An Empirical Study of Multi-Point Constructive Search for Constraint-Based Scheduling

J. Christopher Beck
Department of Mechanical & Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca

Abstract

Multi-point constructive search (MPCS) performs a series of resource-limited backtracking searches where each search begins either from an empty solution (as in randomized restart) or from a solution that has been encountered during the search. We perform a systematic study of MPCS to evaluate the performance impact of various parameter settings. Results using job shop scheduling instances with two different optimization criteria, demonstrate that MPCS-based search is significantly better than standard chronological backtracking and randomized restart while raising a number of questions as to the underlying reasons for the observed performance.

Introduction

A number of metaheuristic and evolutionary approaches to optimization make use of multiple "viewpoints" by maintaining a set of promising solutions that are used to guide search. In evolutionary algorithms, a population of solutions is maintained and combined to produce a subsequent set of solutions which are then filtered based on quality. In metaheuristics, such as advanced tabu search (Nowicki & Smutnicki 2005), a set of high quality solutions are maintained and revisited in order to intensify search in promising areas of the search space.

Multi-point Constructive Search (MPCS) (Beck 2005a) is a framework designed to allow constructive search to exploit multiple viewpoints. As with randomized restart techniques (Gomes, Selman, & Kautz 1998), search consists of a series of tree searches that are limited by some resource bound, typically a maximum number of fails. When the resource bound is reached, search restarts. The difference with randomized restart is that MPCS keeps track of a small set of "elite solutions": the best solutions it has found. When the resource bound on a search is reached, the search is restarted either from an empty solution as in randomized restart, or from one of the elite solutions. Preliminary empirical results on job shop scheduling problems and quasi-group completion problems have shown that MPCS-based search can significantly out-perform both standard chronological backtracking and randomized restart (Beck 2005a).

In this paper, we undertake the first systematic empirical study of MPCS. In particular, we investigate the different parameter settings and their impact on search performance on two related scheduling problems: job shop scheduling with makespan minimization and job shop scheduling with weighted tardiness minimization. Empirical results demonstrate the superiority of MPCS-based techniques over randomized restart and chronological backtracking over a wide range of parameter settings. More interestingly, the results reveal differences in performance on the two closely related problem types studied and, in fact, call into question the original motivation for MPCS.

In the next section, we present the MPCS framework and the parameter space that will be investigated. We then turn to our empirical study, consisting of five experiments varying the identified parameters and an additional experiment to compare the best parameter settings found against chronological backtracking and randomized restart. Finally, we discuss our results and their implications in terms of developing an understanding of the observed performance.

Background

Pseudocode for the basic Multi-Point Constructive Search (MPCS) algorithm is shown in Algorithm 1. The algorithm initializes a set, e, of elite solutions and then enters a while-loop. In each iteration, with probability p, search is started from an empty solution (line 6) or from a randomly selected elite solution (line 12). 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. When an optimal solution is found and proved or when some overall bound on the computational resources (e.g., CPU time, number of fails) is reached, the best elite solution is returned.

The elite solutions can be initialized by any search technique. In this paper, we use independent runs of a standard chronological backtracking with a randomized heuristic and do not constrain the cost function. The search effort is limited by a maximum number of fails for each run. We assume that (probably very poor) solutions can be easily found. We do not start the while-loop with an empty elite set in order to ensure that the initial solutions are independently generated.

A solution is found from an empty solution (line 6) using any standard constructive search with a randomized heuristic

Algorithm 1: MPCS: Multi-Point Constructive Search

```
   MPCS():
1  initialize elite solution set e
2  while termination criteria unmet do
3      if rand[0,1) < p then
4          set upper bound on cost function
5          set fail bound, b
6          s := search(∅, b)
7          if s ≠ NIL and s is better than worst(e) then
8              replace worst(e) with s
       else
9          r := randomly chosen element of e
10         set upper bound on cost function
11         set fail bound, b
12         s := search(r, b)
13         if s ≠ NIL s is better than r then
14             replace r with s
15 return best(e)
```

and a bound on the number of fails. It is possible that no solution is found within the fail bound.

Each individual search is bounded by a fail bound: a single search (lines 6 and 12) will terminate, returning the best solution found, after the it has failed (i.e., backtracked) the corresponding number of times. We experiment with different policies for setting and increasing the fail bound.

Searching from a Solution

A search tree is created by asserting a series of choice points of the form: $\langle V_i = x\rangle \vee \langle V_i \neq x\rangle$ where V_i is a variable and x the value that is assigned to V_i. Given the importance of variable ordering heuristics in constructive search, we expect that the order of these choice points will have an impact on search performance and so MPCS can use any variable ordering heuristic to choose the next variable to assign. The choice point is formed using the value assigned in the *reference* solution or, if the value in the reference solution is inconsistent, a heuristically chosen value. More formally, let a reference solution, s, be a set of variable assignments, $\{\langle V_1 = x_1\rangle, \langle V_2 = x_2\rangle, \ldots, \langle V_m = x_m\rangle\}, m \leq n$, where n is the number of variables. The variable ordering heuristic has complete freedom to choose a variable, V_i, to be assigned. If $x_i \in dom(V_i)$, the choice point is made with $x = x_i$. Otherwise, if $x_i \notin dom(V_i)$, any value ordering heuristic can be used to choose $x \in dom(V_i)$.

We need to account for the possibility that $x_i \notin dom(V_i)$ because the reference solution is not necessarily a valid solution later in the MPCS search process. To take a simple example, if the reference solution has a cost value of 100 and we now constrain search to solution of less than 100, we will not reach the reference solution. Rather, via constraint propagation, we will reach a dead-end or different solution.

Setting the Bounds on the Cost Function

Before we search (lines 6 and 11), we place an upper bound on the cost function. As we are conducting constraint-based search, the bound may have an impact on the set of solutions that will be searched and, therefore, on the solutions that may enter the elite set. Intuitions from constructive search and metaheuristics differ on the appropriate choice of an upper bound. In single point constructive search for optimization with a discrete cost function, the standard approach is to use $c^* - 1$, as the upper bound, where c^* is the best solution found so far. Using a higher bound would only expand the search space without providing any heuristic benefit. In contrast, in pure local search there is no way to enforce an upper bound and so search space reduction is not an issue. It is common to replace an elite solution when a better, but not necessarily best known, solution is found: since the elite solutions are used to heuristically guide search, even solutions which are not the best known can provide heuristic guidance.

We experiment with the following two approaches to setting the upper bound:

1. *Multi-Point Global Bound (mpgb)*: Always set the upper bound on the search cost to $c^* - 1$.
2. *Multi-Point Local Bound (mplb)*: When starting from an empty solution, set the upper bound to be equal to one less than the cost of the worst elite solution. When starting from an elite solution, set the upper bound to be one less than the cost of the starting solution.

MPCS Parameter Space

The primary purpose of the empirical study is to understand the impact of the different parameter settings on the performance of MPCS algorithms.We study both the global and local bound setting approaches in all experiments while varying a number of other parameters.

Fail Sequence The resource bound sets the number of fails allowed for each search. We look at three different ways of setting and increasing this bound where, for each method, the fail limit is independent of the choice to search from an empty solution or from an elite solution:

- Luby - the fail bound sequence follows the optimal sequence when there is no knowledge about the solution distribution: 1,1,2,1,1,2,4,1,1,2,1,1,2,4,8, ... (Luby, Sinclair, & Zuckerman 1993).
- Geometric (Geo) - the fail bound is initialized to 1 and reset to 1 when a new best solution is found. When a search fails to find a new best solution, the bound is doubled.
- Polynomial (Poly) - the fail bound is initialized to 32 and reset to 32 whenever a new best solution is found. Whenever a search fails to find a new best solution, the bound grows polynomially: 32 is added to the fail limit. The value 32 was chosen to give a reasonable increase in the fail limit on each iteration.

Elite Set Size Previous studies of MPCS used a elite set size of 8. There does not seem to have been significant experimentation in the elite set size in the metaheuristic community. Anecdotally, a hybrid tabu search with an elite set

smaller than six performs much worse than larger elite sets on job shop scheduling problems.[1] In this paper, we experiment with elite sizes of $\{1, 4, 8, 12, 16, 20\}$.

The Proportion of Searches from an Empty Solution The p parameter controls the probability of searching from an empty solution versus searching from one of the elite solutions. Previous studies have examined $p = 0.5$. In this paper, we study $p = \{0, 0.25, 0.5, 0.75, 1\}$.

Initialization Fail Bound The initialization of the elite solutions is done by independent runs of a randomized chronological backtracking limited by a fail bound. A separate run is done for each elite solution. To experiment with the balance between the quality of the starting solutions and the effort spent, we vary the fail bound on each independent run as follows: 1, 10, 100, 1000, 10000.

Backtrack Method Finally, we experiment with the form of the individual searches. Whether search begins from an empty solution or an elite solution, we have a choice as to how the search should be performed. In particular, we will experiment with using standard chronological backtracking or limited discrepancy search (LDS) (Harvey 1995). In either case, the search is limited by the fail bound as described above. It is interesting to note that such a choice can also be made in randomized restart search, but we were unable to find previous work that experimented with using LDS instead of chronological backtracking.

Empirical Study

In this section, we present five experiments into the impact of varying the parameters described above and one experiment comparing the best parameter values to existing search techniques. In some cases, especially when there are discrepancies in the results for the two problem types, we conduct further experiments to elucidate the relationships among the parameter values and problem types.

Experimental Details

Two related problem types are used in our experiments: job shop scheduling problems (JSPs) with the objective of minimization of either makespan or weighted tardiness. We refer to these problems as *makespan* and *tardy* JSPs respectively. The problem details are as follows:

- Makespan JSPs: Twenty 20×20 problem instances were generated using an existing generator (Watson *et al.* 2002). The durations of the operations are independently drawn with uniform probability from [1, 99]. The machine routings are generated to create work-flow problems where each job visits the first 10 machines before any of the second 10 machines. Within the two machine sets, the routings are generated randomly with uniform probability. Work-flow JSPs were used as they have been shown to be more difficult than JSPs with random machine routings (Watson 2003).
- Tardy JSPs: Ten 15×15 work-flow problem instances were generated in the same way as above with seven machines in the first partition and eight in the second. Each job was assigned the same due date equal to the Taillard lower bound on the *makespan* of the problem instance (Taillard 1993). These due dates were assigned to make it very unlikely that any problem would have a optimal solution with a cost of zero. Weights were independently assigned to each job uniformly from the interval [1, 9].

All experiments are run with a 20 CPU minute time-out and each problem instance is solved 10 times independently. All algorithms are implemented in ILOG Scheduler 6.0 and run on a 2.8GHz Pentium 4 with 512Mb RAM running Fedora Core 2.

In all experiments, the dependent variable is mean relative error (MRE) relative to the best solution known for the problem instance. The MRE is the arithmetic mean of the relative error over each run of each problem instance:

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

where K is a set of problem instances, R is a set of independent runs with different random seeds, $c(a, k, r)$ is the lowest cost found by algorithm a on instance k in run r, and $c^*(k)$ is the lowest cost known for k. As these problem instances were generated for these experiments, the best known solution was found either by the algorithms tested here or by variations used in preliminary experiments.[2]

For all MPCS algorithms and both problem types, texture-based heuristics (Beck & Fox 2000) are used to identify a resource and time point with maximum contention among the activities and then choose a pair of unordered activities, branching on the two possible orders. 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. When starting search from an elite solution, the same code is used to choose a pair of activities to be sequenced and the ordering found in the solution is asserted. The standard constraint propagation techniques for scheduling (Nuijten 1994; Laborie 2003; Le Pape 1994) are used for all algorithms.

Default Parameters Based on preliminary experimentation and the parameter values adopted in previous studies, we identified the default parameter values shown in Table 1 for each problem type. Unless otherwise specified, these are the values that are used in the experiments below.

Problem Type	Fail Seq.	$\|e\|$	p	Initial Fail Bound	Backtrack Method
Makespan	Luby	8	0.5	1000	chron
Tardy	Poly	8	0.5	1000	chron

Table 1: The default parameters for the MPCS algorithms on the makespan and tardy JSP problems.

[1] Jean-Paul Watson – personal communication.

[2] Problem instances and best known solutions are available from the author.

Experiment 1: Fail Sequence

The results of running mpgb and mplb with each fail sequence on the makespan problems are displayed in Figure 1. With both mpgb and mplb, the Luby fail sequence results in the best performance. The Poly policy has better performance with low run-times (less than 300 seconds for mpgb and less than 700 seconds for mplb) but is later surpassed by the methods using Luby. The Geo policy is a distant third in both cases. The overall best performance is observed when mpgb is combined with Luby.

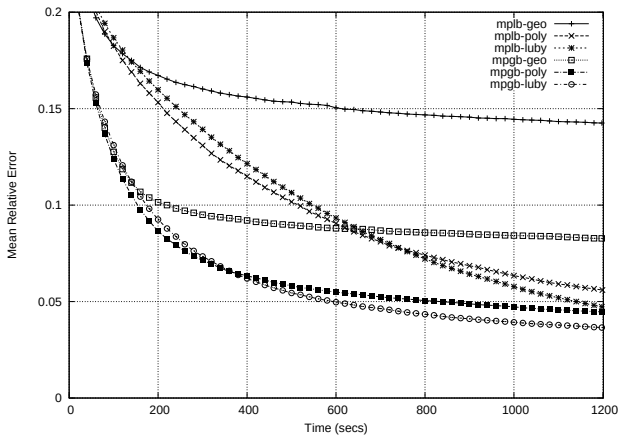

Figure 1: The mean relative error on makespan JSPs for three different fail sequences for *mpgb* and *mplb*.

The results for tardy JSPs are displayed in Figure 2. The y-axis is substantially different from that used with the makespan JSPs. This is because the algorithms have a much higher MRE on the tardy JSPs (i.e., by almost an order of magnitude). Because the best solution is found by the methods that we are experimenting with, the lower MRE indicates a much higher variance of solution quality compared to the makespan JSPs. Runs with identical parameters (except for the random seed) had a wide variance in performance. This behaviour is seen through-out our experiments with tardy JSPs. We believe that this difference is an effect of the poorer back-propagation from the tardiness optimization criteria leading less time window pruning by the global constraints and, as a consequence, a degradation of the quality of the texture-based heuristics.

Figure 2 indicates that for tardy JSPs, Poly appears to perform best. With mplb, Poly is better at all time points while with mpgb the difference between Poly and Luby is marginal, though Poly is at least as good as Luby at all time points. In both conditions, the Geo fail sequence performs worst. Overall, mpgb-poly delivers the lowest MRE.

Experiment 2: Elite Set Size

In this experiment, we vary the size of the elite set. Six different values are used: 1, 4, 8, 12, 16, 20. As each problem type and cost bound method (i.e., mpgb and mplb) result in six different performance curves, we display the results in Figures 3 through 6.

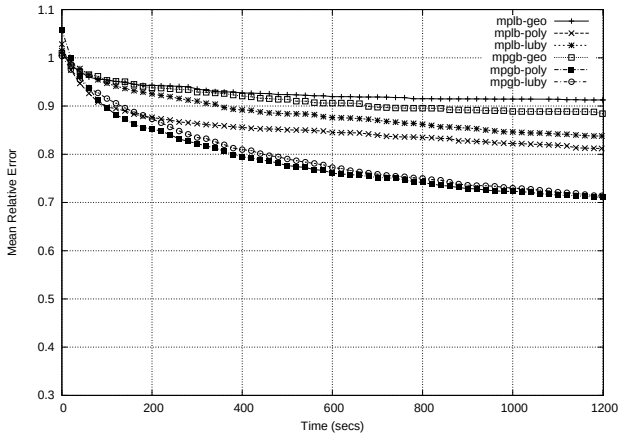

Figure 2: The mean relative error on the tardy JSPs for each fail sequences and for *mpgb* and *mplb*.

For the makespan JSPs (Figures 3 and 4), the *smallest* elite size leads to the best performance. Note that with $|e| = 1$, mpgb and mplb are identical because the local cost bound equals the global cost bound.

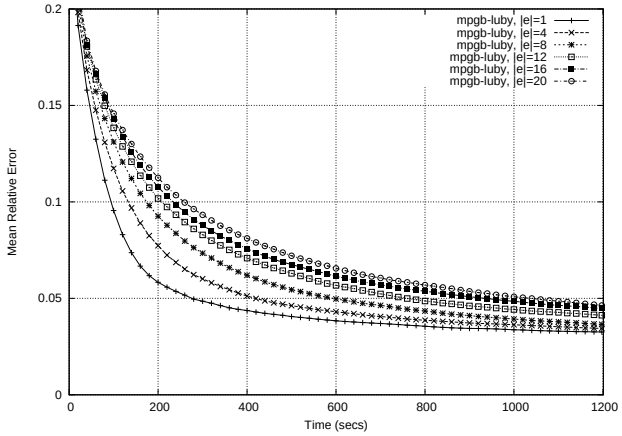

Figure 3: The mean relative error for *mpgb* on the makespan JSPs as the size of the elite set is varied.

Figures 5 and 6 demonstrate substantially different results. Regardless of the MPCS method used, the larger elite set size results in lower mean relative error.

Further Investigations The differences between makespan and tardy JSPs demand further experimentation to understand whether the differences due to artifacts of the experimental design (e.g., the different fail sequences) or to some inherently different behaviour of the algorithms on the two different problem types.

To control for the fail sequence, we repeated the experiment using the Poly fail sequence for the makespan JSPs and the Luby fail sequence for the tardy JSPs. The results (not shown) indicate that the fail sequence is not responsible for the discrepancy: regardless of fail sequence for tardy JSPs, a larger elite size reduces the MRE while for the makespan

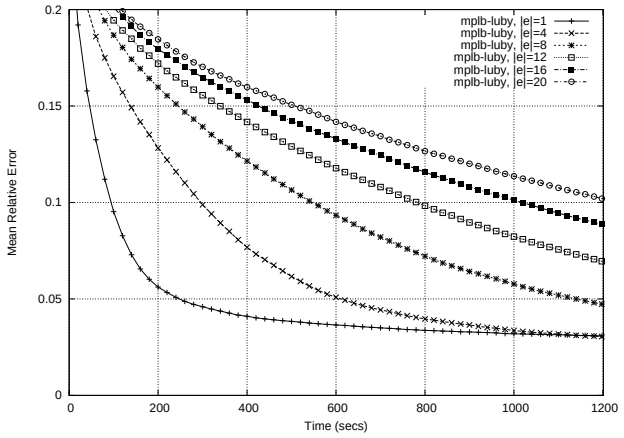

Figure 4: The mean relative error for *mplb* on the makespan JSPs as the size of the elite set is varied.

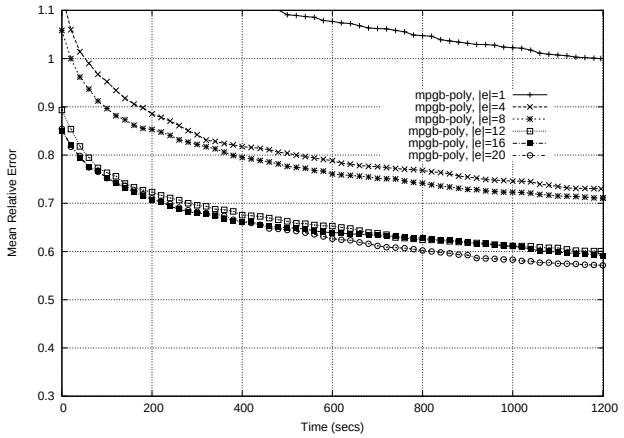

Figure 5: The mean relative error for *mpgb* on the tardy JSPs as the size of the elite set is varied.

JSPs, a larger elite size increases the MREs.

Unlike the results with the makespan JSPs, Figures 5 and 6 both clearly show that the algorithms with higher elite sizes *start* from better solutions. This is due to the fact that the initialization of the elite set is an independent set of $|e|$ runs with a fixed backtrack limit. Given the high variance observed for the tardy problems, repeated random runs at the beginning of the search apparently lead to very fast improvements in the best solution found. Both figures show that the mean starting solution for $|e| = 20$ is better than the mean final solution for $|e| = 1$. To investigate the impact of the starting solution quality, we fix the number of starting runs to be independent of the elite size and initialize the elite set to the best $|e|$ solutions found during the initialization phase. Figures 7 and 8 show the results for mpgb-poly and mplb-poly respectively when 20 initialization runs are done with each elite size (the asterisk in the label denotes the algorithm difference with the previous experiment). Comparing the results with Figures 5 and 6 shows a substantial change. We now see a relationship that is much closer to that observed

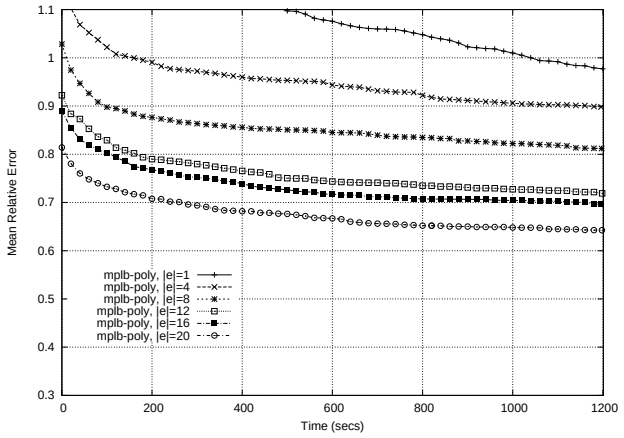

Figure 6: The mean relative error for *mplb* on the tardy JSPs as the size of the elite set is varied.

with the makespan JSPs. A smaller elite size tends to correlate with better performance. The relationship, however, is not as clear as with the makespan JSPs. For mpgb, $|e| = 16^*$ results in the worst performance and $|e| = 4^*$ results in the best. For mplb, $|e| = 1^*$ performs best.

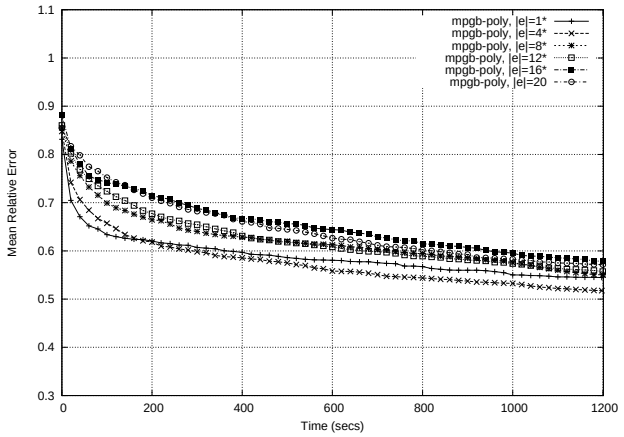

Figure 7: The results for mpgb for different elite sizes each with 20 independent initialization runs. The asterisk indicates the difference from standard MPCS framework.

Given the importance of diversity for elite solution sets within the metaheuristic literature, the performance of the algorithms with an elite size of 1 is very surprising and seems to contradict our original intuitions and motivations for the MPCS approach. We will return to this point in the discussion.

Experiment 3: The Probability of Search from an Empty Solution

Experiment 3 varies p, the probability of performing a search from an empty solution as opposed to a search from an elite solution. The values used are: 0, 0.25, 0.5, 0.75, 1.

Figure 9 shows that for mpgb p-values of 0, 0.25, and 0.5

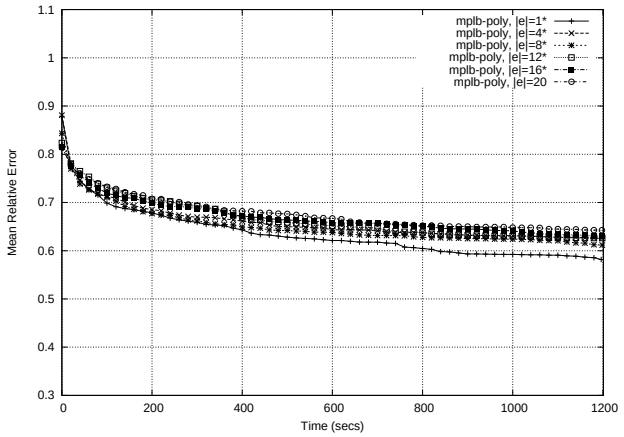

Figure 8: The results for mplb for different elite sizes each with 20 independent initialization runs. The asterisk indicates the difference from standard MPCS framework.

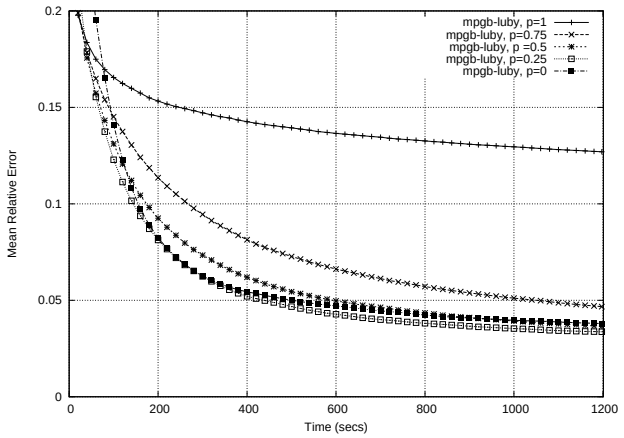

Figure 9: The mean relative error for varying p-values for *mpgb* on makespan JSPs.

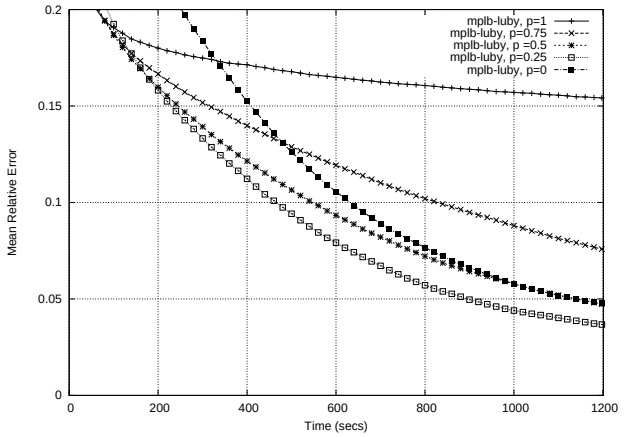

Figure 10: The mean relative error for varying p-values for *mplb* on the makespan JSPs.

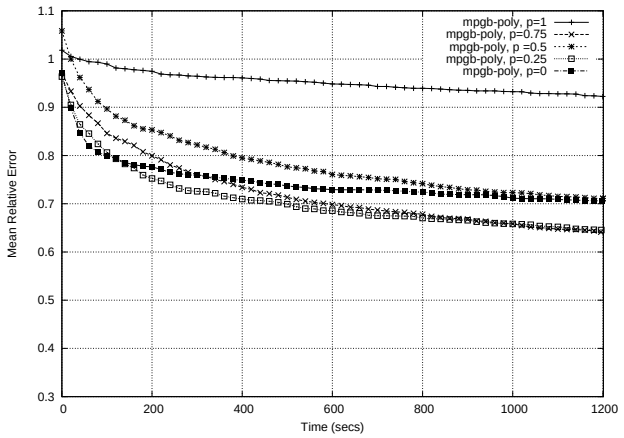

Figure 11: MRE for varying p-values for *mpgb* on the tardy JSPs.

result in comparable performance with $p = 0.25$ resulting in slightly better performance. When the mplb variant is used (Figure 10), the results are more widely spread with $p = 0.25$ again resulting in the lowest MRE. In both cases, setting $p = 1$, which is equivalent to random restart, results in the worst performance.

Turning to the tardy JSPs (Figures 11 and 12), we see that $p = 1$ results in the worst performance and the best performance comes from $p \in \{0, 0.25, 0.75\}$. For both mpgb and mplb on the tardy JSPs, $p = 0.25$ results in the best performance, however, its results are only slightly superior to the other top-performing p-values.

Experiment 4: Initialization Fail Bound

This experiment evaluates if it is worthwhile to spend more effort performing standard chronological backtracking in the initialization phase to find good starting solutions. We have already observed in Experiment 2 that, at least for the tardy JSPs, the starting solution can have a significant impact on the algorithm performance.

In our different experimental conditions, the fail bound in each run in the initialization phase is set to 1, 10, 100, 1000, and 10000. Rather than investigating the trade-off between longer initialization searches or a longer main phase, we are interested in the absolute behaviour of the MPCS algorithms in terms of their starting solution quality. Therefore, the 20-minute CPU time starts from *after the elite pool is initialized*. This means that the experiment only tests the raw impact of starting with better solutions, it does not test the trade-off between spending time finding better initial elite solutions and doing MPCS.

Table 2 shows the MRE of the starting solution (start) and the best solution (best). Increasing the fail bound in the initialization phase does lead to higher quality starting solutions. However, for the makespan problems, the best solutions found are surprisingly uniform. It appears that even spending five orders of magnitude more effort in the initialization phase via longer chronological searches does not result in any better overall performance.

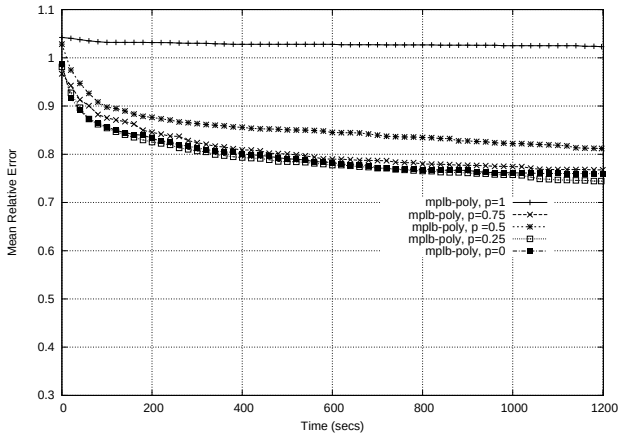

Figure 12: MRE for varying p-values for *mplb* on the tardy JSPs.

Makespan JSPs				
Initial	mpgb		mplb	
Limit	start	best	start	best
1	0.469	0.037	0.469	0.048
10	0.465	0.037	0.465	0.047
100	0.421	0.037	0.423	0.047
1000	0.354	0.038	0.359	0.047
10000	0.307	0.039	0.305	0.047
Tardy JSPs				
1	1.766	0.671	1.756	0.824
10	1.800	0.679	1.783	0.783
100	1.676	0.669	1.718	0.813
1000	1.429	0.664	1.563	0.804
10000	1.329	0.659	1.312	0.747

Table 2: The mean relative error for both multi-point algorithms with differing limits on the initialization for the makespan and tardy JSPs.

Turning to the tardy JSPs, we see a greater impact from the higher initialization fail bounds both in terms of the increase in the quality of the starting solution and in the increase in quality of the final solution. However, given the results of our experiments in varying the elite size and, in particular, the importance of the number of runs in the initialization phase as shown in Figure 7, it appears that the initialization fail bound has much less of an impact. It seems clear that a better starting solution *can* lead to better overall performance. However, it also seems clear that attempting to achieve a better starting solution by increasing the initialization fail bound does not succeed.

Experiment 5: Backtracking Method

In this experiment, we examine the way in which individual searches are done. In addition to chronological backtracking, as used in the above experiments, we use LDS.

Figure 13 shows that for the makespan JSPs, chronological backtracking out-performs LDS when run inside the multi-point framework. This performance difference is observed for both mpgb and mplb.

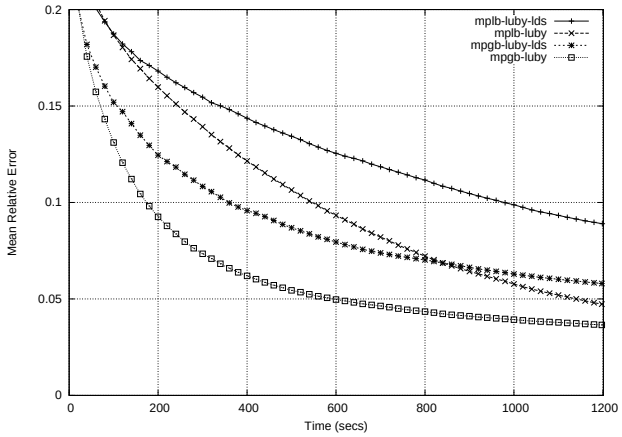

Figure 13: The mean relative error for mpgb and mplb using the Luby fail limit and either chronological backtracking or LDS on the makespan JSPs.

For the tardy JSPs (Figure 14), the results differ from the makespan JSP results: the multi-point variants that use LDS perform better than those using chronological backtracking.

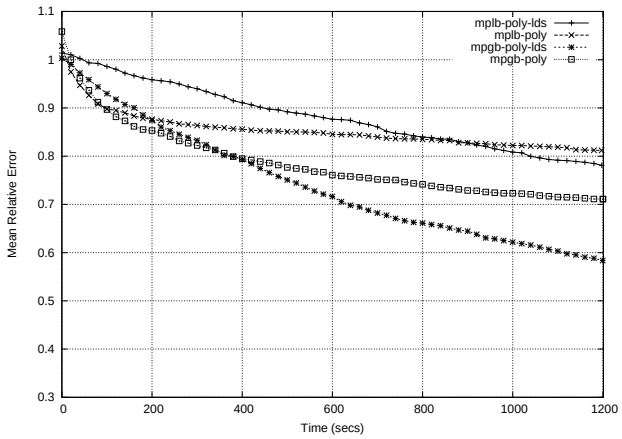

Figure 14: The mean relative error for mpgb and mplb using the Luby fail limit and either chronological backtracking or LDS on the tardy JSPs.

Further Investigations As in our experiment on elite set size (Experiment 2), this discrepancy between the makespan and tardy JSPs demands more experimentation. Table 3 displays the final mean relative error achieved by mpgb and mplb using both Luby and Poly fail sequences on both problem types. The bold entries in each cell indicate the lowest MRE. The table shows that the results in Figures 13 and 14 tend to hold with different fail sequences. On the makespan problems, chronological backtracking performs best in all conditions. On the tardy problems, LDS performs best on all condition but one: mplb-luby.

While the possibility remains that the results in Table 3 stem from some artifact of the experimental design, we are

Problem Type	Backtrack Method	mpgb		mplb	
		Luby	Poly	Luby	Poly
Makespan	Chron	**0.037**	**0.045**	**0.047**	**0.056**
	LDS	0.058	0.081	0.089	0.119
Tardy	Chron	0.714	0.711	**0.838**	0.812
	LDS	**0.696**	**0.583**	0.858	**0.781**

Table 3: The mean relative error achieved after 1200 CPU seconds for each multi-point variant, each fail sequence, and both problem types. The bold entries indicate the lowest MRE in each cell.

unable at this point to find any obvious candidates. We return to this point in the discussion below.

Experiment 6: Comparison with Other Techniques

The MPCS framework generalizes randomized restart: setting $p = 1$ results in an algorithm that always starts from an empty solution. If we further set $|e| = 1$, then the algorithm is exactly the same as randomized restart. When the bound on the number of fails is reached, search restarts from an empty solution and always sets the bound on the problem cost to one less than the best solution found so far.

Figure 15 displays the results of the best parameter combination identified in the previous experiments (see Table 4 below for a summary) and the parameter combination that corresponds to randomized restart. The two algorithms have similar parameter values but, as can be seen from the graph, vary significantly in performance: revisiting high quality solutions has a substantial positive impact. A standard chronological backtracking algorithm was also tested using a non-randomized version of the texture-based heuristics and the same propagators as the other algorithms. The MRE after 1200 CPU seconds was 0.202. We do not show the results on the graph as it would require re-scaling. These results are consistent with previously published results on a different set of makespan JSPs (Beck 2005a).

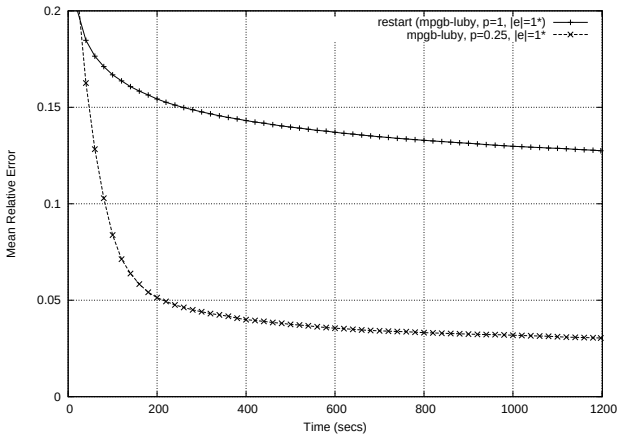

Figure 15: The mean relative error of the best multi-point parameters found in our experiments compared to randomized restart (*restart*) on makespan JSPs.

Figure 16 compares the best MPCS parameters found above to restart on the tardy JSPs. As with the makespan JSPs, a standard chronological backtracking algorithm (using the non-randomized version of the same heuristic and the same propagation techniques) performs very poorly achieving an MRE of 1.22 after 1200 CPU seconds. Four algorithms are shown in Figure 16: restart and restart-lds are randomized restart with chronological and limited discrepancy search respectively (the parameters of the MPCS framework are shown in the figure); mpgb-poly, $p = 0.25$, $|e| = 4^*$ is formed by choosing the best individual parameters from the above experiments; and mpgb-poly, $p = 0.25$, $|e| = 1^*$ is identical except that the elite size is 1 instead of 4. We experiment with the final variation due to the strong performance of $|e| = 1$ in Figure 11 and because it has only one parameter set differently than restart-lds ($p = 0.25$ instead of $p = 1$) and therefore makes an interesting contrast. The results indicate that randomized restart with chronological backtracking performs worst, followed by restart-lds and then the MPCS version with $|e| = 4$. The MPCS variant with $|e| = 1$ performs substantially better than all other techniques and, in fact, achieves the lowest MRE for the tardy JSPs that we have observed in the experiments in this paper.

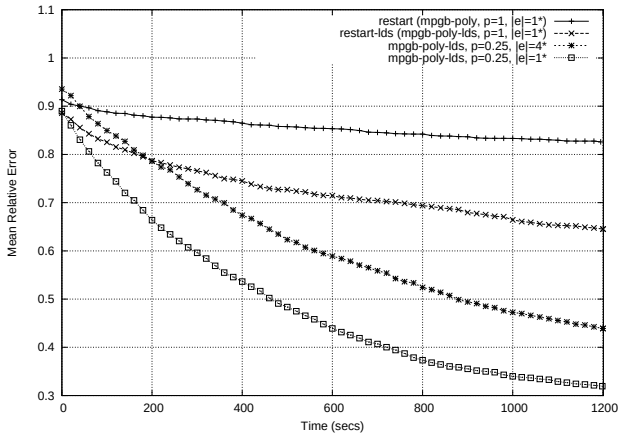

Figure 16: The mean relative error of the best multi-point parameters found in our experiments compared to two randomized restart variations (*restart* and *restart-lds*) and another interesting parameter setting on tardy JSPs.

As further indication of the performance of MPCS techniques, we note that *all* the MPCS parameter settings tested perform better than chronological backtracking and many of them perform better than randomized restart.

Discussion

Table 4 displays the best parameter settings that were found in our experiments. Recall that the asterisk in the $|e|$ column indicates that the number of initialization runs does not depend on the elite set size other than that the number of runs must be equal or greater than $|e|$. In all such algorithms, we fixed the number of initialization runs to 20.

We did not conduct a fully crossed experiment and, unsurprisingly, have shown in Figure 16 that there is an interaction between parameters. Therefore, the values in Table

Problem Type	Fail Seq.	$\|e\|$	p	Init. Fail Bound	Backtrack Method
Makespan	Luby	1^*	0.25	1	chron
Tardy	Poly	1^*	0.25	1	LDS

Table 4: The best parameters for the MPCS algorithms on the makespan and tardy JSP problems.

4 may not represent the optimal settings for these problem instances. However, our main interest in this paper was not to find such settings but rather to begin the development of an understanding of the reasons for the performance of the MPCS techniques based on the performance of the different parameter settings. We now turn to what appear to be the most interesting implications of our experiments.

Intensification vs. Diversification The balance between intensification and diversification is important in the metaheuristic literature (Rochat & Taillard 1995). Intensification suggests searching in the region of good solutions while diversification suggests searching in areas that have not been searched before. Furthermore, one of the important aspects of the metaheuristics based on elite solutions is how the diversity of the elite set is maintained (Watson 2005). Both the results on varying the elite set size and on how the bound is placed on the solution cost, indicate that, for MPCS performance, intensification appears to be more important than diversification. Both a larger elite size and the local bounding technique (i.e., mplb) contribute to a greater diversity. In contrast, a small elite set and the global cost bound intensify search to only allow globally better solutions found in the vicinity of a small number of elite solution.

The results show that an elite size of $|e| = 1$ and global cost bound lead to superior performance. This is surprising as the MPCS techniques were originally motivated to allow good solutions to heuristically guide constructive search and it was expected that elite pool diversity would be an important aspect of the search guidance. However, the global bounding approach results in greater constraint propagation because the upper bound on the cost function is always one less than the best solution found. We speculate that the search space reduction provided by back-propagation from the cost function is large enough to counter-act any heuristic gains from the greater diversity of the local cost bounding approach. This speculation is supported by preliminary experiments on problems with no back-propagation from the cost function which indicate superiority for the local bounding approach (Beck 2005a). Experiments with $|e| = 1$ on such problems are currently underway.

An alternative explanation is that the elite sets are composed of a set of very similar solutions and, therefore, lack diversity. If such is the case, there is no advantage in maintaining a larger set of solutions. This explanation appears unlikely as it would mean that the poor performance of larger elite sizes stem from the overhead of maintaining a larger elite set (see Figure 3). We do not believe that such an overhead is large enough to account for the performance differences. Nonetheless, we are pursuing experiments to measure and manipulate the diversity of the elite set.

Multi-Point or Guided Randomized Restart? While a small elite size increases intensification, the strong performance of $|e| = 1$ on both problem types deserves particular note. Each time an elite solution is revisited with a randomized variable ordering, a different search tree is created. A resource-bounded chronological search means that only the nodes deep in the tree will be visited before the fail limit is reached. However, a different variable ordering results in a different set of nodes that are deep in the tree and close to the elite solution.[3] The strong results of the MPCS algorithms with $|e| = 1$ may be an indication that the mechanism responsible for the strong performance is the sampling of solutions "close" to an elite solution in different search trees rather than in investigating different areas of the search space (Beck 2005b). This speculation suggests that experiments that more directly control the variable ordering heuristic are worth investigating.

Chronological Backtracking vs. LDS The differences between chronological backtracking and LDS, and the strong performance of restart-lds are intriguing. The fact that LDS performs better with the tardy JSPs which have weaker propagation due to the form of the cost function (i.e., a sum of a set of variables rather than the maximum), may indicate a further impact of propagation strength. We expect that with a strongly biased, weak heuristic where incorrect variable assignments occur high in the tree, LDS would outperform chronological backtracking: it would have the opportunity to undo the incorrect assignments while chronological backtracking could not undo such assignments within its resource bound. With that perspective and previous work on LDS, the anomaly in our results would seem to be not the strong performance of LDS on the tardy problems, but the weak performance on the makespan problems. Substantial further work in elucidating the reasons for the performance of MPCS is needed to answer this question.

Future Work

The experiments conducted in this paper have demonstrated that multi-point constructive search techniques can result in strong performance on job shop scheduling problems. There are three chief areas for subsequent work:

1. We lack a real understanding of the behaviour of the MPCS techniques. Inspired by empirical work in the metaheuristic literature (Watson 2003), we are pursuing research to establish measurements of "distance" between solutions in a tree search, to understand the changes in distance with different variable orderings, and to investigate any correlation between distance and solution quality. The impact of diversity on MPCS performance and, in particular, the impact of more intelligent methods for maintaining the diversity of the elite set are interesting questions whose answer depends on understanding the algorithm behaviour.

[3] Similar reasoning applies to the use of LDS.

2. MPCS techniques are not limited to job shop scheduling problems. We are undertaking studies to apply MPCS to other scheduling problems (Cesta, Oddi, & Smith 2002), to constraint satisfaction problems (Beck 2005a), and to search problems in other areas including planning.

3. The MPCS framework can be extended in a number of directions: metaheuristic techniques such as path relinking (Glover, Laguna, & Marti 2004) have elegant counterparts in multi-point constructive search; adaptive, hybrid techniques that share single solutions (Carchrae & Beck 2005) are easily generalizable to share a set of solutions; and machine learning-based approaches for dynamic parameter learning (Horvitz *et al.* 2001) would appear very useful to the MPCS framework.

Conclusion

We presented the first systematic study of multi-point constructive search techniques. Using two related scheduling problems (job shop scheduling with makespan minimization and job shop scheduling with weighted tardiness minimization), we varied the MPCS parameter settings to control the upper bound on the cost function, the fail sequence, the size of the elite set, the probability of searching from an empty solution, the effort spent in the initialization phase, and the style of backtracking search used. Overall our results indicate that the original motivation for multi-point constructive search (i.e., the ability to heuristically sample diverse areas of the search space) may not be responsible for the strong performance exhibited. We showed that the best parameter values found in our experiments result in algorithms that substantially out-perform existing constructive search methods. Furthermore, comparison of existing constructive search techniques to all parameter combinations tested shows that in many cases, MPCS with poor parameter choices still out-performs existing approaches.

Acknowledgments This research was supported in part by the Natural Sciences and Engineering Research Council and ILOG, S.A. Thanks to Jean-Paul Watson and Tom Carchrae for comments on early versions of the paper.

References

Beck, J. C., and 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. 2005a. Multi-point constructive search. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP05)*.

Beck, J. C. 2005b. Multi-point constructive search: Extended remix. In *Proceedings of the CP2005 Workshop on Local Search Techniques for Constraint Satisfaction*, 17–31.

Carchrae, T., and Beck, J. C. 2005. Applying machine learning to low knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *Journal of Heuristics* 8(1):109–136.

Glover, F.; Laguna, M.; and Marti, R. 2004. Scatter search and path relinking: advances and applications. In Onwubolu, G., and Babu, B., eds., *New Optimization Techniques in Engineering*. Springer.

Gomes, C. P.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 431–437.

Harvey, W. D. 1995. *Nonsystematic backtracking search*. Ph.D. Dissertation, Department of Computer Science, Stanford University.

Horvitz, E.; Ruan, Y.; Gomes, C.; Kautz, H.; Selman, B.; and Chickering, M. 2001. A bayesian approach to tacking hard computational problems. In *Proceedings of the Seventeenth Conference on uncertainty and Artificial Intelligence (UAI-2001)*, 235–244.

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.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47:173–180.

Nowicki, E., and Smutnicki, C. 2005. An advanced tabu algorithm for the job shop problem. *Journal of Scheduling* 8:145–159.

Nuijten, W. P. M. 1994. *Time and resource constrained scheduling: a constraint satisfaction approach*. Ph.D. Dissertation, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Rochat, Y., and Taillard, E. D. 1995. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics* 1:147–167.

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

Watson, J.-P.; Barbulescu, L.; Whitley, L. D.; and 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. 2003. *Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem*. Ph.D. Dissertation, Dept. of Computer Science, Colorado State University.

Watson, J.-P. 2005. On metaheuristics "failure modes": A case study in tabu search for job-shop scheduling. In *Proceedings of the Fifth Metaheuristics International Conference*.