topologies. We had no a-priori notion of a prototypical critical path topology for mixed-correlated PFSPs. As intuition suggests, the f_{int} for mixed-correlated PFSPs falls somewhere between those for job 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 $n = 200$, $0.5 \leq \alpha \leq 1.0$).

Machine-correlated PFSPs show a relatively strong agreement between the observed f_{int} values and those computed for the prototypical topology, and the agreement strengthens as α is increased. These discrepancies are due primarily 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 f_{int} observed for job-correlated PFSPs were substantially less than the f_{int} computed for the prototypical topology. Similarly, these discrepancies are due primarily to the unequal block sizes present in many real job-correlated PFSPs; the contribution due to the smaller critical blocks is negligible.

The f_{int} observed for structured PFSPs are generally much larger than those observed for random PFSPs; for a large number of machine and mixed-correlated PFSPs, over 80% of the moves are provably non-improving. A relatively recent trend in PFSP research 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} (Ben-Daya and Al-Fawzan 1998) (Stützle 1999). We feel this trend has been successful in part because researchers use random benchmark problems, 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}

The NS_{op} move operator also uses shifts to produce neighbors, but only considers a subset of the moves that have the potential to immediately improve C_{max}. Consider a job $\pi(i)$ which is part of 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 harder to implement than SH_{op}. No researchers other than ourselves and Reeves/Yamada 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 $B_i(left)$ and $B_i(right)$ denote the positions of the first and last jobs in the block B_i, respectively. By convention, $B_0(left) = B_0(right) = 1$ and $B_{l+1}(left) = B_{l+1}(right) = n$. Note that $\forall i$, $1 \leq i \leq l$, $B_i(right) = B_{i+1}(left)$ and $B_i(left) = B_{i-1}(right)$. Finally, let $mach(B_i)$ 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 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 use the ϵ parameter to control the neighborhood size for larger random PFSPs. Given an n-job and m-machine problem instance, 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). These rules were derived using Taillard's random PFSP benchmark problems; 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, the two critical blocks are on machines 1 and m and contain $\frac{n}{2} - 1$ and $\frac{n}{2}$ 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 heavily depends on ϵ. The number of moves $|NS_{op}|$ for the prototypical job-correlated critical path is therefore given by:

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

In the prototypical machine-correlated critical path, 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:

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

Empirically, the growth in $|NS_{op}|$ for random PFSPs mirrors than of job-correlated PFSPs - quadratic when $\epsilon = 1.0$ and $\epsilon = 0.5$, and linear when $\epsilon = 0.0$ - although the number of observed moves in each case was significantly less. Under the 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 $|NS_{op}|_{JC}$ and $|NS_{op}|_{RND}$ are quadratic in n, making 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.

Next, we computed $|NS_{op}|$ for real random and structured PFSPs. Following the methodology in the previous section (3.2.2), we computed $|NS_{op}|$ values for each of the problem groups described in Section 2.2 using NEH-RS. Tables 3 and 4 show the mean $|NS_{op}|$, averaged over 100 solutions for each of the 100 problems of each group, for random and structured PFSPs, respectively.

Table 3: The mean number of moves under NS_{op} ($|NS_{op}|$) for random PFSPs.

	Problem Size			
	20×20 ($\alpha = 1.0$)	50×20 ($\alpha = 0.5$)	100×20 ($\alpha = 0.0$)	200×20 ($\alpha = 0.0$)
($\|NS_{op}\|$)	110.0	264.8	179.6	365.2

Table 4 shows the $|NS_{op}|$ for structured PFSPs. Independent of α, the observed values for $n = 100$ and $n = 200$ machine-correlated PFSPs only slightly under-cut the values computed for the machine-correlated prototypical critical path topologies. For $n = 20$ and $n = 50$, the observed values are significantly larger; the discrepancy diminishes as $\alpha \to 1.0$, although it never disappears. Here, the discrepancy is due to the existence of small critical blocks in addition to the large dominant block. For $\epsilon = 0.5$ or $\epsilon = 1.0$, the existence of even a single small critical block in addition to the large critical block can significantly increase the number of moves, as each internal job in the large critical block has more than 2 available moves.

Table 4: The number of moves under NS_{op} for structured PFSPs ($|NS_{op}|$). The mean observed values are reported for $\alpha = 0.1$ through $\alpha = 1.0$.The last column shows the $|NS_{op}|$ values for the corresponding prototypical critical path topology, where available.

	α										
Correlation Type	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	Proto.
20×20											
Job $\epsilon = 1.0$	128.6	146.3	151.9	157.8	157.7	162.2	162.1	167.5	168.7	172.1	201.0
Machine $\epsilon = 1.0$	107.0	94.0	88.5	75.6	71.4	68.7	62.7	64.0	55.0	56.6	38.0
Mixed $\epsilon = 1.0$	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 $\epsilon = 0.5$	271.9	300.2	307.8	356.3	394.4	400.0	412.6	419.9	433.8	456.8	626.5
Machine $\epsilon = 0.5$	196.3	173.1	140.1	155.4	145.7	116.0	123.9	115.7	111.9	122.7	98.0
Mixed $\epsilon = 0.5$	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 $\epsilon = 0.0$	175.0	166.7	155.7	142.0	133.9	130.4	125.6	123.5	118.6	117.9	101.0
Machine $\epsilon = 0.0$	183.1	190.2	187.2	187.4	188.5	183.2	189.9	185.3	183.3	187.0	198.0
Mixed $\epsilon = 0.0$	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 $\epsilon = 0.0$	360.8	343.3	333.1	321.4	316.2	300.0	291.4	284.0	271.6	267.4	201.0
Machine $\epsilon = 0.0$	379.3	379.2	374.1	368.9	379.8	373.2	382.4	378.6	379.9	383.1	398.0
Mixed $\epsilon = 0.0$	379.0	379.0	383.5	376.5	384.1	383.4	379.7	377.0	381.8	380.1	N/A

For the larger job-correlated PFSPs ($n = 100$ and $n = 200$), the observed $|NS_{op}|$ are larger than the values computed for the prototypical job-correlated critical path topologies, while for the smaller problems ($n = 20$ and $n = 50$), the values are less. In both cases, the difference diminishes as $\alpha \to 1.0$, and the discrepancy is primarily due to the two large critical blocks having non-equal size; ϵ then influences whether or not a larger or smaller number of moves is generated relative to the number under the prototypical job-correlated topology (in which the two critical blocks are of equal size). The improved accuracy of the predicted values as $\alpha \to 1.0$ is due to the gradual appearance of the bi-modal critical block distribution as $\alpha \to 1.0$.

The results for the larger mixed-correlated PFSPs ($n = 100$ and $n = 200$) closely mirror the results for machine-correlated PFSPs, which is expected given the similarities in their observed critical path topology. However, for $n = 20$ and $n = 50$, the observed values for mixed-correlated PFSPs are much larger than the corresponding values for machine-correlated PFSPs. Here, the discrepancy is due to the non-negligible number of mid-sized critical blocks in real mixed-correlated PFSPs.

Finally, we consider the $|NS_{op}|$ for random PFSPs, shown in Table 3. For $n = 20$ and $n = 50$ (where $\epsilon = 1.0$ and $\epsilon = 0.5$, respectively) random PFSPs induce slightly more

moves under NS_{op} than machine-correlated and mixed-correlated PFSPs, but significantly less than job-correlated PFSPs. In contrast, for $n = 100$ and $n = 200$ (where $\epsilon = 1.0$) random PFSPs induce fewer moves than machine-correlated and mixed-correlated PFSPs, but more than job-correlated PFSPs. The results in this section establish that critical path topologies strongly influence $|NS_{op}|$, and thereby have the potential to impact any local search algorithm employing NS_{op} as the move operator.

4. Search Space Topology of Structured PFSP Instances

By examining the effect of critical path topology on move operator performance, we only obtain an indirect characterization of the underlying search space topology. More direct methods characterize the *fitness landscape*, which is a labeled graph induced by the combination of a problem instance and move operator. For the PFSP, the vertices represent job permutations π and are labeled by $C_{max}(\pi)$, and two vertices representing the solutions π and π' are linked by an edge if and only if π' can be produced by a single application of the move operator (either SH_{op} or NS_{op}) from π. The performance of a local search algorithm (i.e., tabu search or simulated annealing) depends entirely on its ability to efficiently traverse the fitness landscape.

4.1 Characterizing the PFSP Fitness Landscape: Local Optima and the Big-Valley

Relatively little research has been devoted to analyzing the fitness landscapes of the PFSP. Reeves (1995) and Reeves and Yamada (1998) have shown that the local optima of Taillard's benchmark problems are distributed in a "big-valley" under both the SH_{op} and NS_{op}. In a big-valley distribution: 1) local optima are closer together than randomly chosen solutions, 2) better local optima tend to be closer to global optima, and 3) local optima near one another have similar evaluations. First introduced in the context of the Traveling Salesman and Graph Bipartitioning problems (Boese et al. 1994), and later used by Jones and Forrest (1995) to predict the difficulty of test problems for genetic algorithms, the big-valley concept has been used to explain PFSP problem difficulty. Some recent PFSP algorithms such as the path relinking algorithm of Reeves and Yamada (1998) are explicitly designed to exploit such a distribution.

To test for the presence of a big-valley distribution, a number of local optima are generated using a particular move operator and a relatively simple local search algorithm. Most researchers use next-descent or steepest-descent, although Reeves and Yamada (1998) used a more complex fixed-temperature simulated annealing algorithm. In our experiments, we generate 1000 local optima using a next-descent search algorithm (and either SH_{op} or NS_{op}, depending on the context) which is initiated from a random permutation π. Neighbors with equal makespans are accepted, and the next-descent algorithm is applied for a maximum of 5000 iterations. Further, the next-descent algorithm visits neighbors in a randomized order.

Given a set of local optima, we next measure the distance of each local optimum to the global optimum ($dist_{opt}$) and produce a scatter-plot of $dist_{opt}$ versus C_{max}. Positive correlation (as measured either directly or through a randomization test (Reeves and Yamada 1998)) is taken as evidence of a big-valley distribution. If the global optimum is unknown, $dist_{opt}$ is often replaced by the average distance to all other local optima $dist_{avg}$ (Reeves and Yamada 1998). Because the global optima for our structured PFSP instances are unknown, we apply the latter methodology in the remainder of this section.

The distance between two local optima is ideally the minimal number of steps required by a move operator to transform one of the local optima into the other. However, computation of these measures is often intractable, and operator-independent measures are typically used in their place. In the PFSP, all solutions are simply permutations π of the integers 1 through n, and a straightforward operator-independent precedence-based measure is often used to compute pairwise distances. Given two permutations π and π' of length n, the distance $d(\pi, \pi')$ is given by:

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

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

4.2 Local Optima Distributions of Random Problems

In Figure 8, we show the scatter-plots for a random 50×20 problem instance, where local optima were produced using both NS_{op} and SH_{op}. Both move operators produce prototypical examples of a big-valley optima distribution: positive correlation was confirmed using the randomization test described in (Reeves and Yamada 1998). The local optima are clearly

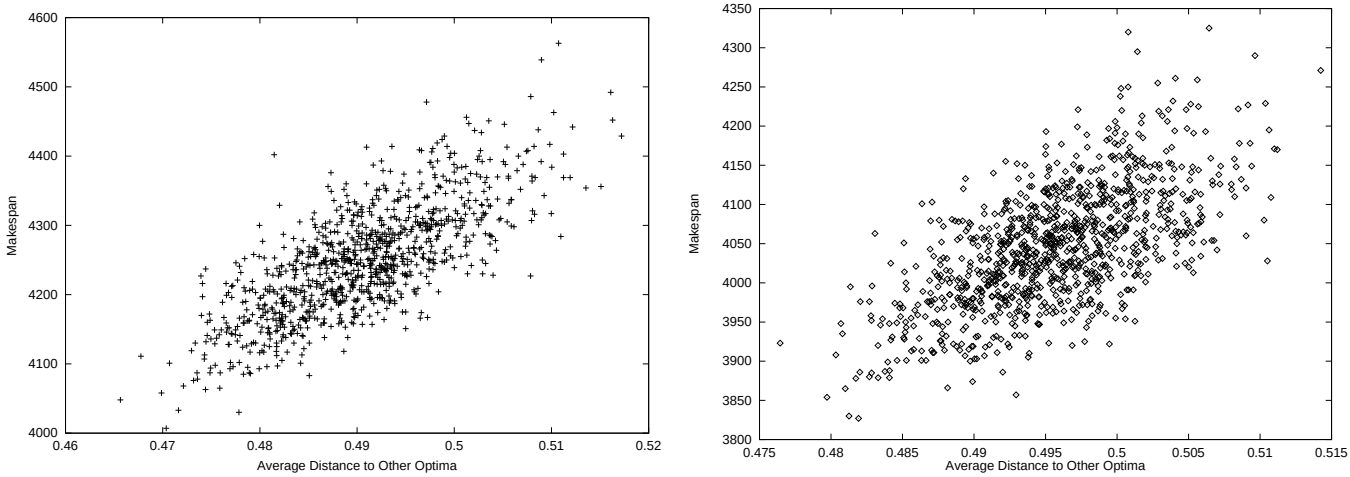

Figure 8: Distribution of 1000 local optima for a randomly generated 50×20 problem instance. The left and right figures show results for NS_{op} and SH_{op}, respectively.

clustered, becoming more compact as better makespans are achieved. Similar results were obtained for all 50×20 instances from Taillard's benchmark, in addition to nine other random 50×20 instances we generated.

Many researchers have argued that big-valley distributions at least partially explain the success of many local search algorithms. Suppose that a particular move operator induces a big-valley distribution on the local optima of a given problem instance. Under this distribution, good local optima tend to be clustered. Further, the cluster becomes tighter for better local optima. If a local search algorithm manages to find a good local optimum, then it is argued that investigation of nearby points (as all local search algorithms do) will likely yield better, or at least equally good, local optima. By iterating this reasoning, a big-valley distribution should yield a direct path to the global optimum.

Finally, we note that although both NS_{op} and SH_{op} induce the big-valley distribution, the empirical evidence is clear that NS_{op} outperforms SH_{op} on random problems. This anomaly can be partially, although not completely, explained by the fact that SH_{op} wastes computation evaluating provably non-improving moves.

4.3 Local Optima Distributions of Structured Problems

We next tested for the presence of the big-valley distribution in structured PFSPs. Figure 9 shows the scatter-plots for a 50×20 job-correlated problem at $\alpha = 0.1$. The left-hand side of Figure 9 clearly demonstrates the continued presence of the big-valley distribution under NS_{op}, as confirmed by a randomization test. For SH_{op}, although remnants of the

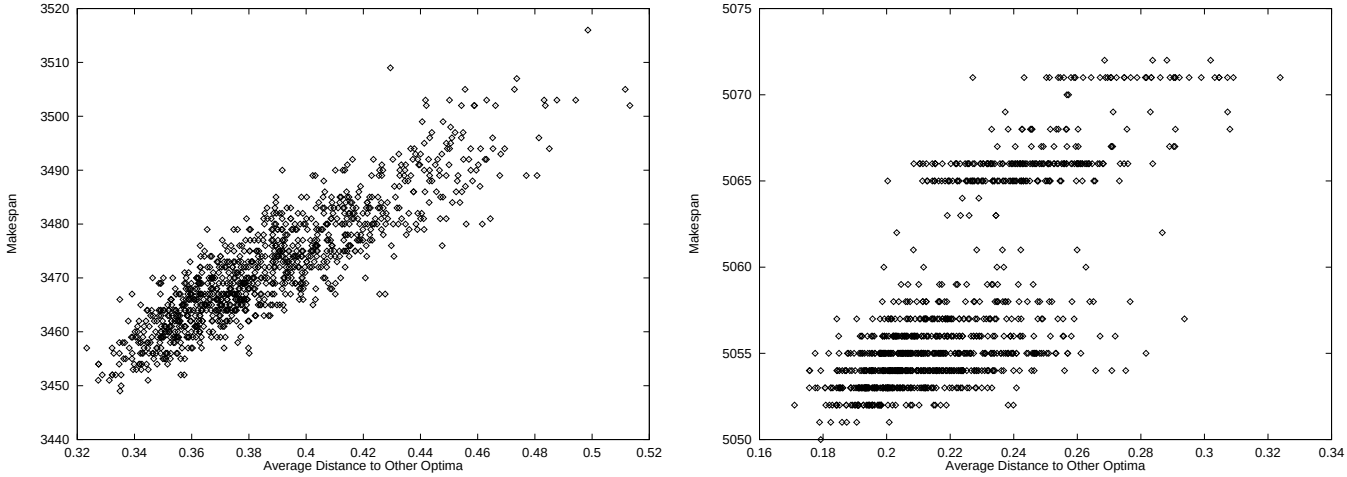

Figure 9: Distribution of 1000 local optima for a job-correlated 50×20 problem instance ($\alpha = 0.1$). The left and right figures show results for NS_{op} and SH_{op}, respectively.

big-valley structure exist, another dominant structural feature begins to emerge: plateaus of local optima at specific makespan values.

We also generated scatter-plots for several additional 50×20 instances of job, machine, and mixed-correlated problems, varying α from 0.1 to 1.0 in increments of 0.1. For SH_{op}, we found similar results for all types of structure. Additionally, as α is increased we find 1) the number of plateaus diminishes and 2) the number of local optima not belonging to a plateau drops rapidly to zero. We also saw the plateau phenomenon for NS_{op} on both machine and mixed-correlated problems, and to a lesser extent on job-correlated problems with larger α.

The existence of plateaus under SH_{op} is predictable, given that large critical blocks dominate the critical path topology of structured PFSPs, and these plateaus clearly have the potential to significantly impact the performance of any local search algorithm which uses SH_{op}. The plateaus typically contain huge numbers of solutions - the challenge of a local search algorithm is to find an exit: a solution which is not part of the plateau. The next-descent procedure used to generate the local optima of Figure 9 fails this challenge; either it is not possible to escape the plateaus without temporarily accepting non-improving moves, or there are improving moves which exit the plateau, but they are rare enough that a randomized next-descent algorithm is unlikely to find them. We conjecture simulated annealing would likely yield a similar fate. In contrast, it seems plausible that tabu search mechanisms might focus search in a single direction, moving off the plateau rather quickly. We empirically test the latter hypothesis in Section 5. Finally, in contrast to SH_{op}, the existence of plateaus under NS_{op} was entirely unexpected, as NS_{op} only considers those

moves that can potentially improve C_{max}. We currently do not have an explanation for this phenomenon, although we discuss possible explanations in Section 5.

5. Algorithm Performance on Structured PFSP Instances

In the previous two sections, we contrasted the search space topologies induced by PFSP move operators for both random and structured PFSPs. In this section, we explore the issue of relative algorithm performance on both random and structured PFSPs.

5.1 The Algorithms

We consider two constructive and two local search algorithms to study our random and structured PFSP instances. NEH is generally accepted as the best heuristic constructive algorithm. It is inexpensive (requiring less than 1 second on any of the PFSP instances we consider), and serves as a baseline algorithm in our study. NEH-RS provides improved performance over NEH through multiple iterations of the core constructive algorithm. Finally, we consider tabu search algorithms using the SH_{op} and NS_{op} move operators. The state-of-the-art PFSP algorithm is a tabu search algorithm using NS_{op}. A tabu search algorithm using SH_{op} was one of the previous contenders.

5.2 NEH and NEH with re-starts

The NEH (Nawaz-Enscore-Ham) algorithm (Nawaz et al. 1983) is widely regarded as the best constructive heuristic algorithm for the PFSP (Taillard 1990). In its most basic form, NEH is a simple greedy procedure, summarized as follows:

(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 first two jobs in the order which minimizes the makespan of the partial schedule.
(3) For k=3 to n do

Insert the k-th job into the location in the partial schedule, among the k possible, which minimizes the makespan of the partial schedule.

Despite its simplicity (time complexity $O(n^2m)$), NEH is known to produce reasonably good solutions to Taillard's benchmark problems in a very short period of time.

The basic NEH algorithm can be extended in a variety of ways. We consider an iterative version in which two distinct *random* jobs are selected in Step (2) of the basic algorithm. We

denote this algorithm by *NEH-RS*, for NEH with Random Re-starts. NEH can also serve as the basis of a branch-and-bound algorithm, an alternative we did not pursue.

5.3 Tabu search algorithms: $tabu_{sh}$ and $tabu_{ns}$

We independently implemented Nowicki and Smutnicki's tabu search algorithm, which uses NS_{op}, and verified their results on Taillard's benchmark problems. In our experiments, we use the original parameter settings, as documented in (Nowicki and Smutnicki 1996). We denote this algorithm by $tabu_{ns}$. A tabu search algorithm using SH_{op} was introduced by Taillard (1990) and has been subsequently improved (Ben-Daya and Al-Fawzan 1998). Many of the basic ideas in these implementations are also incorporated into $tabu_{ns}$ (such as diversification and intensification schemes). Therefore, we chose to use SH_{op} in conjunction with the tabu search component of $tabu_{ns}$, primarily in order to control for performance differences attributable to discrepancies in the tabu search mechanisms. The resulting algorithm is denoted by $tabu_{sh}$.

To produce $tabu_{sh}$, a single modification was made to the tabu search mechanism of $tabu_{ns}$ (in addition to replacing NS_{op} with SH_{op}), based on a performance enhancement reported by Stützle (1999). Normally, $tabu_{ns}$ generates all neighboring moves, categorizing them as either non-tabu, tabu-but-aspired, or tabu; $tabu_{ns}$ then takes the best non-tabu or tabu-but-aspired move. Instead, we generate the possible moves from a solution π by iteratively generating the moves for each possible job in sequence (i.e., all moves for job $\pi(1)$, followed by all moves for job $\pi(2)$, etc.). After generating the moves for a job $\pi(i)$, if a non-tabu or tabu-but-aspired move with a better makespan than $C_{max}(\pi)$ is found, that move is immediately accepted; no other moves are evaluated.

5.4 Methodology

To study relative PFSP algorithm performance, we ran a single trial of each PFSP algorithm in a factorial experiment. The independent variables are problem size (20×20, 50×20, 100×20, or 200×20), structure type (random, job-correlated, machine-correlated, or mixed-correlated), structure level (α), and algorithm (NEH, NEH-RS, $tabu_{sh}$, or $tabu_{ns}$). The sole dependent variable was the best value of C_{max} achieved by each algorithm on each problem instance.

We acknowledge the well-known problems associated with reporting results from a single trial of a stochastic algorithm. However, the time required for relatively short runs ($5 - 10$

minutes) of the 4 algorithms on $12,400$ problem instances is $\approx$ 200 CPU days, making multiple trials impractical. Interestingly, the large variance in results obtained on random PFSP instances is not necessarily seen on structured PFSP instances. For example, we examined the variance of NEH-RS over 10 trials over a wide range of structured PFSP instances and observed negligible variance in all cases. Variance with $tabu_{ns}$ and $tabu_{sh}$ was higher, although it was much less than that observed for random PFSP instances.

All experimental trials were executed on SUN Ultra 10/400 workstations, each with 2 Gb of RAM, running the Solaris 2.8 operating system. All experimental code was written 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 the problems we considered), we allocated 5 minutes of CPU time to the 20×20 and 50×20 instances, and 10 minutes of CPU time to the 100×20 and 200×20 problem instances. Normally, we do not advocate using limits on CPU time in algorithmic comparisons. However, in the case of the PFSP, all of the algorithms we consider share the following characteristics: 1) run-times are dominated ($> 99\%$) by either neighborhood evaluations (in the case of $tabu_{ns}$ and $tabu_{sh}$) or greedy evaluation (in the case of NEH and NEH-RS), and 2) in either case the evaluations are produced using the same mix of algorithmic principles (and in our experiments, the same implementation). Finally, we note that we allowed for five times the number of iterations for $tabu_{ns}$ on random 200×20 problem instances than did Nowicki and Smutnicki (1996).

For each problem instance, we also computed two additional measures: the lower bound on C_{max} and the best C_{max} found by the 4 algorithms we considered. For random problems, the straightforward PFSP lower bound used by Taillard (1993) is computationally inexpensive and produces somewhat accurate results. The Taillard lower bound (which is based on a lower bound computation defined by Kan (1976)) is also accurate 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 problems. In a proportionate flow-shop, the 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 problems, we translate each instance into a proportionate flow-shop instance by letting the processing time for each job i equal $min_j d_{ij}$. The lower bound for the job-correlated instance is taken as the optimal makespan for the corresponding proportionate flow-shop instance. Empirically, this approximation yields relatively tight lower bounds for job-correlated instances.

5.5 Results

A problem instance is often labeled as "difficult" if the quality of a solution obtained by good heuristic algorithms is far from computed lower bounds on the solution quality (e.g., (Taillard 1993)). This definition assumes that computed lower bounds are relatively accurate estimates of the optimal solution quality, which is generally the case for our PFSP instances. Thus, our first exploration into the relative difficulty of structured and random PFSP instances is based on comparing the computed lower bounds and the best solution found by any of our algorithms. Table 5 shows the number of instances, out of the 100 possible, in which the best solution was identical to the computed lower bound.

Table 5: The number of problem instances, out of the 100 possible, for which the best solution found by our test algorithms equaled the computed lower bound for our structured problem groups.

	α									
Correlation Type	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20×20 problems										
Job	13	47	63	68	67	79	83	86	90	87
Machine	12	34	34	48	57	57	69	70	68	72
Mixed	1	4	11	15	25	20	37	37	45	45
50×20 problems										
Job	1	9	26	47	65	67	81	79	89	89
Machine	26	46	57	67	66	78	69	83	84	79
Mixed	11	33	48	47	50	55	58	63	75	63
100×20 problems										
Job	0	0	0	5	23	29	49	55	53	70
Machine	29	55	55	68	74	85	90	87	90	85
Mixed	16	42	52	66	60	64	73	76	81	79
200×20 problems										
Job	0	0	0	0	0	0	0	0	2	4
Machine	35	57	66	71	79	84	80	89	86	87
Mixed	30	50	62	63	77	75	77	84	83	87

We hypothesized that as α increases, the difficulty of structured PFSP instances would decrease (because a smaller number of jobs/machines are dominating the schedule). However, we did not anticipate the rapid, non-linear decrease in difficulty seen in much of Table 5. Considering the machine-correlated instances, the optimal solutions to at least half the problems are found for $\alpha \geq 0.2$. Further, the number of such instances 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 problems, although we see the same overall trends. For job-correlated instances, we see similar patterns for the 20×20 problems. However, the number

of optimal solutions actually *decreases* with increases in problem size. Interestingly, this contradicts Taillard's conjecture that for a fixed m, $\lim_{n\to\infty} Prob(C^*_{max} = LowerBound) = 1.0$, although our results support the conjecture for both machine-correlated and mixed-correlated problem instances. Finally, we note that the lower bound was *never* reached for any of our random problem groups (this is consistent with Taillard (1993), page 281).

In summary, the results in Table 5 establish that a large number of structured PFSPs are actually very easy to solve to optimality. In contrast, for random problems, the best solutions were generally very far from the computed lower bounds. For both machine and mixed-correlated instances, the majority of instances are relatively easy once α is between 0.2 and 0.3, and become easier at larger problem sizes. We saw a similar dependency on α for small job-correlated instances, but the effect dissipated with increases in problem size. We finally note that comparing the best-known solutions with lower bounds can only *establish* that a problem is easy - the method cannot be used to demonstrate that a problem is difficult. Thus, our 200×20 job-correlated PFSPs *may* be easy, but the inaccuracy of the lower bound for this problem size might prevent us from establishing that fact.

In the preceding discussion, we defined problem difficulty in terms of both the lower bound and the best solution obtained by *any* of our test algorithms. In doing so, we necessarily avoided the question of "What is the relative performance of our algorithms on structured PFSPs?". In the next step of our analysis, we answer this and several related questions regarding relative PFSP algorithm performance.

Table 6: For each random problem group, the number of instances, out of the 100 possible, for which each test algorithm found the best overall solution (the first entry in each cell) and the mean relative error percentage for which the test algorithm failed to find the best overall solution (the second entry in each cell).

NEH	NEH-RS	$tabu_{sh}$	$tabu_{ns}$
20×20			
0/3.87	1/1.55	99/0.09	96/0.04
50×20			
0/5.09	0/2.94	16/0.72	87/0.20
100×20			
0/4.28	0/2.63	0/1.42	100/0.00
200×20			
0/2.94	0/1.80	0/1.38	100/0.00

Tables 6 - 10 show both 1) the number of instances (out of the 100 possible) for which each of our test algorithms achieved the best solution found by *any* of the test algorithms and

2) the mean percentage above the best-known solution on the remaining instances. First, we consider the performance of our algorithms on random PFSPs, as shown in Table 6. NEH-RS outperforms NEH, but both are still 1-3% above the best solution found by $tabu_{ns}$, and the pair of algorithms 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 results reported in the PFSP literature. Although $tabu_{sh}$ actually outperforms (though not at a statistically significant level) $tabu_{ns}$ on some 20×20 and 50×20 problems, it encounters severe scaling problems as problem size is increased. In terms of relative error, $tabu_{ns}$ still produces very good solutions when it fails on the smaller instances, while $tabu_{sh}$ scales poorly on larger problems (although it is still better than NEH-RS). The failure of $tabu_{sh}$ to scale was anticipated, as very large critical blocks occur frequently in the larger random PFSPs, substantially increasing the effort expended in evaluating provably non-improving moves.

Table 7: The number of structured 20×20 PFSP instances, out of the 100 possible, for which each test algorithm found the best overall solution (the first entry in each cell) and the mean relative error percentage for which the test algorithm failed to find the best overall solution (the second entry in each cell).

	α									
Type	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20×20 - NEH										
Job	8/0.28	38/0.15	49/0.10	72/0.10	71/0.08	77/0.07	84/0.06	78/0.06	86/0.07	94/0.08
Machine	9/0.49	18/0.30	21/0.23	37/0.26	44/0.20	40/0.21	59/0.20	63/0.21	63/0.16	62/0.18
Mixed	5/0.39	8/0.36	10/0.31	19/0.24	22/0.24	32/0.25	41/0.20	43/0.22	46/0.16	44/0.15
20×20 - NEH-RS										
Job	40/0.11	76/0.08	83/0.06	95/0.07	95/0.05	96/0.04	99/0.03	95/0.04	98/0.04	99/0.04
Machine	32/0.22	54/0.15	56/0.11	72/0.12	79/0.09	71/0.08	84/0.10	88/0.14	75/0.10	84/0.12
Mixed	28/0.18	30/0.14	44/0.13	55/0.10	61/0.09	61/0.09	76/0.09	72/0.09	78/0.07	84/0.07
20×20 - $tabu_{sh}$										
Job	97/0.05	96/0.09	98/0.04	100/0.00	100/0.00	98/0.03	100/0.00	98/0.06	99/0.03	100/0.00
Machine	94/0.06	95/0.09	97/0.04	100/0.00	99/0.04	96/0.05	100/0.00	98/0.07	99/0.03	100/0.00
Mixed	97/0.06	93/0.08	93/0.10	96/0.07	97/0.05	93/0.06	100/0.00	98/0.04	97/0.05	98/0.03
20×20 - $tabu_{ns}$										
Job	98/0.05	98/0.12	100/0.00	100/0.00	98/0.03	99/0.03	100/0.00	100/0.00	98/0.04	100/0.00
Machine	98/0.16	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00
Mixed	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00	100/0.00

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 achieves the best solution 87% of the time in the *worst* case. Clearly, the state-of-the-art performance of $tabu_{ns}$ on random PFSP instances transfers to the more structured PFSP instances. 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

Table 8: The number of structured 50×20 PFSP instances, out of the 100 possible, for which each test algorithm found the best overall solution (the first entry in each cell) and the mean relative error percentage for which the test algorithm failed to find the best overall solution (the second entry in each cell).

	α									
Type	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
50×20 - NEH										
Job	0/0.54	0/0.30	5/0.22	13/0.15	19/0.11	40/0.10	49/0.08	43/0.08	56/0.05	61/0.06
Machine	17/0.23	33/0.15	43/0.15	45/0.11	49/0.11	62/0.13	59/0.08	75/0.07	73/0.07	68/0.11
Mixed	6/0.35	17/0.19	40/0.17	41/0.14	40/0.13	47/0.11	55/0.10	62/0.11	73/0.08	65/0.08
50×20 - NEH-RS										
Job	0/0.24	7/0.13	33/0.09	41/0.05	68/0.06	73/0.05	82/0.04	78/0.04	90/0.03	91/0.05
Machine	41/0.11	66/0.07	68/0.07	75/0.09	77/0.06	87/0.09	87/0.05	88/0.04	92/0.03	83/0.05
Mixed	17/0.19	47/0.11	66/0.10	66/0.07	70/0.05	74/0.04	83/0.04	86/0.07	90/0.03	84/0.03
50×20 - $tabu_{sh}$										
Job	32/0.12	54/0.08	62/0.07	76/0.05	83/0.06	86/0.04	97/0.03	91/0.04	95/0.03	94/0.04
Machine	62/0.09	80/0.08	85/0.06	92/0.09	89/0.06	95/0.06	96/0.04	95/0.08	96/0.05	96/0.04
Mixed	62/0.07	86/0.05	85/0.05	91/0.03	96/0.05	97/0.04	94/0.04	96/0.02	96/0.04	97/0.03
50×20 - $tabu_{ns}$										
Job	86/0.07	87/0.05	94/0.07	91/0.08	94/0.05	95/0.04	99/0.05	93/0.05	96/0.03	93/0.05
Machine	99/0.03	100/0.00	99/0.05	100/0.00	100/0.00	99/0.03	100/0.00	100/0.00	100/0.00	100/0.00
Mixed	94/0.06	93/0.05	98/0.06	99/0.03	99/0.02	99/0.04	100/0.00	98/0.04	99/0.05	100/0.00

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 larger problem sizes. Both NEH and NEH-RS do relatively poor 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. The largest relative error for any algorithm on the machine-correlated and mixed-correlated PFSPs is 0.49%, in contrast with the maximum of over 5% on random PFSPs; most relative errors are only a fraction of 1% percent. Further, the relative errors for all algorithms decrease as $\alpha \to 1.0$.

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 (0.54% in the worst case), and decreased as $\alpha \to 1.0$.

To summarize, we see substantially different algorithmic behavior between random and structured PFSPs. Random PFSPs show a fairly clear distinction in algorithm performance, with a higher mean relative error for the inferior algorithms. In contrast, the mean relative

Table 9: The number of structured 100×20 PFSP instances, out of the 100 possible, for which each test algorithm found the best overall solution (the first entry in each cell) and the mean relative error percentage for which the test algorithm failed to find the best overall solution (the second entry in each cell).

	α									
Type	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
100×20 - NEH										
Job	0/0.46	0/0.38	0/0.31	0/0.25	0/0.19	3/0.15	6/0.11	17/0.09	9/0.08	26/0.07
Machine	17/0.14	46/0.09	39/0.08	58/0.08	65/0.06	70/0.04	76/0.05	65/0.05	71/0.07	77/0.03
Mixed	12/0.25	34/0.11	47/0.09	63/0.06	58/0.09	60/0.05	75/0.05	74/0.04	73/0.05	73/0.04
100×20 - NEH-RS										
Job	0/0.20	2/0.16	5/0.12	9/0.10	7/0.08	23/0.06	82/0.02	55/0.04	55/0.04	64/0.04
Machine	58/0.06	75/0.04	72/0.04	79/0.04	87/0.04	88/0.03	95/0.03	87/0.02	87/0.03	93/0.01
Mixed	24/0.15	63/0.06	70/0.06	81/0.03	78/0.06	89/0.03	90/0.04	89/0.02	89/0.03	94/0.03
100×20 - $tabu_{sh}$										
Job	100/0.0	41/0.06	42/0.06	34/0.05	61/0.05	62/0.04	58/0.04	85/0.03	90/0.05	92/0.02
Machine	56/0.08	72/0.05	68/0.05	85/0.06	85/0.05	84/0.03	90/0.05	83/0.04	91/0.06	86/0.03
Mixed	49/0.06	63/0.03	84/0.03	88/0.02	81/0.03	85/0.02	88/0.02	92/0.02	90/0.03	92/0.02
100×20 - $tabu_{ns}$										
Job	0/0.20	76/0.07	71/0.05	81/0.03	71/0.03	71/0.03	39/0.05	55/0.04	56/0.04	65/0.04
Machine	99/0.08	96/0.03	98/0.09	100/0.00	100/0.00	100/0.00	99/0.01	100/0.00	100/0.00	99/0.01
Mixed	87/0.07	98/0.04	93/0.04	95/0.07	96/0.02	96/0.03	100/0.00	99/0.01	99/0.02	98/0.02

error for all algorithms on structured problems is very low. For machine-correlated and mixed-correlated PFSPs, increases in α created 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 are not representative of the real-world. For example, the PFSP does not consider important aspects of real-world scheduling problems such as job-dependent set-up times. In response, AI and OR researchers have developed algorithms for enhanced scheduling problems which handle these additional features. However, there is an alternative route to realism that has rarely been pursued by the scheduling community: consider problem instances with more realistic features. The primary goal of our research was to study how syntactic 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

Table 10: The number of structured 200×20 PFSP instances, out of the 100 possible, for which each test algorithm found the best overall solution (the first entry in each cell) and the mean relative error percentage for which the test algorithm failed to find the best overall solution (the second entry in each cell).

	α									
Type	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
200×20 - NEH										
Job	0/0.30	0/0.24	0/0.24	0/0.22	0/0.19	0/0.18	0/0.15	0/0.14	0/0.13	0/0.12
Machine	26/0.07	41/0.04	48/0.04	59/0.03	60/0.03	63/0.03	69/0.03	76/0.01	78/0.02	82/0.02
Mixed	17/0.11	42/0.08	53/0.05	52/0.04	66/0.04	63/0.03	71/0.03	79/0.02	73/0.03	86/0.02
200×20 - NEH-RS										
Job	8/0.13	11/0.09	7/0.07	14/0.08	14/0.08	9/0.07	9/0.07	7/0.06	9/0.05	6/0.05
Machine	53/0.05	71/0.02	69/0.02	80/0.02	80/0.02	84/0.02	840.02	93/0.01	90/0.01	92/0.02
Mixed	38/0.08	71/0.07	77/0.03	75/0.03	85/0.03	81/0.01	87/0.02	89/0.01	90/0.02	94/0.01
200×20 - $tabu_{sh}$										
Job	34/0.10	33/0.06	47/0.06	33/0.06	34/0.05	43/0.05	50/0.04	45/0.04	45/0.03	58/0.02
Machine	45/0.04	65/0.03	70/0.03	75/0.03	77/0.02	79/0.03	84/0.02	90/0.01	90/0.02	91/0.02
Mixed	37/0.04	58/0.03	69/0.02	73/0.02	78/0.02	80/0.01	84/0.01	90/0.02	84/0.02	94/0.01
200×20 - $tabu_{ns}$										
Job	68/0.05	63/0.04	53/0.06	61/0.05	60/0.04	56/0.04	55/0.04	62/0.03	61/0.03	47/0.03
Machine	94/0.03	97/0.03	96/0.02	99/0.01	95/0.01	97/0.01	99/0.07	100/0.00	98/0.03	100/0.00
Mixed	93/0.01	94/0.03	95/0.03	97/0.01	98/0.03	99/0.01	97/0.01	99/0.01	98/0.01	99/0.01

distributions of random and structured PFSPs. Given these differences, we also anticipated differences in algorithm performance. While we did observe these 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 easily 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, as evidenced by the fact that it took nearly 30 years of research to solve a well-known, and small, 10-job, 10-machine job-shop benchmark (Carlier and Pinson 1989). However, our results indicate that scheduling problems similar to those encountered in practice are actually relatively easy. Thus, we must view with some skepticism algorithm evaluations based strictly on randomly generated benchmark problems.

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.

References

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

K. D. Boese, A. B. Kahng, and S. Muddu. 1994. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters* **16/2** 101–113.

J. Carlier and E. Pinson. 1989. An algorithm for solving the job-shop problem. *Management Science* **35(2)** 164–176.

J. Carlier and E. Pinson. 1998. Evaluating the performance of tabu search procedures for flow shop sequencing. *Journal of the Operational Research Society* **49** 1296–1302.

M. R. Garey, D. S. Johnson, and R. Sethi. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* **1(2)** 117–129.

J.H. Hooker. 1995. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics* **1** 33–42.

T. Jones and S. Forrest. 1995. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. *Proceedings of the 6th International Conference on Genetic Algorithms*, 184–192.

A. R. Kan. 1976. *Machine Scheduling Problems: Classification, complexity and computations.* Martinus Nijhoff, The Hague.

M. Nawaz, E. Enscore, and I. Ham. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science* **11(1)** 91–95.

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

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

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

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

C. R. Reeves. 1999. Landscapes, operators and heuristic search. *Annals of Operational Research* **86** 473–490.

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

T. Stützle. 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.

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

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

E. Taillard. 1995. Comparison of iterative searches for the quadratic assignment problem. *Location Science* **3(2)** 87–105.

J.P. Watson, L. Barbulescu, A. E. Howe, and L. D. Whitley. 1999. Algorithm performance and problem structure for flow-shop scheduling. *Proceedings of the 16th National Conference on Artificial Intelligence.* 688–695.