J Sched (2011) 14: 121–140
DOI 10.1007/s10951-009-0113-0

Understanding the behavior of Solution-Guided Search for job-shop scheduling

Ivan Heckman · J. Christopher Beck

Published online: 5 August 2009

Abstract This paper investigates reasons behind the behavior of constructive Solution-Guided Search (SGS) on job-shop scheduling optimization problems. In particular, two, not mutually exclusive, hypotheses are investigated: (1) Like randomized restart, SGS exploits heavy-tailed distributions of search cost; and (2) Like local search, SGS exploits search space structure such as the clustering of high-quality solutions. Theoretical and experimental evidence strongly support both hypotheses. Unexpectedly, the experiments into the second hypothesis indicate that the performance of randomized restart and standard chronological backtracking are also correlated with search space structure. This result leaves open the question of finding the mechanism by which such structure is exploited as well as suggesting a deeper connection between the performance of constructive and local search.

Keywords Search · Empirical analysis · Algorithm behavior · Constraint programming

I. Heckman
Department of Computer Science, University of Toronto, Toronto, Canada
e-mail: iheckman@gmail.com

J.C. Beck (✉)
Department of Mechanical & Industrial Engineering, University of Toronto, Toronto, Canada
e-mail: jcb@mie.utoronto.ca

1 Introduction

Solution-Guided Search (SGS)[1] is a complete, constructive search technique that has been shown to outperform standard constructive search techniques on a number of constraint optimization and constraint satisfaction problems including job-shop scheduling (Beck 2007, 2006, 2005). Recent work on a hybrid of SGS with tabu search achieves state-of-the-art performance on a well-known set of job-shop scheduling problem benchmarks and demonstrates that SGS is critical for this performance (Watson and Beck 2008).

The fundamental novelty of SGS is the use of sub-optimal solutions, found earlier in the search, to guide subsequent search. While such guidance is not unusual in local search and metaheuristic methods, it has not been deeply studied in a constructive (i.e., tree search or branch-and-bound) context.

In this paper, using job-shop scheduling as a case study, we empirically investigate two conjectures regarding the reasons for SGS performance (Beck 2007): the exploitation of heavy tails and the exploitation of search space structure as measured through descriptive models of algorithm behavior.

- *Heavy-tailed distributions in constructive search*: The studies of Gomes et al. (2005) found that constructive search can exhibit tremendous variability in the cost of finding a solution, with distributions in search cost that can be characterized as heavy-tailed. These empirical and

[1] In previous publications (Beck 2007, 2006, 2005), SGS has been called Multi-Point Constructive Search and Solution-Guided Multi-Point Constructive Search. In this paper, we adopt the shorter name and abbreviation.

theoretical studies led to a randomized rapid restart technique that can greatly improve performance of constructive search algorithms. We investigate if and when heavy-tailed distributions appear in job-shop scheduling optimization, and their relation to SGS performance in order to test the hypothesis that, like randomized restart, SGS is able to avoid the especially long runs observed in chronological backtracking search. Our results provide strong theoretical and empirical evidence that SGS exploits heavy-tailed run-time distributions.

- *Descriptive models of algorithm behavior*: A descriptive model of algorithm behavior is a tool used to understand an algorithm's performance on a particular class or instance of a problem. There has been considerable work over the past 15 years in developing models of problem hardness (Gent et al. 1996; Smith and Dyer 1996) as well as work that has focused more directly on modeling the behavior of specific algorithms or algorithm styles. Building on Watson et al. (2003), we evaluate static models of SGS search cost. These models examine the correlation between search cost and measures of different search space characteristics, such as backbone size and the distance between local optima and the nearest global optima. We show that, like tabu search, SGS improves when characteristics like the clustering of high-quality solutions exist in a problem instance. Interestingly, randomized restart and chronological backtracking *also* show high correlations with search space features. This latter result comes as a surprise and is the first evidence of its kind of which we are aware.

In the next section, we review SGS. Section 3 addresses SGS performance in the absence and presence of heavy-tailed distributions, while Sect. 4 looks at measuring the correlation of SGS performance with different search space features. In each of these sections, we place our work in context, present empirical studies, and discuss the implications of our results. Section 5 provides a summary and conclusions.

2 Solution-Guided Search

Solution-Guided Search (SGS) is a search algorithm which incorporates ideas and methods from local search within a constructive search framework. It consists of a sequence of resource-limited, randomized backtracking search iterations guided by a pool of the best solutions encountered so far during the search. Studies of SGS (Beck 2005, 2006, 2007) have found it to outperform standard backtracking and randomized restart on both job-shop scheduling optimization and the quasi-group with holes completion constraint satisfaction problem.[2]

Algorithm 1: SGS: Solution-Guided Search

```
SGS():
1  initialize elite solution set e
2  while termination criteria unmet do
3      if rand[0, 1) < p then
4          set fail limit, b
5          s := search(∅, b)
6          if s is better than worst(e) then
7              replace worst(e) with s
       else
8          r := randomly chosen element of e
9          set fail limit, b
10         s := search(r, b)
11         if s is better than r then
12             replace r with s
```

Pseudocode for the basic SGS algorithm is shown in Algorithm 1. The algorithm initializes a set, e, of guiding or "elite" solutions and then enters a while-loop. In each iteration, with probability p, search is started from an empty solution (line 5) or from a randomly selected elite solution (line 10). In the former case, if the best solution found during the search, s, is better than the worst elite solution, s replaces the worst elite solution. In the latter case, s replaces the starting elite solution, r, if s is better than r. Each individual search is limited by a fail bound: a maximum number of fails that can be incurred. The entire process ends when the problem is solved, proved insoluble within one of the iterations, or when some overall bound on the computational resources (e.g., CPU time, number of fails) is reached.

SGS has been defined in detail elsewhere (Beck 2007). Here we summarize a number of points to make the algorithm clear.

- The elite solutions can be initialized by any search technique. The intention is that some quick heuristic is used to find a diverse set of solutions which will likely be of poor quality.
- Each individual search is bounded by an evolving fail bound: a single search (lines 5 and 10) will terminate, returning the best solution encountered (if any), after it has failed (i.e., encountered a dead-end) the corresponding number of times. This bound may grow over time to make the search complete (see below).

[2] See Heckman and Beck (2007) for details of SGS in a constraint satisfaction context.

- With some probability, p, search is started from an empty solution (line 5). Searching from an empty solution simply means using any standard constructive search with a randomized heuristic and a bound on the number of fails.
- To exploit constraint propagation, an upper bound of the cost function is set at each iteration. Here, we simply set the upper bound on the cost to be one less than that of the best solution found so far. (See Beck (2007, 2005) for alternative approaches.)
- To start constructive search from an elite solution, a search tree is created using any randomized variable ordering heuristic and specifying that the value assigned to a variable is the one in the elite solution, provided it is still in the domain of the variable. Otherwise, any other value ordering heuristic can be used to choose a value. In summary, the guiding elite solution is used as a value ordering heuristic.

The variable ordering heuristic is randomized, while the value ordering heuristic is deterministic. This is sufficient for repeated searches to explore different areas of the search space which is why some randomization must be present. Note that "randomized" does not mean that variables are chosen with uniform probability. As described below, we use sophisticated scheduling-specific heuristics to choose a small set of possible decisions and randomly select from this set with uniform probability.

3 SGS and heavy-tailed distributions

It has been shown that constructive search algorithms can be improved through a randomized restart technique which exploits the heavy-tailed nature of a randomized algorithm's run-time distribution (Gomes et al. 1998). By demonstrating that SGS can take advantage of these heavy tails in the same way, we can incorporate this work into our understanding of SGS.

In this section, we review research on heavy-tailed distributions in backtracking search including how to find the distributions and how to theoretically and empirically boost performance with a rapid-restart strategy. We argue that SGS benefits the same way. Empirical investigations are then performed to confirm the theoretical argument: using job-shop scheduling optimization problems we show that SGS performance improves over basic backtracking at precisely the same problem sizes at which heavy-tailed distributions appear.

3.1 Background

Given a single problem instance and a backtracking algorithm with some degree of randomness, the cost of a single run can be highly variable. Sometimes a solution is found quickly, while with other random seeds, the time to find a solution is so high that the problem is practically unsolvable. Studies have found these distributions to have heavy tails (Gomes et al. 1998, 2000; Gomes and Selman 1999). More specifically, the probability that the cost of the next stochastic run (random variable X) is greater than some value x can be modeled by a Pareto–Levy distribution:

$$Pr\{X > x\} \sim Cx^{-\alpha}, \quad x > 0,$$

where the parameter C is a positive scaling factor and α, the index of stability, determines which of the moments are infinite.[3] When $\alpha < 1$, all moments are infinite, when $1 \leq \alpha < 2$ the mean is finite, but the variance and all higher moments are infinite, and so on.

Heavy tails can be checked visually by plotting $1 - F(x)$, where $F(x)$ is the proportion of runs complete after a search cost of x. For heavy tails, the decay on this chart should appear close to linear on a log–log scale plot, with the slope giving an estimation of α. For non-heavy tails, the decay will appear more than linear, or fit a very steep line where none of the lower moments are infinite.

As shown theoretically and empirically by Gomes et al. (1998), if a randomized backtracking algorithm is repeatedly restarted after some resource limit, the expected run-time distribution will no longer be heavy-tailed. Equation (1) expresses the expected tail of the run-time distribution of a randomized restart strategy, the probability that the random variable representing the cost of this strategy, S, will be greater than some cost, s. A is the random variable representing the original randomized backtracking search, and c is the *fixed* restart cut-off used. The form of (1) follows an exponential distribution: heavy tails are eliminated as exponential distributions have finite mean and variance.

$$P[S > s] = (1 - p)^{\lfloor s/c \rfloor} \; P[A > s \bmod c]. \tag{1}$$

Instead of a fixed limit, an increasing sequence of limits at each successive restart results in a complete search procedure as the limit will eventually be large enough to exhaust the whole tree. Luby et al. (1993) have formulated a universal sequence of limits that is optimal given no prior information of the original distribution, and only a log factor away from the optimal fixed cut-off computed from complete information of the search cost distribution. See Wu and van Beek (2007) for recent work on fail sequences.

[3] Of course, as constructive search is complete and the size of the full search space is finite, none of the moments can actually be infinite. Yet, some of the runs will take so long as to be practically infinite, in that we will never wait until they are finished. The distributions are actually "truncated heavy tails" (Gomes et al. 2000). The infinite model works well for the range of search costs investigated here.

Much research has been done on when and how these heavy-tailed distributions come about. Williams et al. (2003) have shown how they are related to back-door variables and a related paper presents a theoretical model of backtracking search which predicts heavy-tailed behavior when search is unbalanced (Chen et al. 2001). Gomes et al. (2004) correlate heavy tails with thrashing behavior in inconsistent sub-trees.

3.2 SGS as a randomized restart algorithm

When the probability of starting a non-guided iteration in SGS is 1, the algorithm is identical to randomized restart. If the underlying randomized backtracking search has heavy tails, each run will sample from it and heavy tails will be eliminated. The question then becomes: when SGS is guided by elite solutions, will heavy tails still be eliminated? The main difference is that SGS will be sampling from a distribution of guided runs. If we let G be a random variable representing this distribution, the number of backtracks needed for the next guided run, we get a similar formula for the probability mass of the tail as (1), with $p = P[G \leq c]$, and a final term of $P[G > c \mod s]$. As long as $P[G \leq c] > 0$, the distribution will still have an exponential form, and SGS will eliminate heavy tails. Hence, this guided distribution, G, does not even have to be heavy-tailed for SGS to perform better than chronological search. It suffices that the original backtracking distribution, A, is heavy-tailed for SGS to perform better than chronological search. Yet, if G is heavy-tailed, then a rapid restart technique is just as necessary in SGS as it is for randomized restart.

In the balance of this section, we empirically confirm this argument. In Experiment 1, we find job-shop scheduling problem instances in which the search cost distribution of chronological backtracking search is heavy-tailed. We then examine how search performance of SGS and a randomized restart algorithm changes as the instances start to exhibit heavy-tailed distributions. Finally in Experiment 3, we examine the distribution of *guided* runs. In addition to supporting our analytical argument, these experiments are the first of which we are aware to demonstrate heavy-tailed distributions for job-shop scheduling problems.

3.3 Experiment 1: Heavy tails in job-shop scheduling

In our first experiment, we attempt to find heavy-tailed behavior in backtracking search on instances of the job-shop scheduling problem.

3.3.1 The job-shop scheduling problem

An $n \times m$ job-shop scheduling problem (JSP) contains n jobs each composed of m completely ordered activities. Each activity, a_i, has a predefined duration, d_i, and a resource, r_i, that it must have unique use of during its duration. There are m resources and each activity in a job requires a different resource. A solution to the JSP is a sequence of activities on each resource such that the *makespan*, the time between the maximum end time of all activities and the minimum start time of all activities, is minimized.

All job-shop problems experimented with here are square, in that the number of jobs is equal to the number of resources $n = m$. Ten problems of each order (13×13, 14×14, 15×15, 16×16) are generated using an existing generator (Watson et al. 2002). The routings of the jobs through the machines are uniformly generated and the activity durations are independently and uniformly drawn from [1, 99].

3.3.2 Experimental details

To determine the distribution of search cost in our job-shop scheduling problems each problem instance is run 1000 times using a randomized chronological backtracking search with different random seeds. Due to the size of the problems of interest, we only measure the search until a near optimal (4% from optimal) solution is found. Given an optimal makespan, c^*, we only search for a makespan, $c = 1.04 \times c^*$. This goal was chosen to allow most runs to finish within the 10,000,000 choice point limit. A similar technique has been used in investigating heavy tails in optimization problems (Gomes and Selman 1999). We previously solved each problem instance to optimality so that c^* is known.

Ideally, no global limit would be put on each overall search cost. But considering how many runs were needed, and that given the size of the problems certain runs could take days, a high limit of 10,000,000 choice points is used. Although this affects the strength of the statistics (some extreme values expected by heavy-tailed distributions would be missed), a modified maximum likelihood estimate of the index of stability has already been used for such a case (Gomes et al. 1998).

A random solution is generated at the start of each run in order to set the initial upper-bound, and for guiding the solutions in Experiment 3. These are generated through a constraint-based *schedule-or-postpone* (Scheduler 2006) technique that attempts to assign start times to all activities. At each step it chooses randomly from all activities which can be scheduled next and assigns the activity its earliest start time. An activity can be scheduled next if the preceding activity in its job has been assigned (or if it is the first activity in a job), and if the activity has not been postponed. An activity becomes postponed if assigning its earliest start time leads to a failure. An activity stays postponed until this earliest start time is updated via constraint propagation.

For all subsequent algorithms, texture-based heuristics are used (Beck and Fox 2000). These heuristics use *texture measurements*, algorithms that distill relevant structure from the current search state, to dynamically focus the attention

Table 1 Mean and standard deviation of the number of choice points to find a 4% from optimal solution to JSP instances of varying size along with mean α values of their tails measured in Experiment 1. Also included is the number of problem instances for which the estimated α value is less than one. **Bold** entries indicate lowest mean number of choice points for each problem size

Size	$\overline{\alpha}$	$\alpha < 1$ count	Chron		Restart		SGS	
			mean	sd	mean	sd	mean	sd
13 × 13	3.46	0	**1532**	447	2694	1889	11081	2120
14 × 14	2.02	1	**3939**	3327	4365	1479	18364	25374
15 × 15	1.35	2	21292	35336	**8050**	7349	20934	8185
16 × 16	1.09	6	92694	127361	**17062**	11551	19585	3553

on the variables in the problem that appear most critical. In the case of scheduling problems, the texture measurements assess contention: the amount of competition among activities for each resource at each time point. The textures identify a resource and time point with maximum contention. A pair of unordered activities competing for this resource and time point are then heuristically chosen and a branching decision is made on the two possible orderings. The heuristic is randomized by specifying that the resource and time point is chosen with uniform probability from the top 10% most critical resources and time points (see Beck and Fox (2000) and Beck (1999) for extensive details). The standard constraint propagation techniques for scheduling (timetable constraint, precedence graph, and edge finding) (Nuijten 1994; Laborie 2003; Le Pape 1994) are also used in all experiments and algorithms.

All algorithms were implemented in ILOG Scheduler 6.2 and run on a 2 GHz Dual Core AMD Opteron 270 with 2 GB RAM running Red Hat Enterprise Linux 4.

3.3.3 Results

Displayed in Fig. 1 are log–log survival functions for a pair of instances of each order. The y-value corresponds (log scale) to the proportion of the 1000 runs that still could not find a good solution (within 4% of optimal) after creating x (log-scale) choice points. The overlaid line is the fitted Pareto–Levy distribution of the tail using a modified maximum likelihood estimate of α formulated in Gomes et al. (1998). Estimates of the mean α values for each problem size are shown in Table 1. For heavy tails, we would expect the log–log plot of the tail to be linear, and have an α value that is associated with all infinite moments $\alpha < 1$. From the graphs and table, we see that heavy-tailed behavior emerges by size 16 × 16.

3.4 Experiment 2: Comparing algorithms

Having identified sizes of the JSP in which heavy-tailed distributions can be observed, we now compare search performance for three algorithms.

3.4.1 Experimental details

The same ten job-shop scheduling instances for each size between 13 × 13 and 16 × 16 from the first experiment are used again here. Three algorithms are tested:

- Standard chronological backtracking (Chron): An algorithm similar to that used in Experiment 1 except the heuristic is not randomized, and the texture heuristic is used for variable and value ordering. Since it is not randomized, one run is performed for each instance.
- Randomized restart (Restart): A randomized restart algorithm using the randomized texture heuristic from Experiment 1. Initially, the fail limit on a single search is 32 and whenever a new best solution is found, the fail bound is reset to 32. Whenever a search fails to find a new best solution, the fail bound is incremented by 32. This is referred to as the *polynomial* fail sequence. The polynomial sequence is used here instead of the Luby sequence because previous experiments (Heckman 2007, Chap. 3) indicated a lower run-time (though larger search tree) for the polynomial sequence. Each instance is run 10 times. As in SGS, on each successive restart, the best solution found so far is used as the new upper bound on the makespan.
- Solution Guided Search (SGS): To simplify the algorithm, and from good performance found in Beck (2007), the following parameters are used: elite set size: $|e| = 1$; probability of search from an empty solution: $p = 0.25$; initialization limit: 100; fail sequence: polynomial with a step of 32, as with Restart. The same randomized texture heuristic as in randomized restart is used, except, when guiding from a solution, the activity ordering in the guiding solution is always asserted. Each instance is solved 10 times.

A time limit of 1500 seconds is used with each run, giving enough time for all algorithms to find a 4% from optimal solution. All other experimental details (hardware, software, propagators) are identical to Experiment 1.

3.4.2 Results

Displayed in Fig. 2 are, for each problem size, the mean relative errors (MRE) relative to the known optimal solution

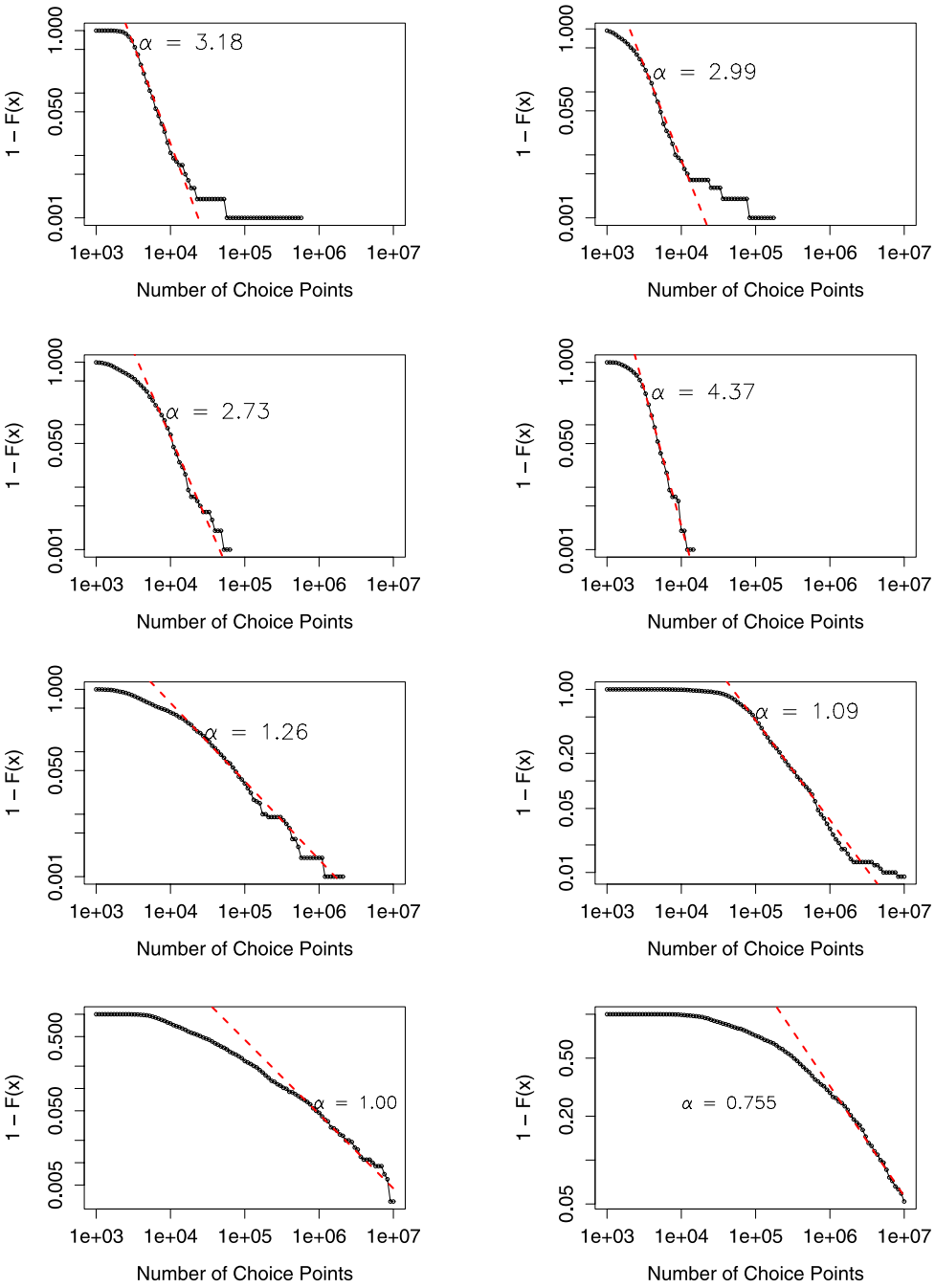

Fig. 1 Log–log plot of the survival function of backtracking search to find 4% of optimal for two random JSP instances for sizes, from top to bottom, 13 × 13, 14 × 14, 15 × 15, and 16 × 16. Also plotted, is the fitted Pareto distribution of the tail with the estimated parameter α

at 10 second intervals. The MRE is the mean of the relative error of each of the 10 problems at a given size n:

$$MRE(a, K_n, R, t) = \frac{1}{|R||K_n|} \sum_{r \in R} \sum_{k \in K_n} \frac{c(a,k,r,t) - c^*(k)}{c^*(k)}, \quad (2)$$

where K_n is the set of problem instances of size n, R is the set of independent runs with different random seeds on these instances, $c(a,k,r,t)$ is the makespan found by algorithm a on problem k, on run r, after t seconds.

While standard chronological search performs much better on the smaller problems, by size 16 × 16, it becomes the worst in terms of average performance. Table 1 displays the average number of choice points to find a 4% of optimal solution, along with data from Experiment 1: the mean estimated α values and a count of the number of problem instances (out of 10) for which the estimated α was less than one. Chronological search can still perform best on some larger instances, but it is also highly variable, which we would expect from heavy tails. Also from the table we can see, in contrast to the graphs in Fig. 2,

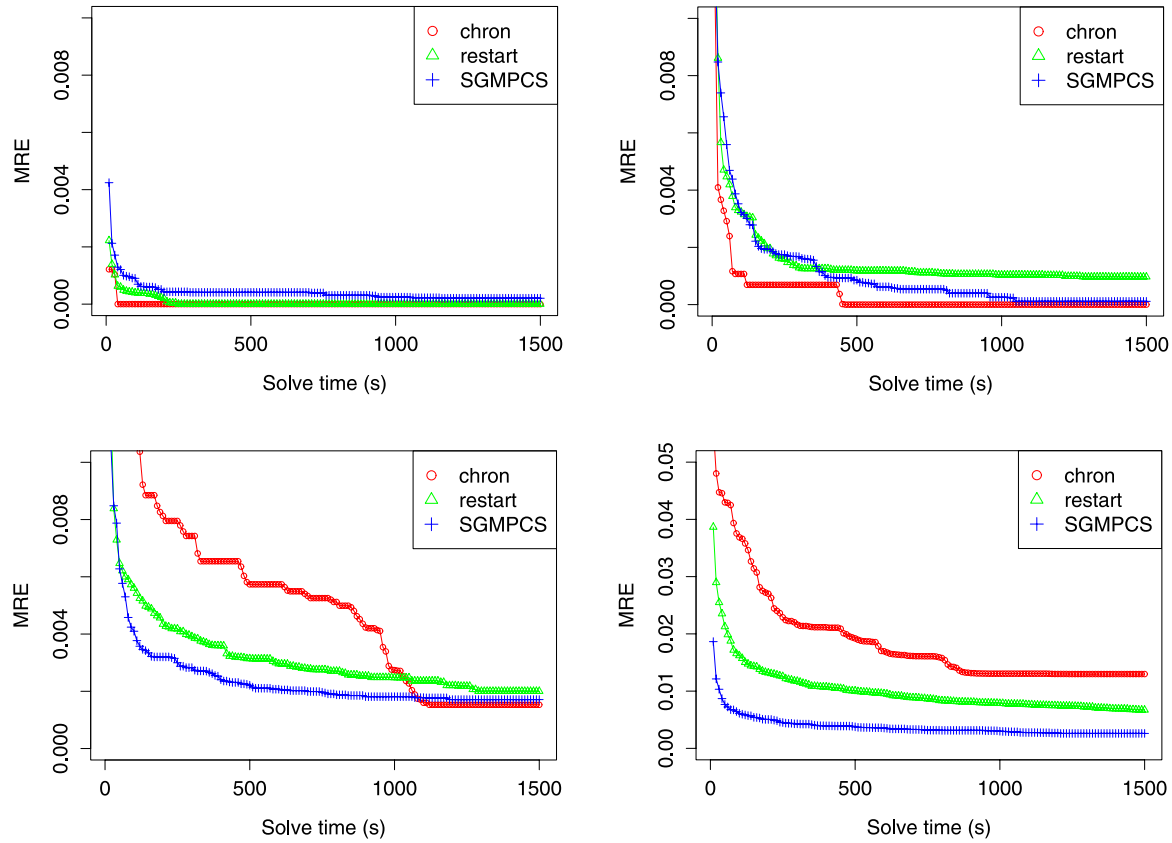

Fig. 2 Mean relative error of best solutions found over time for JSP problems of sizes 13 × 13 (*top left*), 14 × 14 (*top right*), 15 × 15 (*bottom left*), and 16 × 16 (*bottom right*). The y-axis of size 16 × 16 is different from the others since the chronological plot would not appear otherwise

that simple restart always performs better than SGS. This may seem contradictory, but what was plotted in Fig. 2 was the mean relative error over time. In contrast, Table 1 measures the mean cost of each run to achieve 4% relative error.

The graph in Fig. 3 shows how average performance relative to chronological search changes as the distributions become heavy-tailed, measured through the average estimate of α. Plotted on the x-axis are the means of the 10 α-values for each size. On the y-axis is average performance relative to chronological search for each size n:

$$\overline{R}_n = \frac{(\overline{A}_n - \overline{C}_n)}{\overline{C}_n}, \tag{3}$$

where $\overline{A}_n$ is the mean number of choice points to find a 4% of optimal solution on problems of size n for either SGS or Restart, and $\overline{C}_n$ is the mean performance for standard chronological search on the same problems. It can be observed that as the α value approaches 1 from above, the performance scores for SGS and Restart both become negative, indicating improved mean performance relative to chronological search.

3.5 Experiment 3: Finding run-time distributions of guided search

We have seen that heavy-tailed behavior can occur in JSP instances and that randomized restart and SGS can both benefit from these search cost distributions. The purpose of our final experiment in this section is to look at how search guidance may change this underlying distribution. Randomized restart samples from randomized heuristically guided runs whose distribution has been shown to be heavy-tailed in a number of circumstances. SGS samples from randomized *solution-guided* runs. The question investigated here is whether the guided distribution is still heavy-tailed. Even if the guided distribution is heavy-tailed, the extremely long guided runs will still always be avoided because of the restarts SGS uses, and the overall search cost distribution of SGS will not be heavy-tailed. If the guided distribution is not heavy-tailed, then perhaps there is a way to guide from solutions without the need of restarts. For example, we could imagine a non-restarting approach that changes its guiding solution whenever a better solution is found but continues with chronological backtracking. The purpose of this experiment is to see if a rapid restart technique is necessary in order for SGS to avoid heavy tails.

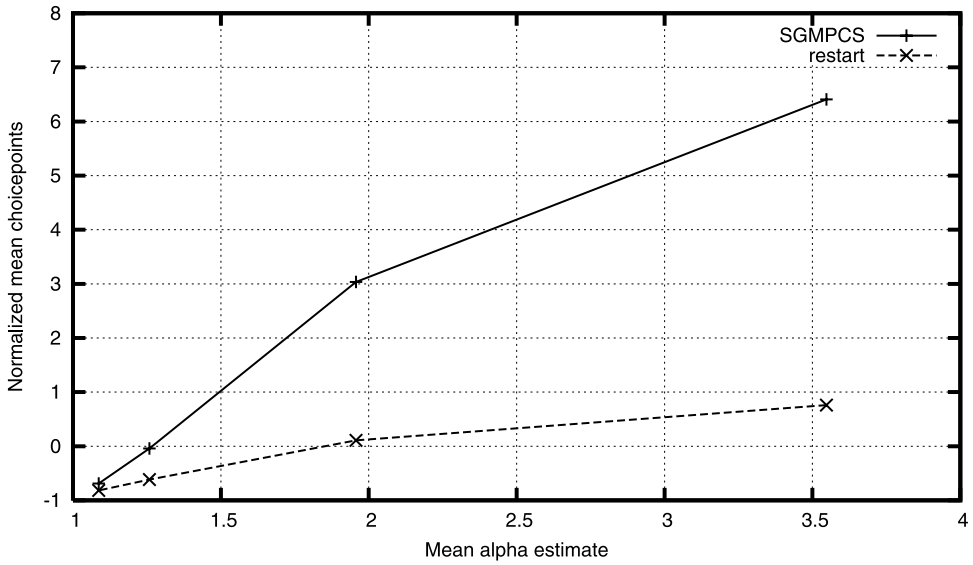

Fig. 3 Mean relative performance from randomized restart and SGS for problem sizes of changing average α estimates

3.5.1 Experimental details

In Experiment 1, we found the distribution of all possible runs that a randomized restart search may encounter. This was done by taking an instance and running it multiple times with a very high limit on the search and a different random seed each time. In the case of SGS, the population of all possible individual runs is characterized by the various random seeds as well as all possible sub-optimal guiding solutions. Therefore, to determine the distribution of search cost for guided runs each problem instance is run 1000 times with both a new random seed and a new randomly generated guiding solution. The random solution is generated in the same way as Experiment 1, but is now used for both setting an upper bound and guiding the run. The same randomized texture heuristic is used, but instead of randomly choosing an ordering for the pair of activities, the ordering in the guiding solution is asserted. Only the ten instances of size 16×16 are run for these experiments. All other experimental details are identical to Experiment 1.

3.5.2 Results

The log–log plots of the survival function over all instances are shown in Fig. 4, where the y-axis represents the proportion of runs over all instances of size 16×16 that have not found a 4% from optimal solution after x choice points have been created. From the plots, we can see the search cost distributions for guided and non-guided runs both exhibit heavy tails. In the guided case, the heavy tails occur sooner and are less steep. It is unclear at this point what significance, if any, this difference has. The maximum likelihood estimates of the α parameter are 0.728 and 0.547 for the randomized and guided distributions respectively. Plots focusing on the left-hand tail show little evidence of heavy tails as seen in Fig. 5, with α estimates for the left-hand Pareto distribution of 5.09 and 4.17 for the random and guided distributions, respectively.

Table 2 Estimates of α for guided and non-guided runs on ten 16×16 JSP instances

Instance	Non-guided α	Guided α
all	0.728	0.582
0	1.00	0.586
1	0.755	0.61
2	0.579	0.676
3	1.62	0.532
4	1.78	1.14
5	0.718	0.483
6	1.43	0.714
7	3.42	0.658
8	0.194	0.239
9	1.09	0.582

Heavy tails over multiple instances is of little use to randomized restart or SGS trying to solve a single instance. As shown in Table 2 and Fig. 6, the results when broken down per instance are more varied, but still tend to confirm heavy tails for both cases. The column labeled 'all' represents fitting a distribution of all runs over all problems, as shown in Fig. 4.

3.6 Discussion

We have shown that heavy-tailed behavior can occur in job-shop scheduling optimization. As expected, the average performance of SGS and randomized restart search both start to

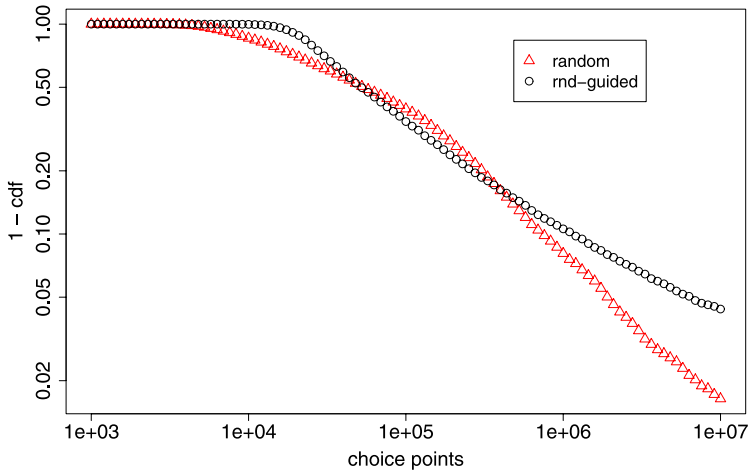

Fig. 4 Right-hand heavy tail plots of guided and non-guided (random) backtracking search on ten 16 × 16 JSP instances

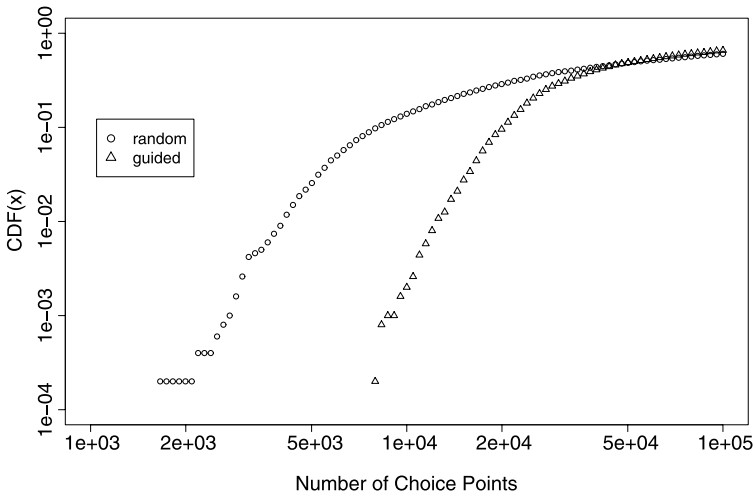

Fig. 5 Left-hand plots of guided and random backtracking search on ten 16 × 16 JSP instances

improve over standard chronological search on those problems where heavy-tailed behavior appears. We found the distribution of guided runs used by SGS is also heavy-tailed, suggesting that a rapid restart strategy is just as necessary as a component of SGS as it is for the standard randomized restart algorithm.

Our study may have been weakened by two details added due to time constraints: only searching to a suboptimal solution and censoring measurements at 10,000,000 choice points. This limit censored less than 5% of all 10,000 guided runs and less than 2% of all non-guided runs on the size 16 × 16 instances. Regarding searching to a suboptimal solution, a previous paper by Gomes and Selman (1999) employs a similar technique in studying heavy tails in optimization.

Restart techniques in optimization add extra benefits as well as complications to modeling their performance. At each restart, more is known about the bounds on the optimization function as the upper bound is reduced to the best found in previous iterations. Starting at the top of the search tree, this improved upper bound can by itself help find a better solution sooner through increased propagation. In satisfaction problems, however, each restart is the same as the last.[4] In contrast, the distribution of run-times that a restart technique samples from is likely to change as the bounds become more constrained.

Our definition of the distribution of guided runs may also be a bit simplistic. A randomized run is defined over every possible guiding solution. Yet, in SGS, a single elite solution may remain in the elite set for a large portion of the overall search. As well, not just any solution can be guiding solution, only better quality solutions are allowed into the elite set. Regarding the comparison of distributions for guided and non-guided runs, we may ask how different these heuristics really are. Either values are randomly chosen once at the start of search when the guiding solution is generated, or dynamically chosen at each choice point. This may alone explain the differing results. When

[4]This does not account for techniques which combine restarts with nogood learning (Baptista and Silva 2000).

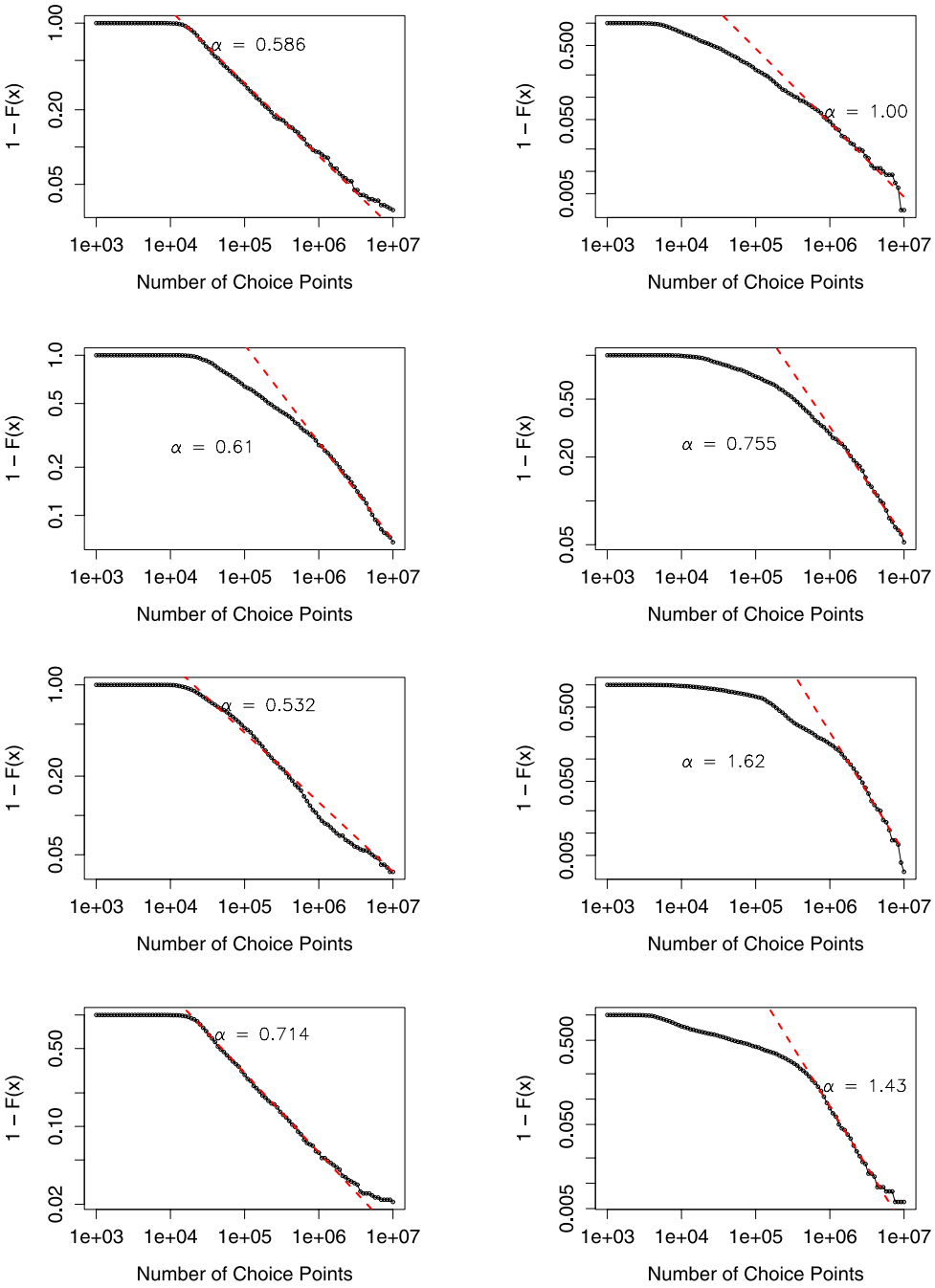

Fig. 6 Log–log plot of the survival functions of backtracking search to find a 4% of optimal solution, *left* guided by a random solution and *right* using a random variable ordering for randomly generated order 16 JSP instances 0, 1, 3 and 6. Also plotted is the fitted Pareto distribution of the tail with the estimated parameter α

guided by a randomly generated solution, the same possibly bad decision will be made whenever search comes to a particular variable whereas in the random heuristic it has another chance each time. These heavier tails for guided runs may be explained by the formal model of heavy tails of Chen et al. (2001): the static random value ordering when guiding by a solution leads the search tree to be more unbalanced, and hence more likely to exhibit heavy tails.

While the heavy tailed distribution explains why SGS and randomized restart perform better than basic backtracking,we have not yet explained why SGS can perform better than randomized restart. We turn to this question in the next section.

4 Problem difficulty for SGS in job-shop scheduling

In this section, we investigate how SGS performance may be related to static search space features. This is accomplished through a set of experiments analogous to those of Watson et al. (2003). Where Watson et al. examined the correlation

between search space features and tabu-search performance, we look at the correlation of the same features and the performance of SGS.

By design, SGS is guided by sub-optimal solutions. Differences in the performance of SGS and randomized restart (with all other parameters held constant) show that such a guidance makes a difference in algorithm performance (Beck 2007). Our conjecture is that, like local search techniques, SGS performance depends, at least partially, on the existence of search space features such as the clustering of good solutions. For example, one could imagine a pathological problem instance where high-quality sub-optimal solutions are maximally distant from the one optimal solution (Beck and Watson 2003). With such a topology, guiding search with sub-optimal solutions will not result in strong performance. Therefore, our hypothesis is that SGS search cost is correlated with the presence of search space features that have been shown to correlate with the search cost for local search algorithms (Watson et al. 2003).

In this section, we first review existing work on descriptive models of algorithm behavior in local search and describe the experiments and results of Watson et al. in detail. Then, we experiment to show how the various search space features measured by Watson et al. correlate with SGS performance on the same set of problem instances.

4.1 Background

An open problem in search algorithm research is to explain the large variation in search costs observed between problem instances (Mitchell et al. 1992; Gent et al. 1996; Parkes 1997; Singer et al. 2000). The focus has been predominantly in local-search algorithms where there is still a lack of understanding of why local search procedures work as well as they sometimes do.

Watson et al. (2003) took four features used in past studies (Clark et al. 1996; Parkes 1997; Mattfeld et al. 1999; Singer et al. 2000) and investigated their correlation to the search cost of tabu search on the job-shop scheduling optimization problem. The four features were the number of optimal solutions, backbone size, distance between local optima, and distance between local optima and the nearest optimal solution. Watson et al. refer to the use of these various search space features to account for the variability of search cost between instances as *static cost models* of problem difficulty. By static they are referring to the fact that the features are largely independent of the algorithm dynamics, relying on unchanging features of the search space.[5] Following Watson et al., we investigate the following static cost models:

- Number of optimal solutions ($|optsols|$): One of the earliest static cost models investigated was the number of optimal solutions (Clark et al. 1996). Intuitively, the more often solutions occur in the search space, the quicker it should be for a local search procedure to encounter one. Yet, on satisfiable problems, local search difficulty decreases past the phase transition peak even as the number of solutions continues to drop (Singer et al. 2000; Yokoo 1997).
- Backbone size ($|backbone|$): Used initially by Parkes (1997) to explain the search cost peak of local search on boolean satisfiability (SAT) problems, a backbone is the set of variables which always have the same value in any satisfying solution. As the backbone size becomes larger, solutions will necessarily cluster in the search space and hence be more difficult for local search procedures to locate. Parkes observed that many large backbone instances start to appear at the critically constrained region. Parkes also observed that when backbone size is fixed, difficulty always decreases as constrainedness increases.
- Distance between local optima ($\overline{loptdist}$): It has been observed that many local search algorithms consist mainly of movement from one local optima to another (Mattfeld et al. 1999; Watson et al. 2003). Seen as such, local search performance should be related to the distance between local optima: intuitively, the more widely spread the local optima the more effort is needed to move amongst them. Mattfeld et al. (1999) first used this measure to explain the differences in difficulty between general JSPs, where the ordering of operations are chosen at random, and workflow JSPs, where a structure is imposed on the orders.
- Distance between quasi-solutions and the nearest optimal solution ($\bar{d}_{\text{lopt-opt}}$): Singer et al. (2000) observed in local search for SAT, that search consisted mainly of movement from near-solutions, found early in the search, to a satisfying solution. They measured the average distance between these *quasi-solutions* (assignments where only 5 of the 100 clauses were unsatisfied) and the nearest satisfying solution, and found that it correlated well with local-search performance.
- Fitness–distance correlation (FDC): While not included by Watson et al., the correlation between the evaluated cost or fitness of a solution and its true distance to an optimal solution has been used in past studies (Jones and Forrest 1995) to predict problem difficulty.

4.1.1 *Problem difficulty for tabu search in job-shop scheduling*

As this work follows directly from that of Watson et al., we recount the details and results of their relevant experiments. The purpose of these experiments was to investigate various search cost models of problem difficulty for tabu search on a set of 'typical' instances of the general JSP.

[5] Dynamic cost models which do rely on dynamic features of specific algorithms—such as the set of solutions encountered by an algorithm—are investigated for tabu search and job-shop scheduling in Watson (2003).

Problem instances Watson et al. used 6×4 and 6×6 JSP instances: 6 jobs and 4 or 6 machines. Such small problem sizes were used because some of the static models require all optimal solutions to be enumerated, and the number of solutions can quickly grow into the billions at larger sizes. The 1000 instances were generated for each size with an earlier generator (Watson et al. 2002), in which the routings through the machines were randomly generated and the operation durations were independently and randomly drawn from [1, 99].

Search cost for tabu search on an instance, $cost_{\text{med}}$, was defined as median number of iterations needed to find an optimal solution over 5000 independent runs.

Static cost models The static models of search cost investigated by Watson et al. attempted to predict search cost from a search space feature of an instance. The accuracy of these models was quantified by the r^2 value of the linear regression model between the search space feature of an instance, and the log of the median search cost to find an optimal solution $log(cost_{\text{med}})$. We now recount how each feature was measured, and the resulting accuracy of their models on the 6×6 JSPs.

- *Number of optimal solutions* $|optsols|$: Watson et al. enumerated all solutions to each instance by using a constructive search algorithm (Beck and Fox 2000). They found a weak correlation ($r^2 = 0.2223$) between the log of the number of solutions and $log(cost_{\text{med}})$ for 6×6 JSPs.
- *Backbone size* $|backbone|$: The backbone of an instance is the set of solution components that have the same value in all solutions. This requires all solutions to be enumerated, as well as a definition of a solution component. Backbones of JSP instances were measured using a solution's *disjunctive graph* representation (Blazewicz et al. 1996). A solution to a 6×6 JSP instance in this representation consists of sets of $\binom{6}{2}$ Boolean variables for each machine. The variables represent the precedence relations for all pairs of operations that run on the same machine. The backbone size of an instance is the proportion of variables that have the same value in all optimal solutions: the proportion of paired orderings which are the same in each optimal solution. Watson et al. also found a weak correlation between log of median search cost and the square of backbone size ($r^2 = 0.2231$). This was a very similar correlation as found for $|optsols|$, and was explained through the high negative correlation between these two features ($r = -0.9103$).
- *Average distance between local optima* $\overline{loptdist}$: The final three search space features require a definition of distance between solutions and a random sampling of local optima. Watson et al. defined the distance between two solutions, s_1 and s_2, as the value of (4) (Mattfeld et al. 1999), where $precedes_{ijk}(s)$ is the predicate of whether job i is processed before job j on machine k in solution s, and $\oplus$ is the Boolean XOR operator. The actual distance used for the search space features was normalized: $\bar{D}(s_1, s_2) = 2D(s_1, s_2)/mn(n-1)$,

$$D(s_1, s_2) = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} precedes_{ijk}(s_1) \oplus precedes_{ijk}(s_2). \quad (4)$$

  Five thousand random local optima were generated for each instance using a hill climbing procedure with a random starting point and the N1 JSP move operator (Laarhoven et al. 1992). Watson et al. found a relatively low r^2 value (0.2744) for the static cost model using the average distance between the 5000 randomly generated local optima.
- *Distance between local optima and nearest optimal solution* $\bar{d}_{\text{lopt-opt}}$: Watson et al. adapted Singer et al.'s quasi-solution measure for JSP optimization by measuring the average distance between random local optima and the nearest optimal solution $\bar{d}_{\text{lopt-opt}}$. This measure used the 5000 local optima, and all optimal solutions generated for each instance as described earlier. The correlation found between the log of median tabu search cost and the square root of $\bar{d}_{\text{lopt-opt}}$ was the best of all static models investigated ($r^2 = 0.6541$).

4.2 Experiment 4: Evaluation of static cost models

In the following experiments, we measure the correlation of SGS search cost against the search space featured measured by Watson et al.[6]

4.2.1 Problem instances and search space features

The same 1000 6×6 JSP instances from Watson et al. are used in these experiments. The search space measurements for each instance, as described earlier, were taken directly from Watson et al.'s experiments. Although FDC did not appear in the original publication, fitness-distance correlations were measured using the 5000 local optima for each instance, and provided to us.

4.2.2 Algorithms

The cost of finding an optimal solution is measured for three algorithms: SGS, randomized restart (Restart), and standard chronological search (Chron). The last two algorithms are implemented through special parameter settings

[6]We thank Jean-Paul Watson for providing us with the problem instances and search space feature measurements from their experiments.

Table 3 Parameter values for the experiments where Restart and Chron are implemented through special parameter settings of SGS

	Fail sequence	$\|e\|$	p	Init. fail bound	Backtrack method
SGS	Luby	1	0	1	chron
Restart	Luby	1	1	1	chron
Chron	∞	1	1	1	chron

of SGS. Randomized restart is replicated by never guiding with a solution (i.e., setting $p = 1$). Chronological search is implemented with SGS using $p = 1$ along with a fail sequence that never restarts search: $\{\infty\}$. These two settings result in one backtracking search which terminates when the optimal solution is found. SGS always guides by a solution, $p = 0$, and uses the Luby et al. (1993) fail sequence: $\{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 4, 8, \ldots\}$. The Luby sequence is used here instead of the polynomial sequence, as in Experiment 2, because it is independent of the algorithms success in finding new best solutions. Recall that in the polynomial sequence, the fail limit is reset to 32 whenever a new best solution is found. In contrast, the Luby sequence is followed regardless of the success or failure of a given tree search to find a new best solution. Following such an independent sequence removes a source of variance from our experiments. This implementation of Restart and Chron through a parameterization of SGS incurs a run-time overhead in keeping track of the (unused) elite pool. Such an overhead is relevant for Experiment 2, above, where we measured run-time and so this approach was not used. Here, we measure the size of the search tree, not run-time, and there is no time-limit on the runs. Therefore, the run-time overhead does not effect the results and it was judged better to minimize differences in the underlying code among the three approaches.

Displayed in Table 3 are the parameter settings used by each algorithm. Again from past good performance and as a means to simplify SGS, only one elite solution is maintained. Due to the small size of problem used, a fail limit of 1 is used in all cases to generate the initial elite solution. In the Restart and Chron algorithms, this initial elite solution is only used to set the initial upper bound on the makespan.

The same 10% randomized texture-based heuristic (Beck and Fox 2000) and constraint propagation techniques for scheduling used Sect. 3 are used again here for all algorithms. All algorithms were implemented in ILOG Scheduler 6.3 and run on a 2 GHz Dual Core AMD Opteron 270 with 2 GB RAM running Red Hat Enterprise Linux 4.

Since all algorithms contain a degree of randomness, search cost for each instance and algorithm is measured over 1000 independent runs with different random seeds. The measure of search cost for each instance ($cost_{\text{med}}$) is the median number of choice-points—over the 1000 independent runs—needed to *find*, but not prove, the optimal solution. We limit ourselves to finding an optimal solution to be consistent with the previous work of search space features, referenced above, which was done in the context of local search. We believe that examining the relation of proof-time to search space features is an interesting direction that is beyond the scope of this work.

Table 4 Mean, median and standard deviation of $cost_{\text{med}}$ for the original and weakened versions of the three algorithms over the 1000 instances

	$mean(cost_{\text{med}})$	$median(cost_{\text{med}})$	$std.dev.(cost_{\text{med}})$
SGS	256.25	264.59	60.92
Restart	249.00	257.00	58.41
Chron	169.00	173.43	31.74
SGS_{weak}	635.25	980.78	1329.80
$\text{Restart}_{\text{weak}}$	930.25	1059.13	560.29
$\text{Chron}_{\text{weak}}$	336.75	517.72	660.50

4.2.3 *Initial results*

Various statistics of $cost_{\text{med}}$ over the 1000 instances are shown at the top of Table 4. Scatter plots of the first experiments are displayed in the left sides of Figs. 7–11. In all plots, the y-axis is on a log scale, and the least-squares fit line is included. Pearson r and r^2 values are shown in the top half of Table 5.

Log–log scatter plots of $|optsols|$ versus $cost_{\text{med}}$ for the three algorithms appear on the left side of Fig. 7. The low r values of -0.1829, -0.1814, and 0.0975 for SGS, Restart and Chron respectively suggest the log of the number of solutions is a poor static cost model of JSP difficulty for any of the three constructive search algorithms.

Similar, inverted, results can be seen for $|backbone|^2$ in Fig. 8 with r values of 0.2094, 0.2055, and -0.0123 for the same three algorithms. Since we are using the exact same instances as Watson et al., the similarity in the $|optsols|$ and $|backbone|^2$ correlations can be attributed to the high correlation between $log(|optsols|)$ and $|backbone|^2$ ($r = -0.9103$) first noticed by Watson et al.

More positive results can be seen for the average distance between local optima $\overline{loptdist}$. Scatter plots for each algorithm (Fig. 9 Left) show a slight upward trend. The r values, 0.6445, 0.6536 and 0.5856 for SGS, Restart and Chron respectively, are all larger than that found by Watson et al. for tabu search (0.5238). Surprisingly, the r values for all algorithms are quite similar.

Scatter plots of the square root of the distance between local optima and the nearest optimal solution $\sqrt{\bar{d}_{\text{lopt-opt}}}$ versus $cost_{\text{med}}$ (log scale) appear in the left of Fig. 10. The r-values of 0.6026, 0.6150, 0.4230 for the three algorithms are slightly less than those found for $\overline{loptdist}$.

Table 5 Pearson's correlation coefficients $r(r^2)$ between $log(cost_{\text{med}})$ and the various search space features for the original three algorithms, the three algorithms with weakened propagation, and the original results for tabu search by Watson et al. (2003)

	$\lvert optsols\rvert$	$\lvert backbone\rvert^2$	$\overline{loptdist}$	$\sqrt{\bar{d}_{\text{lopt-opt}}}$	FDC
SGS	−0.1829 (0.0334)	0.2091 (0.0437)	0.6445 (0.4154)	0.6026 (0.3631)	−0.2025 (0.0410)
Restart	−0.1814 (0.0329)	0.2055 (0.0422)	0.6536 (0.4272)	0.6150 (0.3782)	−0.2290 (0.0524)
Chron	0.0957 (0.0092)	−0.0123 (0.0002)	0.5856 (0.3430)	0.4230 (0.1789)	−0.1577 (0.0249)
SGS_{weak}	−0.2030 (0.0412)	0.2180 (0.0475)	0.5830 (0.3398)	0.6627 (0.4392)	−0.4554 (0.2074)
$\text{Restart}_{\text{weak}}$	−0.0522 (0.0027)	0.1018 (0.0104)	0.7212 (0.5201)	0.5736 (0.3290)	−0.3534 (0.1249)
$\text{Chron}_{\text{weak}}$	0.0654 (0.0043)	−0.0202 (0.0004)	0.7039 (0.4954)	0.5419 (0.2936)	−0.4790 (0.2294)
TS_{Taillard}	−0.4715 (0.2223)	0.4723 (0.2231)	0.5238 (0.2744)	0.8088 (0.6541)	−0.3433 (0.1178)

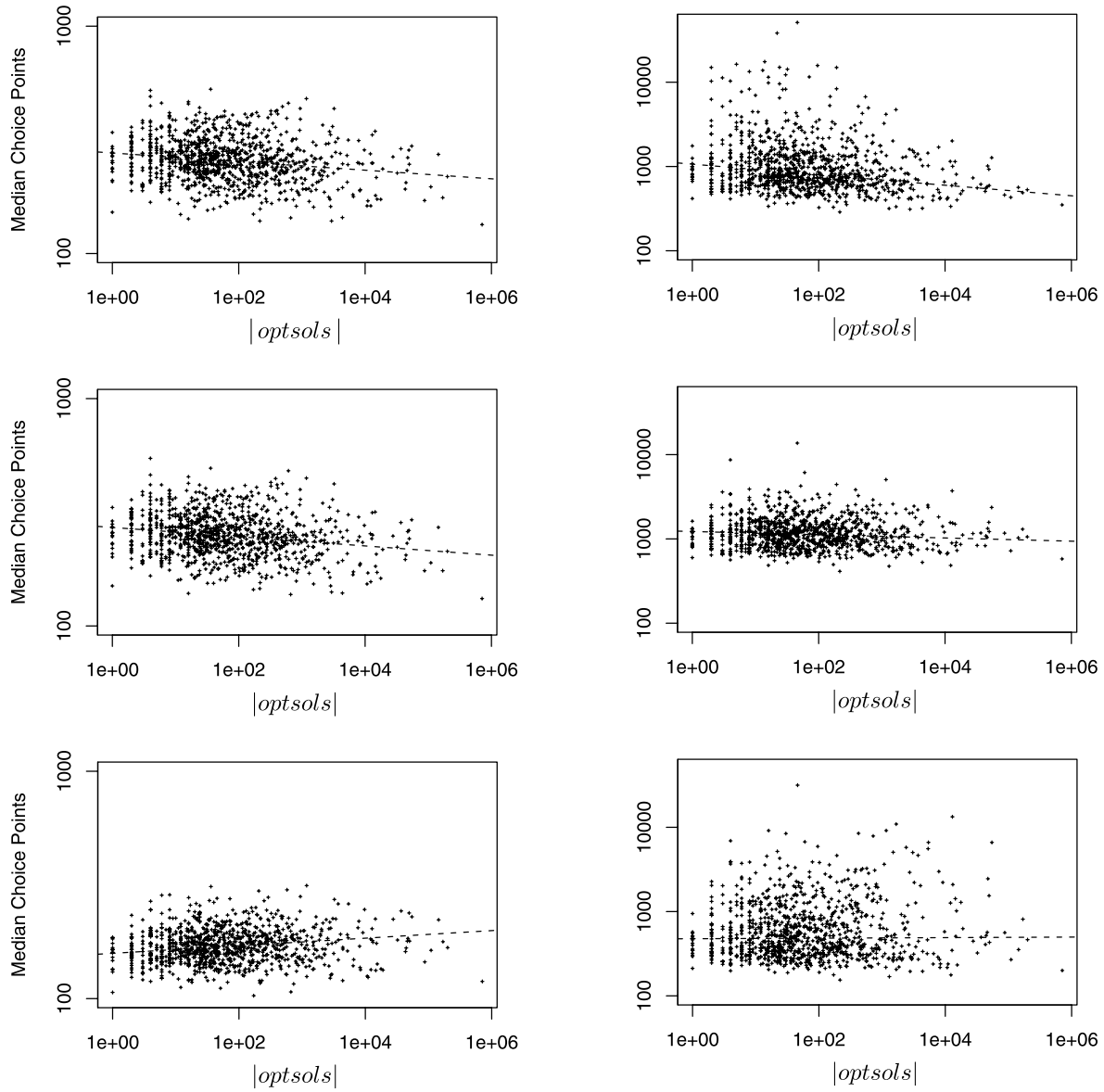

Fig. 7 Log–log scatter plots and least-squares best-fit lines of $|optsols|$ versus $cost_{\text{med}}$ for the original *left* and weakened *right* versions of SGS *top*, Restart *middle*, and Chron *bottom*

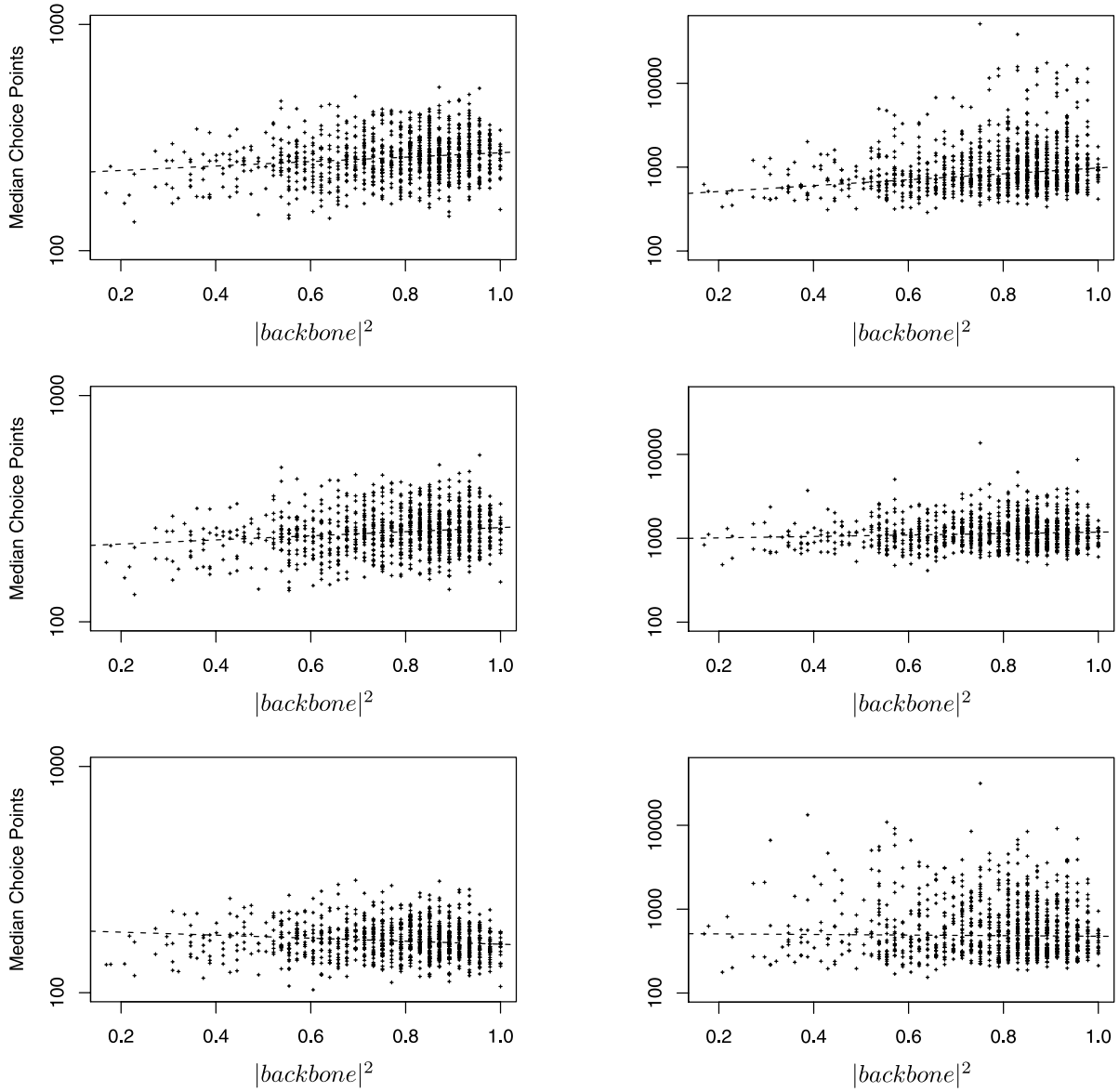

Fig. 8 Scatter plots and least-squares best-fit lines of $|backbone|^2$ versus $cost_{\text{med}}$ for the original *left* and weakened *right* versions of SGS *top*, Restart *middle*, and Chron *bottom* (log-scale y-axis)

Scatter plots of FDC versus log-scale $cost_{\text{med}}$ for the three algorithms are shown on the left side of Fig. 11. The r values, -0.2025, -0.2290, and -0.1577, for the three algorithms suggest that FDC is a poor predictor of search cost for these instances.

4.3 Experiment 5: SGS with weaker propagation

A weakness in the above experiment arises from the small problem size. It may be that the algorithms were mainly finding the optimal solutions through the powerful propagation techniques of constraint programming, and less through the individual backtracking techniques of the varying algorithms. Gomes et al. (2000, p. 81) noted a similar experience on smaller quasigroup-with-holes problems. When the all-different global constraints were used, the heavy-tailed distribution in run-times disappeared on small problems, which they attributed to the fact that the problems were primarily solved through propagation rather than search.

In this second experiment, all algorithms are modified in an attempt to increase the amount of search on these smaller instances by changing the variable ordering and level of constraint propagation. The variable ordering is changed from a randomized texture heuristic to a completely random variable ordering. The enforcement level on the precedence and capacity constraints are set back to the default level used by the ILOG Scheduler (2006), down from the medium-high level used previously. This lower level does not main-

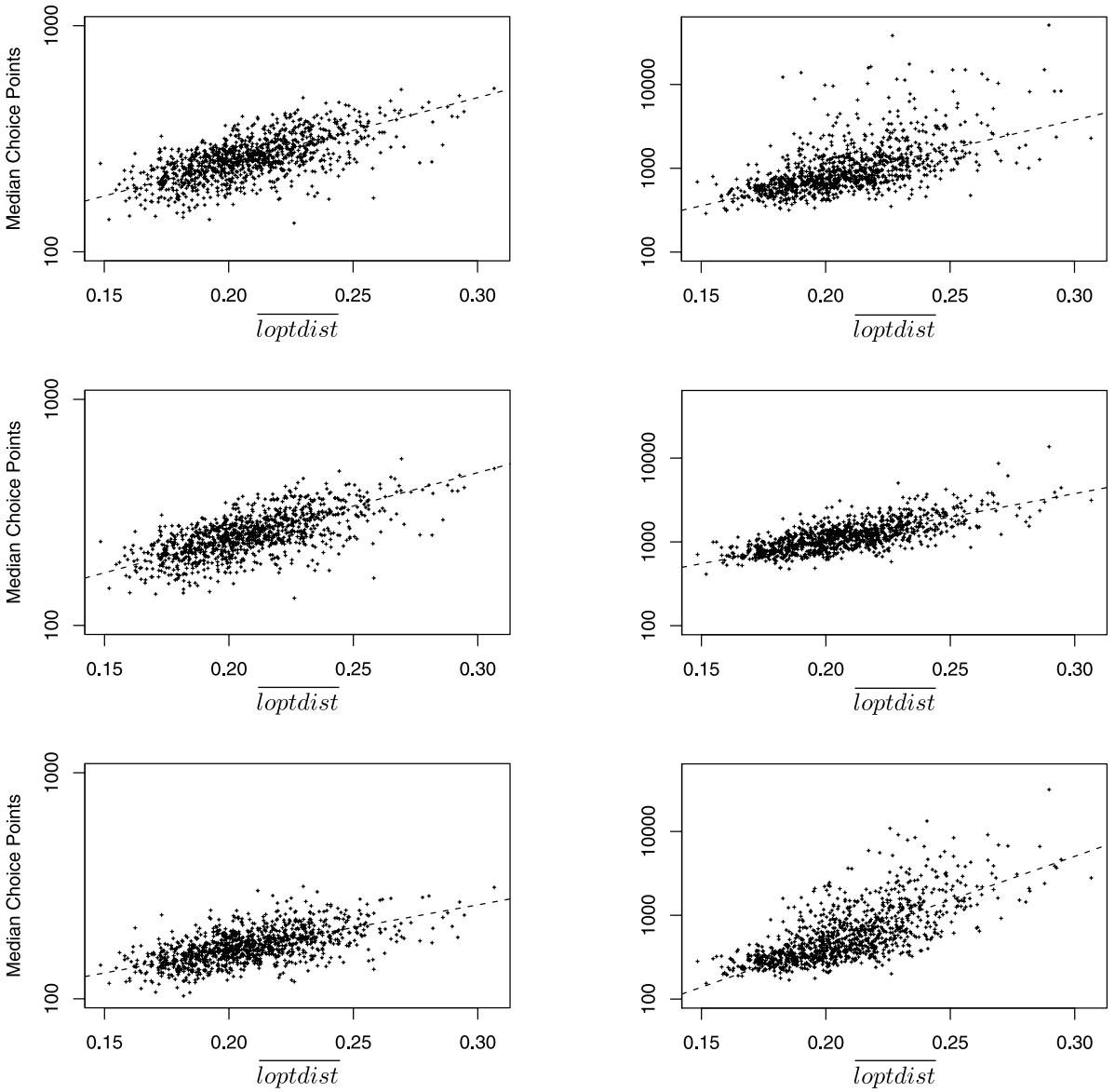

Fig. 9 Scatter plots and least-squares best-fit lines of $\overline{loptdist}$ versus $cost_{\text{med}}$ for the original *left* and weakened *right* versions of SGS *top*, Restart *middle*, and Chron *bottom* (log-scale y-axis)

tain the global constraint on precedence (Laborie 2003), and removes the edge-finder algorithm (Nuijten 1994) from the unary resource constraint enforcement. The weakened $\text{Restart}_{\text{weak}}$ and $\text{Chron}_{\text{weak}}$ algorithms are implemented in the exact same way using the weakened version of SGS with the parameters in Table 3. All other experimental details are the same.

4.3.1 Results

As shown in Table 4, for the weakened algorithms, SGS now outperforms Restart, yet Chronological still performs the best in both cases. Based on the results of Sect. 3, our problem size appears to be too small for SGS to outperform standard backtracking even for the weakened algorithms.

Scatter plots and least-squares fit lines of the search space features versus the median search cost (log scale) of the weakened algorithms appear on the right sides of Figs. 7–11. The plots are similar to the plots for the original algorithms, although the range of the log-scale y-axis is now ten times larger. Pearson r and r^2 values are displayed in Table 5.

For $|optsols|$ and $|backbone|^2$, the r values for SGS_{weak} remained about the same at -0.2030 and 0.2180. The results for $\text{Restart}_{\text{weak}}$ are now as low as those of $\text{Chron}_{\text{weak}}$, with $|optsols|$ r values of -0.0522 and 0.0654 and $|backbone|^2$ r values of 0.1018 and -0.0202. Yet, all correlations for these two features still remain very low.

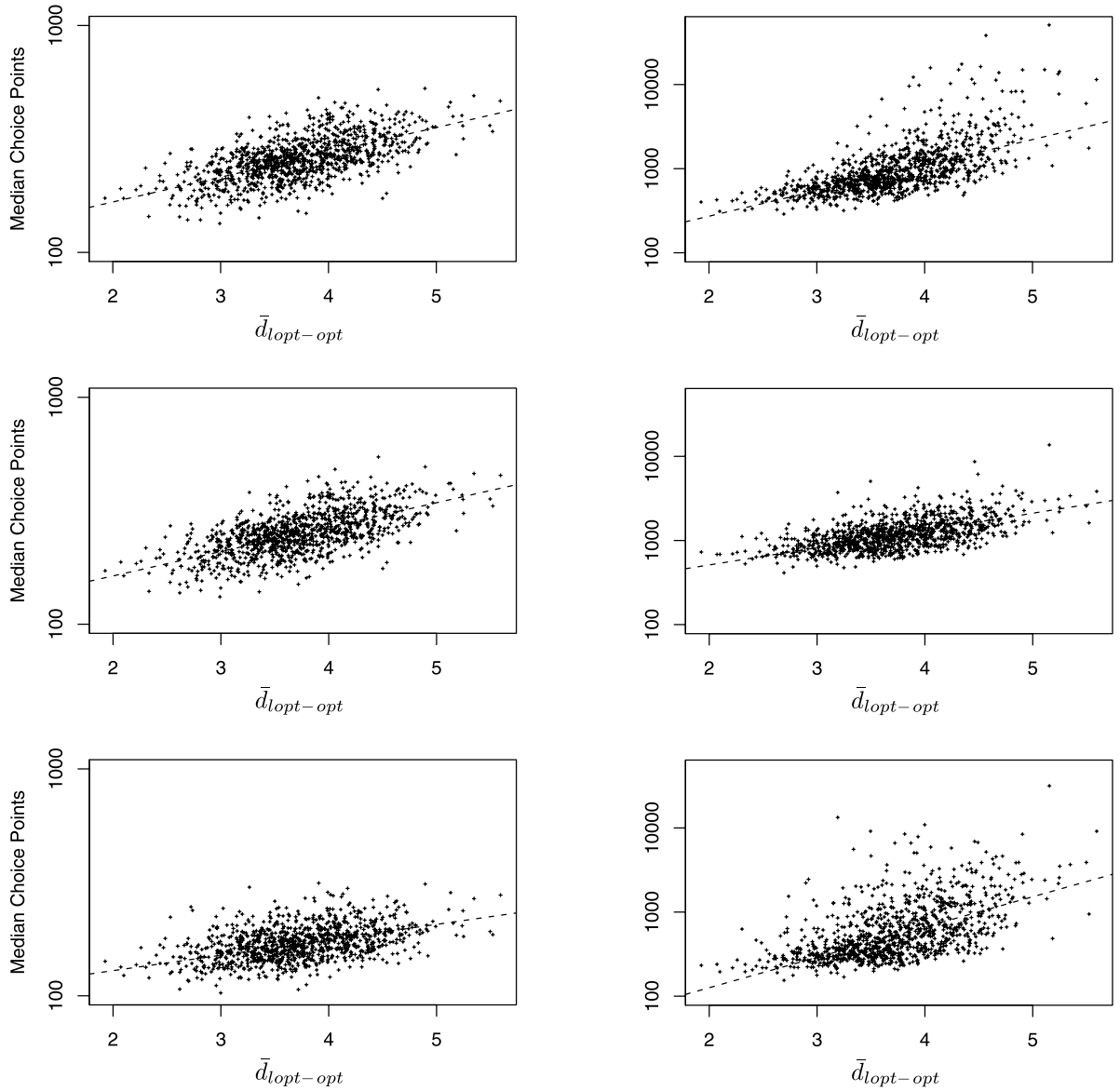

Fig. 10 Scatter plots and least-squares best-fit lines of $\sqrt{\bar{d}_{\text{lopt-opt}}}$ versus $cost_{\text{med}}$ for the original *left* and weakened *right* versions of SGS *top*, Restart *middle*, and Chron *bottom* (log-scale y-axis)

Results for $\overline{loptdist}$ are similar to those of the original algorithms: slightly lower for SGS_{weak}, and slightly higher for $\text{Restart}_{\text{weak}}$ and $\text{Chron}_{\text{weak}}$. The r-value for $\text{Restart}_{\text{weak}}$ and $\overline{loptdist}$ is the highest seen in any of the experiments $r, (r^2) = 0.7212, (0.501)$.

Results for $\sqrt{\bar{d}_{\text{lopt-opt}}}$ are also similar to the first experiment. This time, the $r(r^2)$ values of SGS_{weak} are slightly higher at 0.6627 (0.4392).

The correlations for FDC show the greatest change from the previous experiment with all correlations being stronger. Surprisingly, $\text{Chron}_{\text{weak}}$ has the strongest correlation, with an r value three times as large as the one found with the original algorithm (-0.4790 vs. -0.1577).

4.4 Discussion

For *all* constructive algorithms investigated, results were surprisingly similar to those found by Watson et al. with tabu search. That these correlations are observed in Restart and Chron is quite unexpected. This is the first evidence that we are aware of that demonstrates a correlation between such search space features and standard constructive search techniques.

It should be emphasized that these experiments show *correlation* between search space structure and algorithm performance. It is natural, but perilous, to use such correlation as a basis for conjectures about causation. Knowing the tabu search algorithm, it is reasonable to conjecture that the

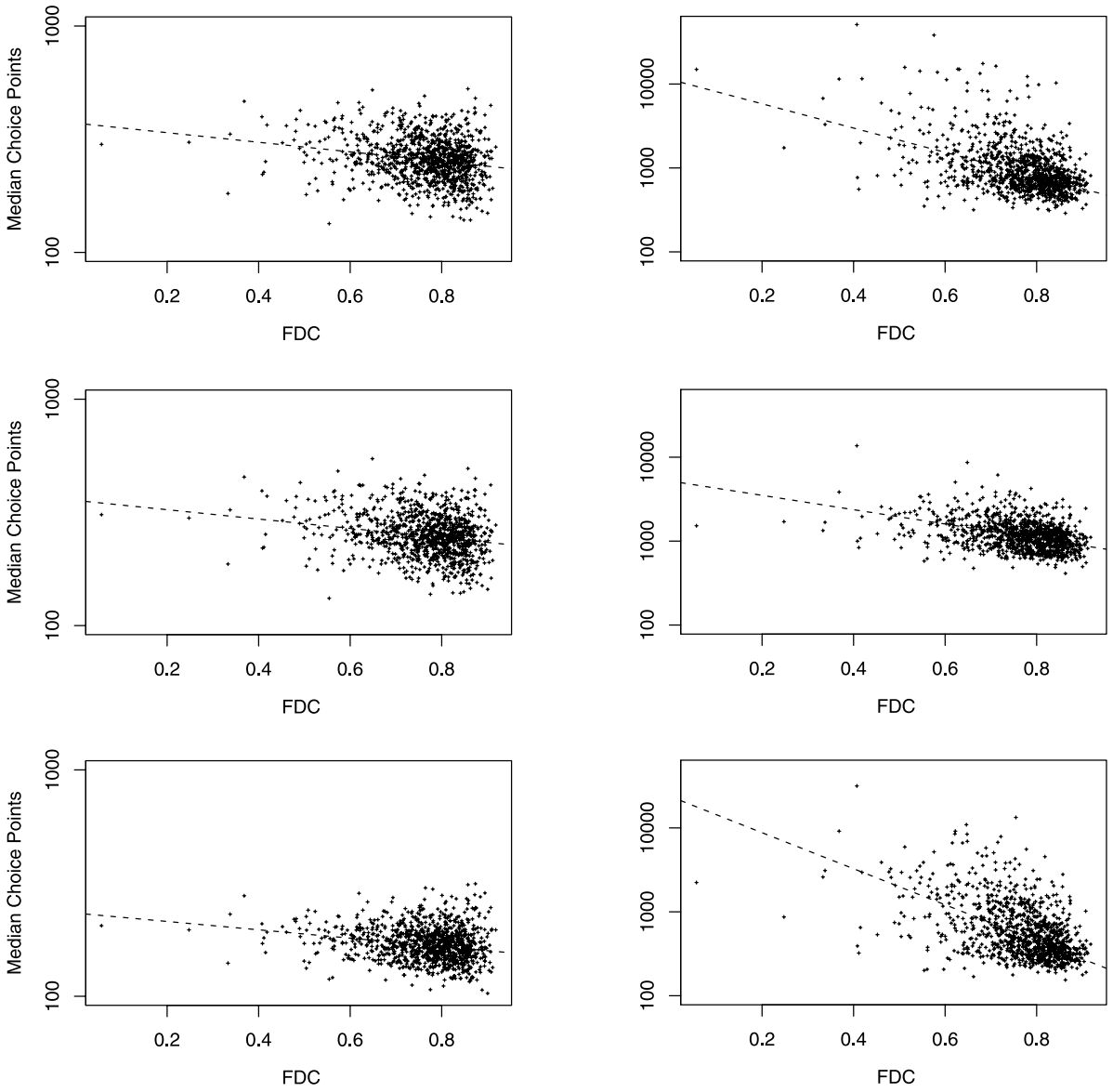

Fig. 11 Scatter plots and least-squares best-fit lines of *FDC* versus $cost_{\text{med}}$ for the original *left* and weakened *right* versions of SGS *top*, Restart *middle*, and Chron *bottom* (log-scale y-axis)

correlation between tabu-search performance and $\sqrt{\bar{d}_{\text{lopt-opt}}}$ indicates a causal relationship: the low mean distance between high-quality local optima and global optima mean that by following the cost gradient and therefore finding local optima, tabu search is likely to also be searching close to global optima. This proximity raises the likelihood of finding a global optima and hence, increases the performance of tabu search. A similar conjecture can be made with respect to SGS: since it is guided by high-quality solutions, when high-quality solutions are close to optimal solutions, search performance is improved. This, indeed, is what we set out to test in this experiment.

However, consistency demands that we apply the same conjectures to Restart and Chron. Correlations between search cost for these techniques and search space features are, in some cases, stronger than those of SGS and tabu search. We are left with two options: either there exists some mechanism by which Chron and Restart performance are influenced by static search space features or there exist some more fundamental determinants of search performance that influence the search space features we have measured. The design and analysis of experiments that address these two options is a rich area for future work.

One of the limitations of this study is the fact that our search space structure measurements are precisely those used by Watson et al. in the context of tabu search. While the variables used in the constructive search approaches, the tabu search of Watson et al., and the search space mea-

surements are the same (i.e., the sequence of each pair of activities on the same resource), it is not clear that a local optima for a particular neighborhood function in tabu search should have any meaning for constructive search. It would be interesting to formalize a notion of distance more suited to a search tree and to repeat the above experiments.

5 Conclusion

This paper has addressed two questions:

1. Does SGS, like randomized restart, avoid the long runs that are characteristic of heavy-tailed distributions? Our hypothesis was that one factor for the good performance of SGS is the exploitation of heavy tails. We showed that SGS theoretically benefits in the same way as other rapid randomized restart techniques when heavy-tailed distributions are present. In the experiments, we confirm that the distribution in run-times of the backtracking search on JSP instances can be heavy-tailed. It was shown that SGS and randomized restart techniques perform better at the same instances in which heavy tails start to appear. Guided runs used by SGS were also shown to exhibit heavy tails.
2. Does SGS, like local search, respond to topological structure in the search space? By evaluating the static cost models from Watson et al. (2003) on three constructive search algorithms, we were able to find correlations between SGS behavior and various search space features. Surprisingly, correlations of similar, or greater, strength were found for algorithms that do not explicitly guide search with sub-optimal solutions (randomized restart, standard chronological search). Stronger correlations were found for models based on average distance between local optima $\overline{loptdist}$, where the strongest relationships of any of our experiments were found for randomized restart (Restart) and chronological search (Chron). Slightly stronger correlations were also found between fitness-distance correlation and all constructive algorithms with weakened propagation. The best predictor of search cost for tabu search, based on the distance between local optima and the closest optimal solution $\sqrt{\bar{d}_{\text{lopt-opt}}}$, did give the best model for SGS, although it was weaker than that of tabu search, and $\overline{loptdist}$ with Chron and Restart. From the correlations measured, local-search based search space features do seem to be related to the performance of all three constructive search algorithms investigated. It is unclear what mechanisms may be at work to explain these correlations for constructive search algorithms that are not explicitly being guided by high-quality solutions. We believe this is the first time such an analysis has been applied to such constructive search algorithms.

There are a number of directions that could be pursued based on this work. First is the question, noted above, of an explanation for the correlation of chronological backtracking and randomized restart search with search space structure relevant to local search techniques.

We have only focused on job-shop scheduling optimization problems and, indeed, a single optimization function (makespan minimization). The hypotheses that formed the basis for this paper should be tested for other optimization and satisfaction problems. In particular, we speculate that the impact of back-propagation from the objective function may play a role in the observed correlation between search space features and the constructive search techniques that are not explicitly guided by elite solutions. Investigating problems with less back-propagation (e.g., job-shop scheduling to minimize tardiness) and with no back-propagation (e.g., constraint satisfaction problems) may help to explain these results.

One limitation of our experiments, especially those related to static cost models, was the computational cost of using large problem instances. As shown in Sect. 3, SGS only out-performs the other techniques tested at larger problem sizes. To better understand how SGS works, we should investigate it in conditions where it does out-perform the other techniques. The 6×6 job-shop instances were taken from Watson et al. (2003), published in 2003. Speed of computers have increased since then, so it may be feasible to consider moderately larger JSP instances. However, enumerating all solutions to 16×16 problems, where the performance difference is first observed, still seems somewhat challenging. We may therefore consider relying on statistical techniques (e.g., Achlioptas et al. 2000) to estimate search space feature measurements.

Acknowledgements This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ontario Research Fund, Microway, Inc., and ILOG, S.A.

References

Achlioptas, D., Gomes, C. P., Kautz, H. A., & Selman, B. (2000). Generating satisfiable problem instances. In *Proceedings of the seventeenth national conference on artificial intelligence* (pp. 256–261).

Baptista, L., & Silva, J. P. M. (2000). Using randomization and learning to solve hard real-world instances of satisfiability. In *Principles and practice of constraint programming* (pp. 489–494).

Beck, J. C. (1999). *Texture measurements as a basis for heuristic commitment techniques in constraint-directed scheduling*. PhD thesis, University of Toronto.

Beck, J. C. (2005). Multi-point constructive search. In *Proceedings of the eleventh international conference on principles and practice of constraint programming (CP'05)* (pp. 737–741).

Beck, J. C. (2005). Multi-point constructive search: Extended remix. In *Proceedings of the CP2005 workshop on local search techniques for constraint satisfaction* (pp. 17–31).

Beck, J. C. (2006). An empirical study of multi-point constructive search for constraint-based scheduling. In *Proceedings of the sixteenth international on automated planning and scheduling (ICAPS06)* (pp. 274–283).

Beck, J. C. (2007). Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research*, *29*, 49–77.

Beck, J. C., & Fox, M. S. (2000). Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, *117*(1), 31–81.

Beck, J. C., & Watson, J.-P. (2003). Adaptive search algorithms and fitness-distance correlation. In *Proceedings of the fifth meta-heuristics international conference*.

Blazewicz, J., Domschke, W., & Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, *93*(1), 1–33.

Chen, H., Gomes, C. P., & Selman, B. (2001). Formal models of heavy-tailed behavior in combinatorial search. In *CP'01: Proceedings of the 7th international conference on principles and practice of constraint programming* (pp. 408–421). London: Springer.

Clark, D. A., Frank, J., Gent, I. P., MacIntyre, E., Tomov, N., & Walsh, T. (1996). Local search and the number of solutions. In *Proceedings of the second international conference on principles and practice of constraint programming (CP'96)* (pp. 119–133). Berlin: Springer.

Gent, I. P., MacIntyre, E., Prosser, P., & Walsh, T. (1996). The constrainedness of search. In *Proceedings of the thirteenth national conference on artificial intelligence (AAAI-96)* (Vol. 1, pp. 246–252).

Gomes, C. P., & Selman, B. (1999). Search strategies for hybrid search spaces. In *Proceedings of the 11th IEEE international conference on tools with artificial intelligence* (p. 359). Los Alamitos: IEEE Computer Society.

Gomes, C. P., Selman, B., & Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the fifteenth national conference on artificial intelligence (AAAI-98)* (pp. 431–437).

Gomes, C. P., Selman, B., Crato, N., & Kautz, H. A. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, *24*(1/2), 67–100.

Gomes, C. P., Fernàndes, C., Selman, B., & Bessiere, C. (2004). Statistical regimes across constrainedness regions. In *Proceedings of the tenth international conference on the principles and practice of constraint programming (CP2004)* (pp. 32–46).

Gomes, C. P., Fernández, C., Selman, B., & Bessière, C. (2005). Statistical regimes across constrainedness regions. *Constraints*, *10*(4), 317–337.

Heckman, I. (2007). *Empirical analysis of solution guided multi-point constructive search*. Master's thesis, Department of Computer Science, University of Toronto.

Heckman, I., & Beck, J. C. (2007). Fitness-distance correlation and solution-guided multi-point constructive search. In L. Perron & M. Trick (Eds.), *Proceedings of the fourth international conference on integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR'07)* (pp. 112–126). Berlin: Springer.

Jones, T., & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. Eshelman (Ed.), *Proceedings of the sixth international conference on genetic algorithms* (pp. 184–192). San Francisco: Morgan Kaufmann.

Laarhoven, P. J. M., Aarts, E. H. L., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, *40*(1), 113–125.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, *143*, 151–188.

Le Pape, C. (1994). Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, *3*(2), 55–66.

Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, *47*, 173–180.

Mattfeld, D. C., Bierwirth, C., & Kopfer, H. (1999). A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, *86*, 441–453.

Mitchell, D., Selman, B., & Levesque, H. (1992). Hard and easy distributions of SAT problems. In *Proceedings of the tenth national conference on artificial intelligence (AAAI-92)* (pp. 459–465).

Nuijten, W. P. M. (1994). *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Parkes, A. J. (1997). Clustering at the phase transition. In *Proceedings of the fourteenth national conference on artificial intelligence (AAAI-97)* (pp. 340–345). Providence, RI.

Scheduler (2006). *ILOG Scheduler 6.2 User's manual and reference manual*. ILOG, S.A.

Singer, J., Gent, I. P., & Smaill, A. (2000). Backbone fragility and local search cost peak. *Journal of Artificial Intelligence Research*, *12*, 235–270.

Smith, B. M., & Dyer, M. E. (1996). Locating the phase transition in constraint satisfaction problems. *Artificial Intelligence*, *81*, 155–181.

Watson, J.-P. (2003). *Empirical modeling and analysis of local search algorithms for the job-shop scheduling problem*. PhD thesis, Dept. of Computer Science, Colorado State University.

Watson, J.-P., & Beck, J. C. (2008). A hybrid constraint programming/local search approach to the job-shop scheduling problem. In *Proceedings of the fifth international conference on integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR'08)* (pp. 263–277).

Watson, J.-P., Barbulescu, L., Whitley, L. D., & Howe, A. E. (2002). Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. *INFORMS Journal on Computing*, *14*(2), 98–123.

Watson, J.-P., Beck, J. C., Howe, A. E., & Whitley, L. D. (2003). Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, *143*(2), 189–217.

Williams, R., Gomes, C., & Selman, B. (2003). On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Proceedings of sixth international conference on theory and applications of satisfiability testing (SAT-03)*.

Wu, H., & van Beek, P. (2007). On universal restart strategies for backtracking search. In *Proceedings of the thirteenth international conference on the principles and practice of constraint programming (CP 2007)* (pp. 681–695).

Yokoo, M. (1997). Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Principles and practice of constraint programming* (pp. 356–370).