Algorithm Performance and Problem Structure for Flow-shop Scheduling

Jean-Paul Watson, Laura Barbulescu, Adele E. Howe, and **L. Darrell Whitley**
Computer Science Department
Colorado State University
Fort Collins, CO 80523
e-mail: {watsonj, laura, howe, whitley}@cs.colostate.edu

Abstract

Test suites for many domains often fail to model features present in real-world problems. For the permutation flow-shop sequencing problem (PFSP), the most popular test suite consists of problems whose features are generated from a single uniform random distribution. Synthetic generation of problems with characteristics present in real-world problems is a viable alternative. We compare the performance of several competitive algorithms on problems produced with such a generator. We find that, as more realistic characteristics are introduced, the performance of a state-of-the-art algorithm degrades rapidly: faster and less complex stochastic algorithms provide superior performance. Our empirical results show that small changes in problem structure or problem size can influence algorithm performance. We hypothesize that these performance differences may be partially due to differences in search space topologies; we show that structured problems produce topologies with performance plateaus. Algorithm sensitivity to problem characteristics suggests the need to construct test suites more representative of real-world applications.

Introduction and Motivation

Algorithms for the permutation flow-shop sequencing problem (PFSP) are typically compared using problems from benchmark test suites available in the OR library (Beasley 1998). Most often, the problems in these test suites are generated by selecting job processing times from a single uniform distribution; the problems are then submitted to a search algorithm for solution. Problems are accepted as "difficult" if the search algorithm has trouble consistently finding a good solution, as measured relative either to the lower bound or to a best known solution.

An underlying assumption of these performance studies is that if an algorithm performs well on difficult synthetic problems, then it will also perform well on scheduling applications. However, real-world problems are *not* random — they typically are characterized by some amount of structure, though it is rarely quantified and categorized. Our research addresses the issue of whether state-of-the-art algorithm performance will hold up on PFSPs that have structural features representative of those found in some real-world problems. For example, consider a circuit board manufacturing line, a real-world problem naturally expressed as a PFSP. Larger circuit boards tend to have more components, and may require longer processing times at each machine on the line. Such jobs have correlated processing times across machines. In contrast, the duration of a circuit board baking process is independent of board size. In this case, the job processing times are similar on a particular machine.

Ideally, we would like to test algorithms on real-world problems. Yet, these are difficult to obtain, incur knowledge acquisition overhead and sacrifice experimental control. Because we have not yet quantified the structure of real-world PFSP problems, we cannot control for the effects of different types and amounts of structure. Thus, test suite generation based on known characteristics of some real-world problems balances experimental control and generality of results.

The No Free Lunch (NFL) theorem for search (Wolpert & Macready 1995) informally states that the mean performance of all search algorithms is identical, independent of the chosen performance measure, when *all* possible (discrete) objective functions are considered. The NFL theorem warns us that better-than-random performance on a subset of problems may not hold for a *different* subset of problems. By relying on benchmark test suites to drive algorithm development, algorithms may be over-fit to their benchmarks.

This paper presents two major results. First, we demonstrate that superior performance on a popular synthetic benchmark test suite fails to transfer to a test suite containing problems with only modest levels of non-random features. Second, we show that the non-random flow-shop problems do not display a search space topology previously associated with difficult problems, thus partially explaining why previously successful algorithms which exploit such structure fail on non-random problems.

Copyright ©1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

A Structured PFSP Generator

Our domain is the well-known n by m permutation flowshop sequencing problem (PFSP) which Garey & Johnson (1979) showed to be NP-hard. Here, n jobs must be processed, in the same order, on each of m machines. Concurrency is not allowed: a machine can only process a single job at a time, and processing must be completed once initiated. Furthermore, machine $j+1$ cannot begin processing a job until machine j has completed processing of the same job. Each job i requires a processing time of d_{ij} on the jth machine. A candidate solution to the PFSP is simply a permutation of the n jobs, π. Given that the first job in π on the first machine begins at time step 0, the makespan of π is defined to be the finish time of the last job in π on the last machine. The objective is to find a permutation π such that the makespan is minimized.

The most commonly used PFSP benchmark problems are those introduced in Taillard (1993) and available through the OR library. In fact, the performance of many PFSP algorithms such as the Reeves/Yamada path relinking algorithm (Reeves & Yamada 1998) are compared *strictly* on these problems. Taillard generated his problems by selecting the processing times 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 a Tabu search algorithm. In expectation, and in contrast to many real-world problems, Taillard's problems contain little or no discernible structural features.

Kan (1976) introduced two methods of creating structure: job correlation and time gradients. In *job-correlated* problems, the processing times of a given job are 'correlated' in the sense that they are sampled from a relatively tight distribution specific to that job. *Time gradients* impose non-random structure by creating a trend in job processing times as a function of machine; processing times on early machines tend to be less than those of later machines. We introduce a third method to create non-random structure: *machine correlation*. In *machine-correlated* problems, the processing times of all jobs on a given machine are sampled from a relatively tight distribution specific to that machine.

We consider only job and machine-correlated PFSPs. Non-random structure is produced by drawing processing times from a number of Gaussian distributions. For job-correlated problems, we use n distributions, one for each job. For machine-correlated problems, we use m distributions, one for each machine. The generation of the x (either $x = n$ or $x = m$) distributions is a three-step process requiring four input parameters. The parameters σ_{lb} and σ_{ub} provide bounds on the distribution standard deviations, while μ_{lb} dictates a lower bound on the distribution means. Finally, α controls the expected degree of distribution overlap. Low degrees of distribution overlap yield PFSPs with more structure because processing times selected from two distributions with little overlap are typically much different. As the degree of overlap increases, the amount of structure decreases; in the limit, the distributions share the same mean, albeit with different standard deviations. Given these parameters, a PFSP is produced as follows:

Step 1: Select the Standard Deviations.
The distribution standard deviations $\sigma_i, 1 \leq i \leq x$, are selected uniformly from the interval $[\sigma_{lb}, \sigma_{ub}]$.

Step 2: Compute the Mean Interval.
The means of the x Gaussian distributions are selected uniformly from a fixed interval. The width of this interval directly influences the degree of distribution overlap. Smaller interval widths reduce the average mean separation.

Approximately 95% of the mass of a Gaussian distribution lies within $\pm 2\sigma$ of the mean. To produce x distributions with little overlap, consider placing the x Gaussian distributions side-by-side along an axis, with overlap only in the 2.5% upper and lower tails of neighboring distributions. The width of this construction is: $canonical_Width = \sum_{i=1}^{x} 4\sigma_i$. We consider a set of distributions to exhibit minimal overlap if and only if each distribution overlaps with approximately the 2.5% upper or lower tails of neighboring distributions. Note that, in general, minimal overlap will *not* be achieved by uniform sampling of the means: an interval of width $canonical_Width$ is the smallest interval in which it is *possible* to achieve the minimal overlap.

In expectation, selecting x distribution means uniformly from an interval of width less than $canonical_Width$ will yield distributions with more overlap. In our generator, $canonical_Width$ is scaled by α, where α is restricted to the interval $[0, 1]$. Lower values of α yield larger overlap in distributions, and therefore processing times.

The bounds on the distribution means are then given by $[\mu_{lb}, \mu_{ub}]$, where

$$\mu_{ub} = \mu_{lb} + \alpha \cdot canonical_Width.$$

Step 3: Select the Distribution Means.
The means μ_i, $1 \leq i \leq x$, of the distributions are uniformly selected from the interval $[\mu_{lb}, \mu_{ub}]$.

For job-correlated PFSPs, the processing times for each job i are selected from the Gaussian distribution $\eta(\mu_i, \sigma_i)$. For machine-correlated PFSPs, the processing times for each machine j are selected from the Gaussian distribution $\eta(\mu_j, \sigma_j)$.

Finally, in contrast to Taillard (1993), we do not filter for difficult problems. Instead, we concern ourselves with algorithm performance on *classes* of problems. Taillard defines difficulty relative to a specific algorithm. Thus, any comparison of different algorithms would be biased by such filtering, as problem difficulty can only be defined relative to an algorithm.

The Algorithms

We compared the performance of algorithms based on three search methodologies: 1) path relinking, 2) incremental construction, and 3) iterative sampling. The

path relinking algorithm by Reeves and Yamada (1998) was selected because it has demonstrated some of the best performance to date on the problems from Taillard's test suite. Most AI search algorithms are based on either incremental construction or iterative sampling. Incremental construction refers to algorithms that use heuristics to build up a schedule; this class was included because excellent heuristics are available for the PFSP domain. Iterative sampling refers to a class of stochastic algorithms ranging from random sampling to random-starts local search; this class was included primarily because of reported successes of such algorithms on various scheduling applications.

Path Relinking

Path relinking is a general search strategy in which the search space is explored by looking for additional optima near two known local optima. During the process of 'linking' or constructing a path between two local optima, the algorithm can check the intervening area for other optima. Path relinking is the basis for the Reeves/Yamada PFSP algorithm (Reeves & Yamada 1998), which we denote by *pathrelink*.

The search spaces for the problems in Taillard's test suite exhibit a 'big-valley' structure (Boese, Kahng, & Muddu 1994). The big-valley structure implies that the best local optima are near to the global optima. Poorer optima tend to be further away from the global optima. Path relinking has been shown to be effective on search spaces with this structure. Iterated exploration linking two highly fit local optima may be likely to expose other good local optima.

At the highest level, *pathrelink* is a steady-state genetic algorithm (Whitley 1989). A population of 15 job permutations, each representing a solution to the PFSP, is maintained; the complexity of the crossover operator forces the small population size. New candidate solutions are generated using one of two methods, each with a fixed probability. The first method uses a stochastic local search operator to improve upon an existing population member, with the goal of moving solutions toward local optima. The second method uses a form of path relinking to find new candidate solutions. First, two parent solutions are selected from the population. Starting from one parent, a path is probabilistically projected toward the other parent. A limit on the number of evaluations allocated to each projection is imposed, and the best solution along the projected path is retained. Due to the stochastic nature of the projection and the limit on the number of evaluations, complete path linking is *not* performed. Rather, search progresses toward the other parent, although not necessarily reaching it. Once a candidate solution is generated via either of these methods, it is placed into the population if the resulting makespan is better than the makespan of the worst element in the population. This process is then repeated for some fixed number of iterations.

Incremental Construction Algorithms

We studied a pure heuristic construction algorithm as well as backtracking algorithms based on this heuristic. The pure heuristic algorithm is NEH (Nawaz, Enscore, & Ham 1983), which is widely regarded as the best performing heuristic for the PFSP (Taillard 1990). The NEH algorithm can be summarized as follows:

(1) Order the n jobs by decreasing sums of total job processing times on the machines.
(2) Take the first two jobs and schedule them so as to minimize the partial makespan as if there were only two jobs.
(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 partial makespan.

Despite its simplicity ($O(n^3m)$), NEH produces reasonably good solutions to Taillard's benchmark problems. However, the solutions produced by *pathrelink* are either competitive with or exceed the previously best known solutions. Yet the comparison is hardly fair: the run-time of NEH is several orders of magnitude less.

The NEH algorithm can be viewed as a greedy constructive search method, which can be extended in several ways. Backtracking mechanisms can be added to recover from poor local decisions made during step (3) of the pseudo-code. The k insertion points can be sorted in ascending order of partial makespan to provide a relative quality measure.

We implemented several systematic backtracking algorithms: Chronological Backtracking (CB), Limited Discrepancy Search (LDS), Depth-bounded Discrepancy Search (DDS), and Heuristic-Biased Stochastic Sampling (HBSS). Chronological backtracking serves as a baseline performer for the heuristic backtracking algorithms. For LDS (Harvey & Ginsberg 1995) and DDS (Walsh 1996), a discrepancy is defined as any point in the search where the advice of the heuristic is not followed. LDS iteratively increases the maximum number of discrepancies allowed on each path from the root of the search tree to any leaf. In contrast, DDS iteratively increases the depth in the search tree at which discrepancies are allowed. Both algorithms assume the availability of a good heuristic, such as NEH. DDS further assumes that discrepancies required to achieve near-optimal solutions should occur at relatively shallow depths in the search tree. As no agreed-upon convention exists, we consider *any* move other than that suggested by the heuristic as a single discrepancy. This is unlike LePape & Baptiste (1997) , where $k-1$ discrepancies are counted if the k^{th} available move is chosen.

HBSS (Bresina 1996) is an incremental construction algorithm in which multiple root-to-leaf paths are stochastically generated. Instead of randomly choosing a move, an acceptance probability is associated with each possible move. This acceptance probability is based on the rank of the move assigned by the heuristic.

A bias function is then applied to the ranks, and the resulting values are normalized. The choice of bias function "reflects the confidence one has in the heuristic's accuracy - the higher the confidence, the stronger the bias" (Bresina, 1997:271). We used a relatively strong quadratic bias function, due to the strength of the NEH heuristic: $bias(r) = r^{-2}$, where r is the rank of a move.

Iterative Sampling Algorithms

We also implemented several iterative sampling algorithms. In random sampling, a number of random permutations are generated and evaluated. Another iterative sampling algorithm can be obtained by modifying step (2) of the NEH algorithm. Instead of selecting the two largest jobs, we instead choose two jobs at random. Step (3) of the NEH is then followed, without backtracking, to produce a complete schedule. We denote this algorithm by *NEH-RS* (NEH with random-starts).

Finally, we consider iterative random sampling in which local search is applied to the randomly generated solutions; we denote this algorithm by *itsampls* (iterative random sampling with local search). Following Reeves and Yamada (1998), we use a shift local search operator coupled with a next-descent search strategy. Let π represent a permutation and π_i be the element in the i^{th} position of the permutation. The operation $\pi_i \mapsto \pi_j$ denotes that the i^{th} element in the original permutation is re-mapped to the j^{th} position. Given two randomly selected positions i and j, $i < j$, the shift operator SH(i,j) transforms π as follows:

$$SH(i,j) : \pi \rightarrow \pi \begin{cases} \pi_k \mapsto \pi_{k+1} & \text{if } i \leq k < j \\ \pi_j \mapsto \pi_i & \\ \pi_k \mapsto \pi_k & \text{otherwise} \end{cases}$$

The operator is applied to all pairs of jobs in a random order, with each improving or equal move accepted. Finally, we note that because of the strong stochastic component, HBSS can be classified as either an incremental construction or iterative sampling algorithm.

Relative Algorithm Performance: Empirical Results

Algorithm performance was measured on six problem classes consisting of three sizes of both job and machine-correlated problems: 50, 100 and 200 jobs, all executed on 20 machines. For each problem class, we generated 100 problem instances with $\mu_{lb} = 35$ and α ranging from 0.1 to 1.0, in increments of 0.1 The values of σ_{lb} and σ_{ub} were set to 1 and 20, respectively. Varying the problem size allows us to assess algorithm scalability, while varying α allows us to assess the influence of structure on algorithm performance.

The process of placing a job into the best possible location of a partial schedule dominates the run-time of all NEH-based algorithms; this corresponds to the loop body in step (3) of the NEH pseudo-code. We define this computation as an evaluation and allocate 100K such evaluations to all NEH-based algorithms. For HBSS and NEH-RS, root-to-leaf paths are repeatedly generated with the best obtained solution recorded. For the systematic heuristic backtracking algorithms (CB, LDS, and DDS), a target makespan must be specified. We begin by setting the target makespan to one less than the makespan obtained by the NEH algorithm. Once an algorithm locates a solution with a makespan equal to or better than the target makespan, a new target makespan is similarly defined, and the algorithm is restarted. For random sampling, 100K solutions are generated and evaluated, with the best retained.

For the *pathrelink* algorithm, either local search or path projection is performed at each iteration. Each local search or path projection involves 1000 steps, each requiring an evaluation. The total number of evaluations is limited to 100,000. For all *itsampls* trials, we allow two 'all-pairs' iterations and limit the total number of evaluations to 100K.

For each algorithm, we recorded the best makespan obtained on each problem. The optimal makespans for these problems are unknown; we measured individual algorithm performance by computing the *percent above the best solution found* by any of the search algorithms considered. Finally, we obtained a summary measure of algorithm performance at each level of α for each problem class by computing the average percent above best for the 100 problems.

Machine-Correlated Problems

Figures 1, 2, and 3 record algorithm performance on 50 by 20, 100 by 20, and 200 by 20 *machine-correlated* problems. All algorithms significantly outperformed random sampling. As a group, the stochastic algorithms (*itsampls*, NEH-RS, and HBSS) outperform the deterministic algorithms (NEH, LDS, and DDS). The superior performance of HBSS and NEH-RS is both sustained and magnified as problem size increases: both algorithms scale extremely well. The performance of *itsampls* fails to scale to 200 by 20 problems; for small values of α, it is outperformed by the deterministic algorithms. Interestingly, the two strongest performers, HBSS and NEH-RS, are based on a domain-specific heuristic (NEH), while *itsampls* is not.

Both LDS and DDS improve over the pure NEH algorithm and significantly outperform chronological backtracking. In comparison to LDS, the slight under-performance of DDS suggests that for machine-correlated problems it is important to consider discrepancies deep in the search tree. An analysis of LDS execution traces supports this observation and also indicates that it is often necessary to consider moves that are deemed extremely poor by the heuristic. Clearly the degradation of the NEH heuristic is not gradual.

The most striking aspect of Figures 1 - 3 is the inconsistent, and often poor, performance of the *pathrelink* algorithm. In Figure 1, *pathrelink* starts to underperform relative to both HBSS and NEH-RS between α equal to 0.1 and 0.2. At larger values of α, *pathrelink* is outperformed by many of the other, simpler algorithms.

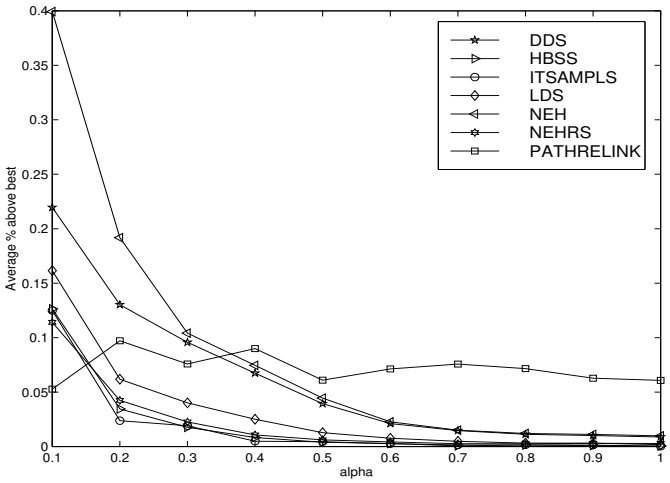

Figure 1: 50x20 machine-correlated

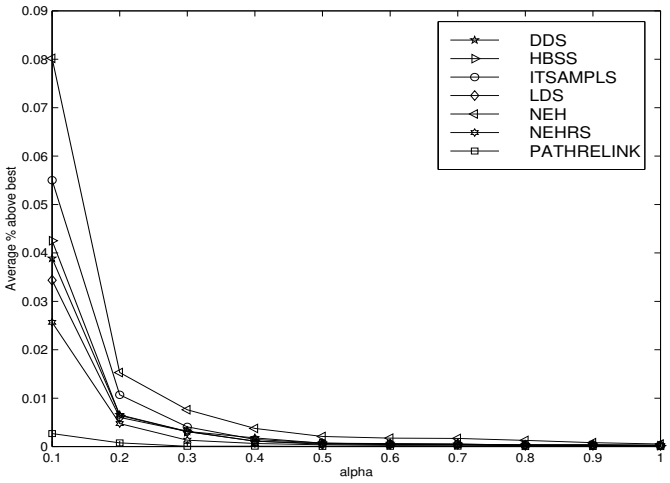

Figure 4: 50x20 job-correlated

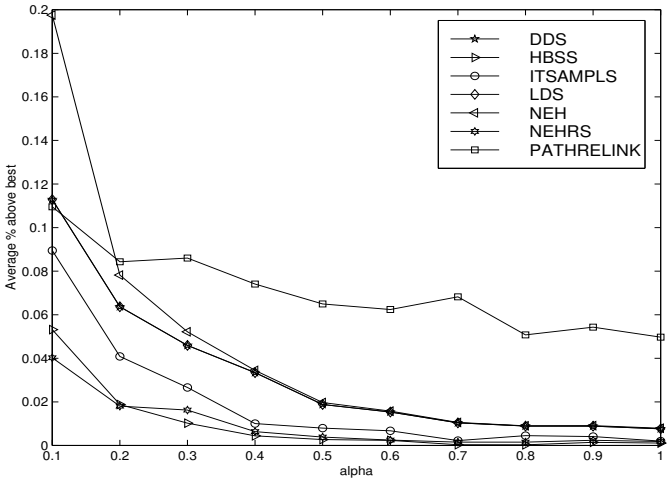

Figure 2: 100x20 machine-correlated

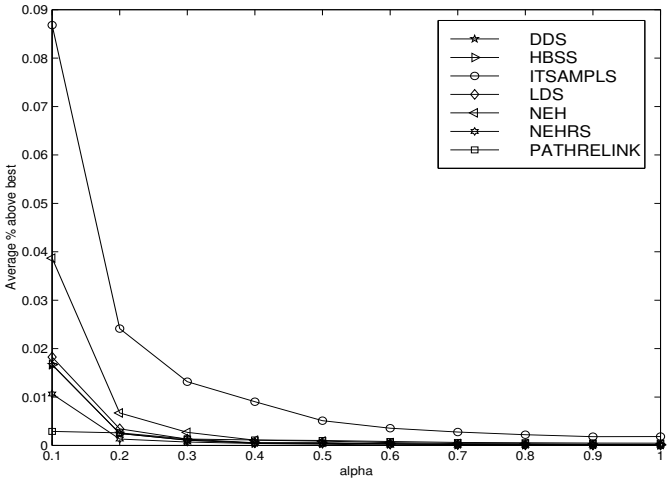

Figure 5: 100x20 job-correlated

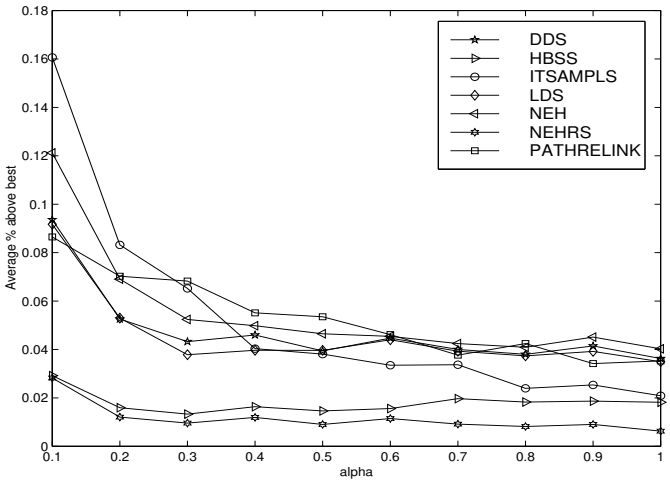

Figure 3: 200x20 machine-correlated

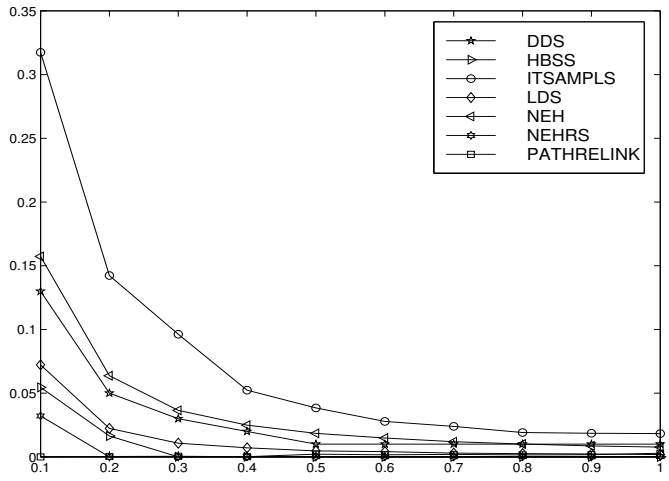

Figure 6: 200x20 job-correlated

Figures 2 and 3 show that for larger problem sizes, *pathrelink* underperforms both HBSS and NEH-RS on problems at all levels of α, including 0.1. The performance of the *pathrelink* algorithm fails in that 1) it does not scale to larger machine-correlated problems and 2) even minor amounts of structure cause it to lag the best iterative stochastic algorithms.

Job-Correlated Problems

Figures 4, 5, and 6 record algorithm performance on 50 by 20, 100 by 20, and 200 by 20 *job-correlated* problems. Again, all algorithms significantly outperformed random sampling. As was the case for machine-correlated problems, the stochastic algorithms outperform the deterministic algorithms, excepting the performance of *itsampls*, which fails to scale to larger problem sizes. Here, the performance degradation of *itsampls* is even more rapid than that exhibited on machine-correlated problems; on 100 by 20 and 200 by 20 problems, NEH obtains superior results at *all* values of α.

NEH-RS remains the strongest overall performer; it only slightly underperforms *pathrelink* when $\alpha = 0.1$. HBSS, LDS, and DDS all improve over NEH and chronological backtracking, but are basically indistinguishable for larger values of α. LDS continues to perform slightly better than DDS. Finally, the move from 100 by 20 to 200 by 20 problems results in greater differences in algorithm performance, although the relative order remains stable.

The performance of *pathrelink* remains the most unexpected result of these experiments. In contrast to the machine-correlated results, *pathrelink* consistently outperforms all other algorithms, on all problem sizes, with the sole exception of NEH-RS. Even more interesting is the fact the performance *pathrelink* appears to be *independent* of α, the level of non-random problem structure.

Assessing Problem Structure

Our study of non-random PFSPs is motivated not only by the fact that they actually model real-world problem attributes, but also as an exploration of what it means to be a "hard" problem and how this influences algorithm performance. Taillard's problems have been filtered to be "hard." In other areas of AI, the phase transition regions of problems such as SAT or Hamiltonian circuits have been explored as a source of difficult test instances (Cheeseman, Kanefsky, & Taylor 1991).

Reeves and Yamada (1998) show that Taillard's difficult flow-shop problems display a *big-valley* problem structure when the *shift* local search operator is used. The notion of *big-valley* is somewhat imprecise. It suggests that 1) local optima tend to be relatively close to other local optima, 2) better local optima tend to be closer to global optima, and 3) local optima near one another have similar evaluations. To find the big-valley phenomenon, however, one must pick the "correct" local search operator. First introduced in the context of the TSP and Graph Bipartitioning problems (Boese, Kahng, & Muddu 1994), some recent state-of-the-art algorithms are explicitly designed to exploit this structure.

We hypothesized that the poor performance of the state-of-the-art *pathrelink* on machine correlated problems algorithm might be attributable to a lack of the big-valley structure in our problems. Thus, we tested the different types of PFSPs for it. In our experimental setup, the underlying distribution (Gaussian or uniform) and the parameters defining the type of structure (correlation on jobs, machines, and α, or no correlation) are the independent variables. The dependent variables are the distance between local optima and the quality of the solutions obtained, quantifying the presence and extent of the big-valley structure.

For each problem, we generate 2000 local optima by starting with random permutations and running local search using the shift operator. The shift operator is repeatedly applied, in a next-descent strategy for all the possible pairs of jobs in the permutation, in a random order, until two passes through all the pairs does not result in any improvement. Because the global optima for our problems are unknown, we next compute for each local optimum its average distance to all the other local optima, as was done in the previous study (Reeves & Yamada 1998). We use an operator-independent precedence-based measure to compute pairwise distances. For two permutations π and π' of length n, the computation is:

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

where the function $preceeds(i,j,\pi)$ returns 1 if i occurs before j in permutation π. Finally, one deficiency of this methodology is that it fails to distinguish cases where multiple local optima are equivalent, i.e., they all reside on the same plateau and can be transformed into one another by non-degrading (in terms of makespan) applications of the shift operator.

Taillard's Problems

In the top graph of Figure 7, the results for a 50x20 problem (TA052) from Taillard's test suite serve as a prototypical example of a big-valley structure. In this figure, the local optima tend to be clustered, with good local optima close to each other. We used Taillard's problem generator included with the benchmark problems in (Beasley 1998) to produce new problems. Scatterplots for these problems were similar to those of the top graph in Figure 7. Thus, *all* problems produced by this generator appear to satisfy big-valley requirements, not just the ones selected as difficult.

To determine the impact of the choice of distribution on Taillard's problem generator, we replaced the uniform distribution on the interval [1,99] with the Gaussian distribution $\eta(50, 16)$. The bottom graph in Figure 7 shows a typical example of the resulting scatterplots.

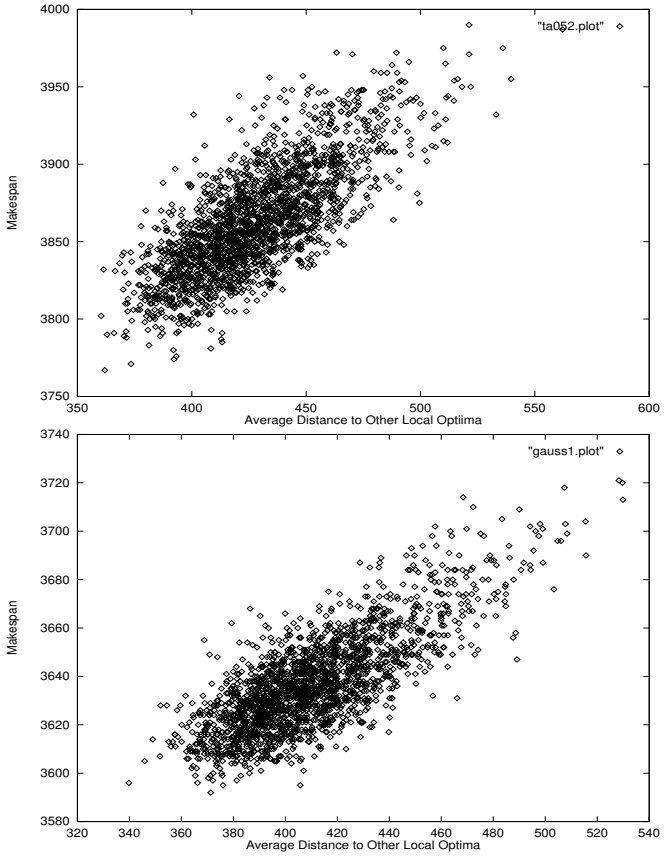

Figure 7: Taillard's TA052 50x20 Instance, uniform distribution, no correlation (top) and a Taillard-Gaussian 50x20 Instance (bottom).

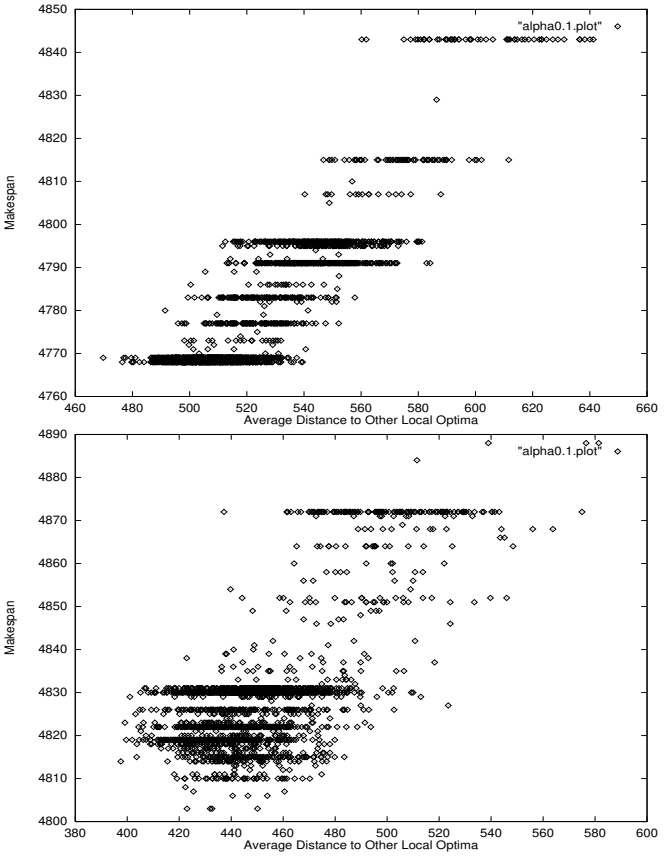

Figure 8: Machine-correlated 50x20 Instances, α=0.1. The top graph is a uniform distribution; the bottom graph is a Gaussian distribution.

The choice of distribution appears to have no significant impact on the existence of the big-valley structure.

Correlated Problems

We next investigated the effect of correlation on the landscape generated by the shift operator, when a Gaussian distribution is used. We generated local optima and distance measures for several 50x20 instances of both job and machine-correlated problems, varying α from 0.1 to 1.0 in increments of 0.1. The bottom graph in Figure 8 shows the result for a machine-correlated problem generated with α equal to 0.1. The results for job-correlated problems were similar. Note that an α of 0.1 represents a very low level of correlation. While there is still evidence of a big-valley structure, another dominant structural feature begins to emerge: strata of local optima at specific makespan values. Further analysis indicates that many members of the same stratum actually belong to the same plateau which can be partitioned into a small number of distinct local optima.

Although not shown, we also varied the amount of problem structure as measured by α. The empirical evidence suggests that the number of plateaus gradually drops to only a few, and all local optima are gradually absorbed into some plateau.

Finally, we checked for an interaction effect of the distribution and correlation. The question is whether the plateaus emerged due to the combined influence of correlation and Gaussian distribution. We therefore generated job and machine-correlated problems using a uniform distribution. The result from a 50x20 machine-correlated instance (α=0.1) is shown in the top graph of Figure 8; the results for job-correlated instances were similar. As with non-correlated problems, the choice of distribution appears to have little or no impact on the results. For job and machine-correlated problems, the big-valley structure is not the dominant structural feature of the fitness landscape. As the level of structure is increased, the landscape is dominated by only a few very large plateaus of local optima.

This suggests that we still do not have a clear picture of how problem structure impacts algorithm performance. *Pathrelink* performs well on random problems and job correlated problems, but not machine correlated problems. Yet, both job correlated problems and machine correlated problems are characterized by plateaus rather than the distribution of local optima normally associated with the big-valley phenomenon.

Implications and Conclusions

PFSP test suites are developed with little regard to structural features present in real-world problems. Thus, the apparent excellent performance of particular algorithms on these problems may not generalize to superior performance on real applications.

We constructed a PFSP generator to produce problems with variable amounts of two types of structure: job and machine correlation. We then compared the performance of several algorithms on problems of various sizes, with different amounts of structure.

For both job and machine-correlated problems, a simple iterative stochastic algorithm, NEH-RS, provides the best overall performance and scalability. In comparison, the *pathrelink* algorithm, which does exceptionally well on the random Taillard PFSP test suite, fails to sustain this performance on problems with a modest amount of machine-correlated problem structure. Clearly, algorithms that work best on 'hard' problems, such as those from Taillard's test suite, may not be the best on more realistic classes of problems.

While interesting, the performance of individual algorithms was not the primary goal of these experiments. Rather, these experiments show that each of the following have a potentially significant influence on individual algorithm performance: the type of problem structure, the amount of problem structure, and the problem size.

We also compared the structure of the search spaces for Taillard's problems and problems from our generator. Previously, it had been proposed that the big-valley search space structure was characteristic of difficult flow-shop problems and that the best algorithms were designed to exploit that structure. However, we found the big-valley structure in both Taillard problems selected as difficult and in randomly generated problems. Additionally, we did not find the big-valley structure in our correlated problems; instead, we found a plateau structure. Based on this analysis, we suggest that some state-of-the-art algorithms may have been optimized for a problem structure particular to the randomly generated, uncorrelated problems. Thus, to better define algorithms for structured problems, we need measures of structure and a better understanding of the interaction of problem structure and algorithm design.

Our results reinforce the following warning: the problems found in artificial test suites provide little indication regarding the performance of algorithms on real-world problems. Furthermore, high levels of complexity may not be required to solve real-world problems: in our work, simple heuristic-based iterative sampling algorithms provided the best overall performance.

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0271. 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 thank the anonymous reviewers for their comments on an earlier version of this paper.

References

Beasley, J. E. 1998. *OR-LIBRARY*. http://www.ms.ic.ac.uk/info.html.

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

Bresina, J. L. 1996. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. 1991. Where the really hard problems are. In *Proceedings of IJCAI-91*.

Garey, M. R., and Johnson, D. S. 1979. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freemand and Company.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*.

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

LePape, and Baptiste. 1997. An experimental comparison of constrait-based algorithms for the preemptive job-shop scheduling problem. In *CP97 Workshop on Industrial Constraint-Directed Scheduling*.

Nawaz, M.; Enscore, E.; and Ham, I. 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.

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

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

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

Walsh, T. 1996. Depth-bounded discrepancy search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*.

Whitley, L. D. 1989. The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*.

Wolpert, D. H., and Macready, W. G. 1995. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.