Multi-Point Constructive Search: Extended Remix[⋆]

J. Christopher Beck

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

Abstract. Multi-Point Constructive Search maintains a small set of "elite solutions" that are used to heuristically guide constructive search through periodically restarting search from an elite solution. Empirical results indicate that for job shop scheduling optimization problems, multi-point constructive search leads to significantly better solutions for equivalent search effort when compared to chronological backtracking and bounded backtracking with random restart. For satisfaction problems (quasigroup with holes completion), significant reduction in the magnitude of mean search effort (the number of fails and run-time) is also achieved versus chronological backtracking and bounded backtracking with random restart. Two conjectures about the relationship between the clustering of good solutions in a search tree and the performance of multi-point constructive search are made. Preliminary empirical results are consistent with the conjectures, suggesting directions for future work to develop a deeper understanding of the observed performance.

1 Introduction

Metaheuristics such as genetic algorithms, scatter search and path relinking [7], and tabu search with reintensification [14], often maintain a set of sub-optimal solutions that are used to guide search. In the case of scatter search and tabu search with reintensification, a small number (e.g., five to ten) of "elite solutions," typically containing the best solutions that have been encountered so far, are used to define the areas in the search space that appear promising. Search is periodically restarted from the elite solutions to explore these promising areas. In contrast, constructive search follows a heuristic decision-making procedure until a dead-end or (sub-optimal) solution is found and then backtracks to search through alternative decisions. While there are a number of ways in which the alternatives may be explored (e.g.,[17]), such techniques are variations on the way in which a single search tree is explored. We view such search as a "single point" technique: search begins at the single heuristically preferred point in the search tree and gradually moves away. An exception to single point search are techniques that use random restarts [15, 9]. Through periodic restarts, different points in the search space are sampled by a randomized heuristic. Randomized restart techniques do not maintain a set of solutions nor allow an existing set of solutions to impact the parts of the search tree that are subsequently explored.

⋆ This work has received support from ILOG, SA. A shorter version of this paper appears at *CP2005*.

This paper introduces the maintenance of multiple solutions to guide constructive tree search. Given a set of elite solutions, we probabilistically choose to start constructive search either from a random elite solution or from an empty solution. If a good solution is found within some bound of the search effort, it is inserted into the elite set, replacing one of the existing solutions. Our empirical results both on constraint optimization and constraint satisfaction problems demonstrate significantly improved search performance when compared with bounded backtracking with random restart and with standard chronological backtracking. For satisfaction problems, we are able to achieve more than a three-fold reduction in search effort.

The contributions of the paper are the introduction and investigation of multi-point constructive tree search; the introduction of a technique to start constructive search from an existing solution while exploiting existing variable ordering heuristics; the demonstration of significant search performance gains resulting from multi-point search for both constraint optimization and constraint satisfaction problems; and the generation of empirical evidence that is consistent with simple conjectures to explain the multi-point constructive search performance.

Multi-point constructive search is introduced in the following section. Empirical results on the job shop scheduling problem and the quasigroup completion problem are presented in Sections 3.1 and 3.2, respectively. Section 4 presents conjectures to explain the observed behaviour and preliminary empirical evidence supporting the conjectures. Avenues for future work are presented in Section 5.

2 Multi-Point Constructive Search

For clarity, we introduce Multi-Point Constructive Search (MPCS) in the context of constraint optimization before presenting adaptations for constraint satisfaction.

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.

Constructive search from an existing solution and the upper bound on the cost function are discussed in detail below. We first provide more detail on a number of aspects of the algorithm.

- *Elite Solution Initialization* 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

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($\emptyset$, $b$)
7          if $s \neq NIL$ 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 $s \neq NIL$ $s$ is better than $r$ then
14             replace $r$ with $s$
15 return best($e$)
```

empty elite set to ensure that the initial solutions are independently generated and therefore diverse. Elite set diversity is important for metaheuristics; further work is needed to determine its impact on constructive search.

– *Finding a Solution From Scratch* A solution is found from scratch (i.e., line 6) using any standard constructive search with a randomized heuristic and a bound on the number of fails. It is possible that no solution is found within the fail bound.
– *Bounding the Search* The effort spent on each individual search is bounded by an evolving 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 associate a different fail bound, initialized to 32, with each elite solution. Whenever search from an elite solution does not find a better solution, the fail bound for that elite solution is doubled. When an elite solution is replaced (lines 8 and 14), the bound associated with the new elite solution is set to 32. When searching from an empty solution, we use the mean fail bound of the elite solutions and do not increase any fail bounds in the case of failure to find a better solution. The choice of 32 as a starting fail bound was arbitrary, though based on the intuition that the minimum should allow for a small search effort.

2.1 Starting Constructive Search from an Elite Solution

Our goal is to perform some form of backtracking search away from the starting or *reference* solution. To do this, we create a search tree using any variable ordering heuristic and by specifying that the value assigned to a variable is the one in the reference solution provided it is still in the domain of the variable.

A search tree is created by asserting a series of choice points of the form: $\langle V_i = x_i\rangle \vee \langle V_i \neq x_i\rangle$. 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 we use any variable ordering heuristic to choose the next variable to assign, V_i. 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. Our variable ordering heuristic has complete freedom to choose a variable, V_i, to be assigned. If $x \in dom(V_i)$, where $\langle V_i = x\rangle \in s$, the choice point is made with $x_i = x$. Otherwise, if $x \notin dom(V_i)$, any value ordering heuristic is used to choose $z \in dom(V_i)$ and the choice point is asserted with $x_i = z$. Because our criterion for assigning the value that is in the reference solution is $\langle V_i = x\rangle \in s$, the case where *no* value is assigned to V_i in s (i.e., when s is a partial solution and $m < n$) is covered.

Because we place an upper bound on the cost function (line 10) or, more generally, allow the addition of constraints such as nogoods to the model, the reference solution is not necessarily still a valid solution. To take a simple example, if the reference solution had 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 close to the reference solution from where we will backtrack. This is why the possibility that $x \notin dom(V_i)$ must be taken into account.

This technique for starting constructive search from a reference solution is quite general. Existing, high performance variable ordering heuristics can be exploited and, as noted, we make no assumptions about changes to the constraint model that may have been made after the reference solution was originally found. To our knowledge this is the first work that has sought to start constructive search from an existing solution.

2.2 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 solutions 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 three approaches to setting the upper bound:

1. *Global bound*: Always set the upper bound on the search cost to $c^* - 1$.
2. *Local bound*: 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.

3. *Adaptive*: Whenever a new global best solution is found, use the global bound policy for a fixed number of searches and then revert to using the local bound policy. Whenever a new best solution is found, the counter for the number of searches is reset. This means that if new best solutions are consistently found, the global bound policy will continue to be used. In all our experiments, we set the fixed number of searches to be $|e|$, the size of the elite set. This size was chosen because it seemed reasonable that the use of the global bound should be a function of the number of elite solutions. Note however that if $p = 0.5$, as it does in all our experiments, we expect that only half of the elite solutions will be a starting point for a global bound search. No experiments have been performed yet with different parameter values.

For problems with strong propagation from the cost function, we expect the global bound policy to out-perform the local bound policy: when a new best solution results in substantial reduction in the search space, it will pay off to take advantage of it. In contrast, in problems with minimal or no back-propagation, we expect the heuristic guidance from the local bound policy will result in stronger performance. The adaptive approach is an attempt to combine the expected strengths of these two policies.

2.3 Applying MPCS to Constraint Satisfaction Problems

The core of MPCS is the maintenance and heuristic use of sub-optimal solutions. To adapt the approach to a satisfaction context, therefore, we must decide how to compare non-satisfying "solutions." A common approach is to allow inconsistent variable assignments and use the number of violated constraints as an objective to be minimized. The drawback to this approach is that much of the propagation power of standard constraint solvers is lost when constraints are relaxable. We, therefore, choose another approach by rating *partial* solutions by the number of unassigned variables. When a dead-end is encountered, the number of variables that have not been assigned are counted. Partial solutions with fewer unassigned variables are assumed to be better. We make no effort to search after a dead-end is encountered to try to determine if any of the currently unassigned variables could be assigned without creating further constraint violations.

In addition to enabling the use of standard constraint models and solvers, allowing partial solutions has a number of further implications. First, our assumption that it is easy to find sub-optimal solutions during the elite solution initialization remains true: any partial solution is a sub-optimal solution. Second, because our elite solutions may be partial solutions, it is necessary, as noted above, that our procedure for searching from elite solutions is general enough to handle partial solutions. Third, since the cost function is the number of unassigned variables, reducing the bound on the cost function is not constraining and has no impact on the constraint-based search. We expect this to affect the relative performance of the global and local bound policies. Finally, on a technical level, comparing partial solutions means that rather than evaluating the sub-optimal solutions found by the solver, we must evaluate the dead-ends. In ILOG Solver, this is implemented using the trace mechanism to "catch" each fail.

3 Empirical Studies

In this section, we present experiments that apply MPCS to a constraint optimization and a constraint satisfaction problem: respectively, job shop scheduling and quasigroup-with-holes completion.

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 pre-defined duration, d_i, and a resource, r_i, that it must have unique use of during its duration. There are also 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. The decision variant of the JSP asks if there is a solution for makespan D or less. It is NP-complete [6]. We are interested in the optimization version of the problem: given a maximum CPU time, return the best solution found.

Experimental Details Ten 20×20 random JSPs were generated using an existing generator [21]. The routings of the jobs through the machines are randomly generated and the activity durations are independently and randomly drawn from [1, 99].

For all algorithms, texture-based heuristics [2] are used to identify a resource and time point with maximum competition 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 in the solution is asserted. The standard constraint propagation techniques for scheduling [15, 12, 13] are also used for all algorithms.

We experiment with five algorithms:

- Standard chronological backtracking denoted *chron*.
- Bounded backtracking with restart (*bbt*) following the same fail-bound sequence used for the multi-point techniques. The algorithm is Algorithm 1 with a single difference: line 12 is replaced by a copy of line 6. Every search is from an empty solution, but the fail bound evolves as in multi-point search.
- Multi-Point Constructive Search. Three variations of MPCS are used corresponding to the different ways to set the upper bound on the cost function: multi-point with global bound, *mpgb*; multi-point with local bound, *mplb*; and multi-point with adaptive bound, *mp-adapt*. We set $p = 0.5$, meaning that the computational effort is evenly split between searching from an empty solution and searching from an elite solution. In the metaheuristic literature, the size of the elite solution set is small, typically between five and ten. We have followed this convention in all our experiments by setting $|e| = 8$. The elite solutions are initially populated by independent runs, limited to 1000 fails each, of a randomized algorithm that produces semi-active schedules. No effort was made to tune these parameters: they were chosen based on values often used in the metaheuristic community and intuition.

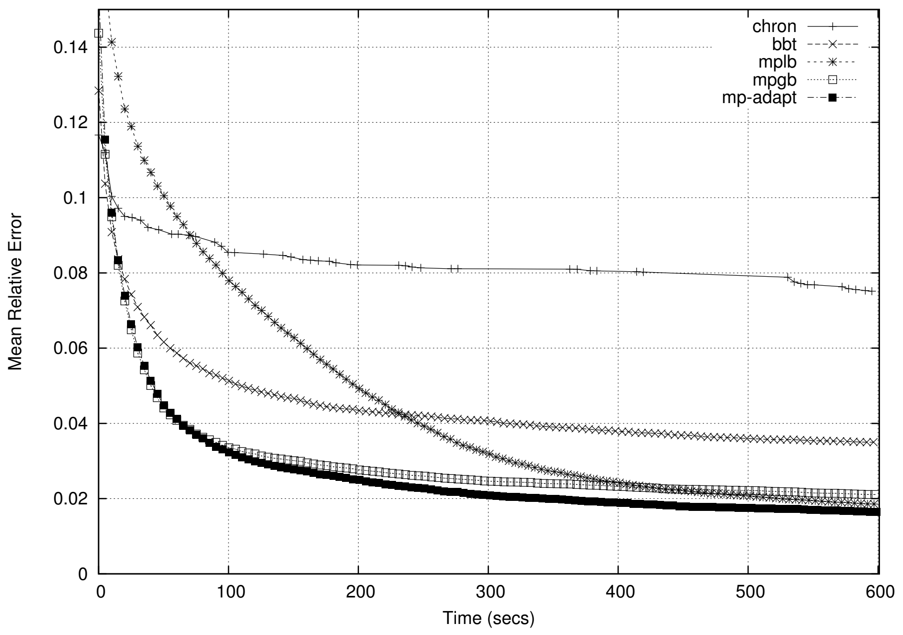

Fig. 1. Mean relative error (MRE) relative to the best known solutions for each algorithm over ten independent runs of ten problem instances.

As the heuristic is randomized, all algorithms are run ten times with aggregate results presented as described below. A time limit of 600 CPU seconds is given for each run: algorithms report whenever they have found a new best solution allowing the creation of normalized run-time graphs. All algorithms are implemented in ILOG Scheduler 6.0 and run on a 2.8GHz Pentium 4 with 512Mb RAM running Fedora Core 2.

It should be noted that the only differences between *bbt* and the MPCS variations is the maintenance and use of the elite solutions. In particular, the same heuristics, propagation, and fail-bound sequence are used across these algorithms.

Results For each algorithm run we calculated the error at different time points relative to the best known solution for the problem instance. The mean relative error (MRE) is the arithmetic mean of the relative error over each run of each problem instance:

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

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.

Figure 1 demonstrates that multi-point search is a significant improvement in terms of finding higher quality solutions over both chronological backtracking and bounded backtracking with random restart. Statistical analysis[1] is performed for time points $t \in \{100, 200, \ldots, 600\}$. The difference between *bbt* and *mpgb* and between *bbt* and *mp-adapt* is statistically significant at all time points. The *bbt* algorithm performs significantly better than *mplb* at $t = 100$ but significantly worse for $t \geq 300$. Turning to the MPCS variations, *mplb* is significantly worse than *mp-adapt* for $t \leq 400$ and significantly worse than *mpgb* at $t \leq 300$. The *mp-adapt* algorithm is significantly better than *mpgb* at all $t \geq 300$.

3.2 The Quasigroup-with-Holes Completion Problem

An $n \times n$ quasigroup-with-holes (QWH) is a matrix where each row and each column is required to be a permutation of the integers $1, ..., n$ and where some of the matrix elements are filled in and others are empty. Finding a complete quasigroup requires that all the empty cells ("holes") are filled with consistent values. The problem is NP-complete and bounded backtracking with randomized restart has been shown to be a particularly strong performer on QWH problems [11].

Experimental Details For this experiment, we generated 100 balanced, order-30 QWH problems (i.e,. $n = 30$) using a generator that guarantees the satisfiability of each instance [1]. Ten sets of problem instances are generated with different numbers of holes, $m = \{315, 320, \ldots, 360\}$. These values were chosen to span the difficulty peak identified in the literature. For each value of m, ten problem instances are generated for a total of 100 instances. Each algorithm was run ten times on each problem instance with a limit on each run of 2,000,000 fails.

The same search framework as for the JSP was used, implemented in ILOG Solver 6.0 on the same machine. The only difference in parameters were for the MPCS variations where the limit to initialize each elite solution was set to 100 fails. All other parameters (i.e., elite set size, fail-bound sequence, etc.) are identical to that used above and no tuning was done. The search heuristics and constraint propagation techniques obviously differ from those used in the JSP. The variable ordering heuristic randomly chooses a variable with minimum domain size while the value ordering is random. When starting search from an elite solution, a variable is chosen by the variable ordering heuristic and then the value that it is assigned to in the reference solution is assigned if possible. If the value in the solution is not in the domain of the selected variable the value ordering heuristic is used to randomly select a value. Global all-different constraints with extended propagation [19] are placed on each row and column.

Results Each problem instance is attempted ten times by each algorithm. In Figure 2 and 3, we present the mean number of fails and mean run-time required for each subset and algorithm: each point is the mean over ten independent runs of ten problem instances. The MPCS variants perform very well, with *mplb* and *mp-adapt* incurring over three times fewer mean fails and lower mean run-time on the hardest problems

[1] All statistical results are measured using a randomized paired-t test [5] and a significance level of $p \leq 0.005$.

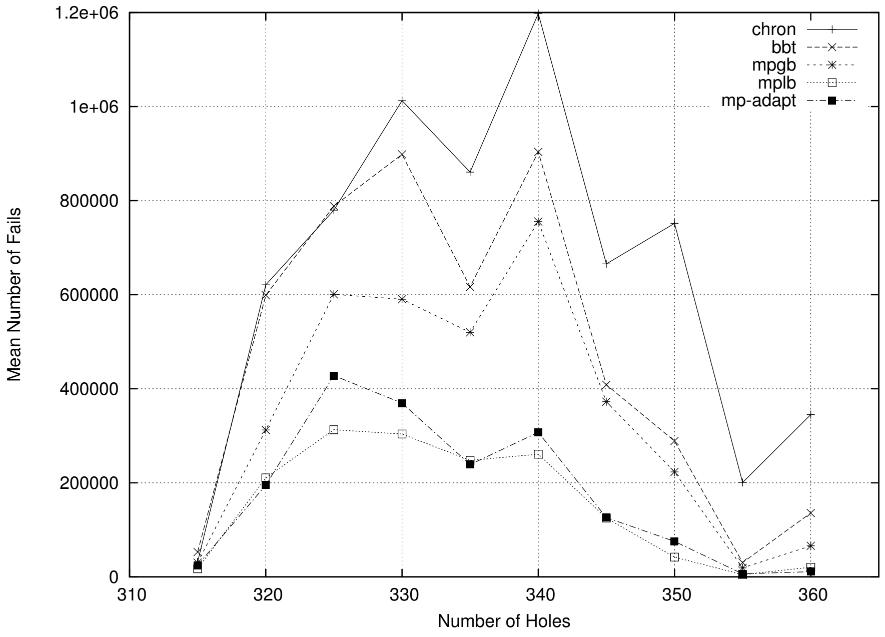

Fig. 2. Mean number of fails to solve order-30 problems in each subset. Each point on the graph represents the mean of ten independent runs of the algorithm on ten problem instances.

compared to *bbt*. In fact, *mplb* and *mp-adapt* are significantly better than all other techniques for $m \geq 325$. In particular, the expected impact of the lack of propagation from the cost bound is observed: unlike the JSP results, *mplb* now out-performs *mpgb* with *mp-adapt* being slightly worse. The *mpgb* algorithm is significantly better than *bbt* only on the easier problems ($m \in \{320, 325, 330, 360\}$). All these statistical results hold for both the mean number of fails and the mean run-time. The *bbt* algorithm is significantly better than *chron* in terms of the mean fails at $m \geq 335$ but in terms of the mean run-time only at $m \geq 350$.

Table 1 presents the percentage of runs for each problem set and algorithm for which a solution was found (recall that all problem instances are soluble). These numbers mirror the mean fails reported in Figure 2 with *mplb* and *mp-adapt* finding solutions on almost all of the runs. The fact that not all runs resulted in a solution means that the results in Figures 2 and 3 are lower bounds on the true mean results.

4 Discussion

The empirical results demonstrate the MPCS performs very well on both constraint optimization and constraint satisfaction problems. The central open question is an explanation of the strong performance. There remains significant empirical work to be done to develop such an explanation but we can make two related conjectures about

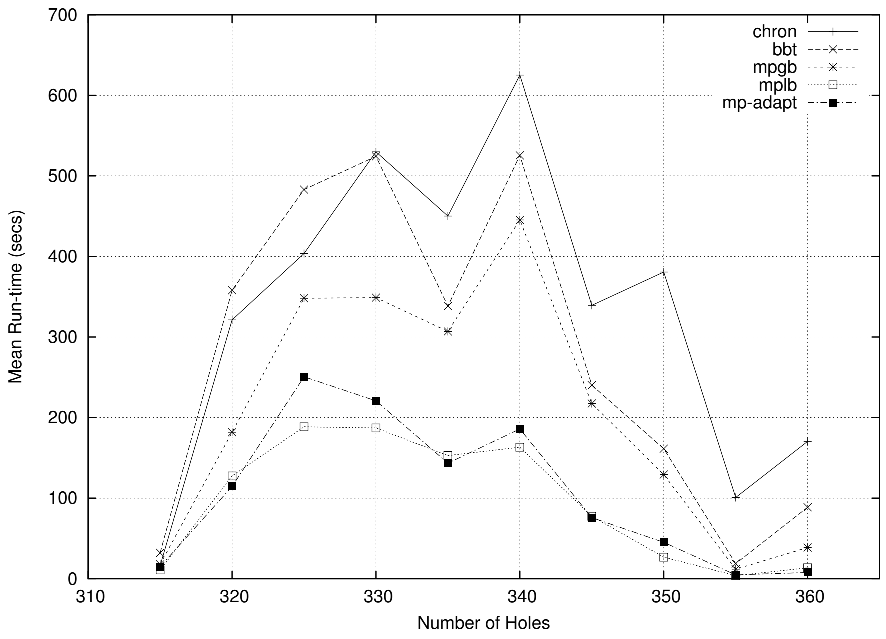

Fig. 3. Mean run-time to solve order-30 problems in each subset. Each point on the graph represents the mean of 100 values: ten independent runs of the algorithm on ten problem instances.

problem structure that may be necessary for strong performance of MPCS and conduct a preliminary experiment to seek support for these conjectures.

Conjecture 1: Solutions Cluster Given a search tree generated by chronological backtracking, we conjecture that good solutions tend to cluster together, meaning that few backtracks and heuristic decisions are needed to move between some high quality solutions. More formally, we conjecture that the cost difference between two solutions is correlated with "tree distance:" the distance between the solutions in the search tree, where distance might be measured, for example, by the number heuristic decisions needed to move from one solution to the other in the tree. We have no direct evidence for such clustering. However, clustering with a different definition of distance, has been reported in the JSP [22] and for SAT problems [20]. Furthermore, such clustering appears to be consistent with, and therefore may have a connection with, the heavy-tailed phenomenon [10]. Finally, we can anecdotally report that during the execution of chronological search for JSPs, it is not uncommon to observe long periods where no new better solutions are found followed by a short burst where a number of such solutions are found in a few seconds.

Conjecture 2: Solution Clusters Change The tree distance between a given pair of solutions changes if the variable order changes. More formally, the tree distances between a given pair of solutions in different search trees are not highly correlated. Toy examples can easily be generated where two sibling solutions in one variable order are

Algorithm	% of Runs Successful											
	315	320	325	330	335	340	345	350	355	360	All	Hardest Sets
chron	**100**	82	76	66	68	58	78	78	97	90	79	71
bbt	**100**	84	76	71	85	71	90	99	**100**	**100**	88	82
mpgb	**100**	96	88	87	83	78	92	99	**100**	**100**	93	88
mplb	**100**	**98**	**93**	**94**	94	**95**	**98**	**100**	**100**	**100**	**97**	**96**
mp-adapt	**100**	97	89	90	**98**	**95**	**98**	**100**	**100**	**100**	**97**	95

Table 1. The percentage of runs for each subset for which a solution was found by each algorithm. Also displayed is the percentages for all problems and for the 600 problems in the hardest problem sets: 325-350. The highest percentage in each column is shown in bold.

far apart for the reverse order. It seems reasonable that such a phenomenon may exist in practice, but again, we have no direct empirical evidence.

The combination of these conjectures leads to a possible explanation of the behaviour of MPCS: solution clustering implies that search in the region of good solutions is likely to find other good solutions; changing clusters implies that the randomized variable ordering makes it very likely repeated searches from a given elite solution will encounter different solutions. Under this interpretation, bounded backtracking with random restart may exploit Conjecture 1: when searching in an area devoid of good solutions (i.e., an anti-cluster), search is restarted because there is a reasonable probability that the new search will find a cluster. Bounded backtracking does not, however, exploit Conjecture 2 because good solutions are never purposely revisited. If this is true, then eliminating the (conjectured) changes in solution clusters should eliminate the advantage that MPCS has over bounded backtracking. The simplest way to remove the possibility of changing clusters is to remove the variability in variable ordering. Therefore, we make the following hypothesis:

Hypothesis With a static variable ordering, MPCS will perform no better than bounded backtracking with restart.

To test this hypothesis, we use lexicographic variable ordering for the QWH problems. Note that even with a lexicographic variable ordering, searching from an elite solution is unlikely to visit the same search states. The randomized value ordering means that once the elite solution has been revisited and backtracking returns to a node, by definition, the value assignment in the elite solution has already been explored. A random consistent value is then chosen leading to a different order of exploration of the search tree.

Aside from the lexicographic variable ordering heuristic, all parameters, the value ordering, the fail-bound sequence, and the maximum fail bound (i.e., 2,000,000) remain as above. We present the mean number of fails and the mean run-time (Figures 4 and 5, respectively) to solve 100 order-27 QWH problems. As above there are ten problems for each number of holes and we solve each problem ten times with each algorithm. The *mplb-lex* algorithm failed to find a solution in 24 of the 1000 runs and *bounded-bt-lex* similarly failed in 31 of the 1000 runs. The other algorithms found solutions in all runs.

The results support our hypothesis: *mplb* is statistically significantly better than *bbt* (both in terms of the number of fails and the run-time) for all subsets with $m \geq 268$, while there are *no significant differences* in either measure between *mplb-lex* and *bbt-*

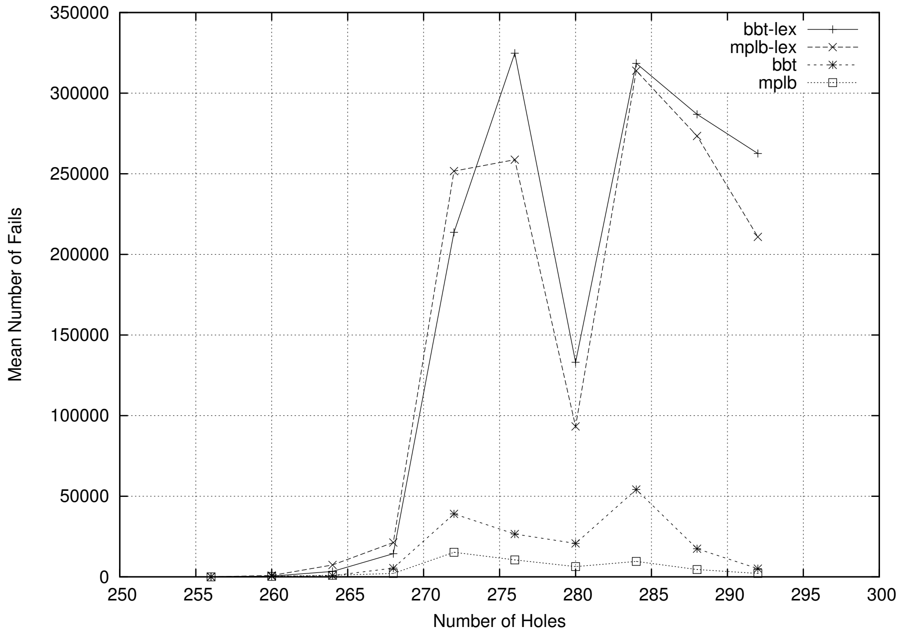

Fig. 4. Mean number of fails to solve order-27 problems in each subset. Each point on the graph represents the mean of ten independent runs of the algorithm on ten problem instances.

lex for any of the subsets (even at a lower confidence level, $p \leq 0.01$ – recall that our standard measure of significance is $p \leq 0.005$).

While our experimental results appear to strongly support our hypothesis, there are a number of weaknesses with this experiment that suggest caution. First, we tested an implication of two conjectures and therefore, on a logical basis, confirmation of our hypothesis does not imply that the premises are true. By the contrapositive, disconfirmation of our hypothesis would place the validity of the conjectures in doubt, so, at least, we can claim that our experiment has not shown the conjectures to be false. Second, as can be observed from the differences in the performance of *bbt* and *bbt-lex*, the static variable order has a large impact on search performance that cannot be attributed to changing solution clusters. Our manipulation was somewhat blunt and observed differences in performance may be caused by factors other than the conjectured change in solution clustering. The effect of these factors may swamp the impact that we intended to evaluate. With these caveats, our experimental results are consistent with our two conjectures about the structure of search trees. More focused empirical work is necessary to further test these promising conjectures and develop a true understanding of the behaviour of multi-point constructive search.

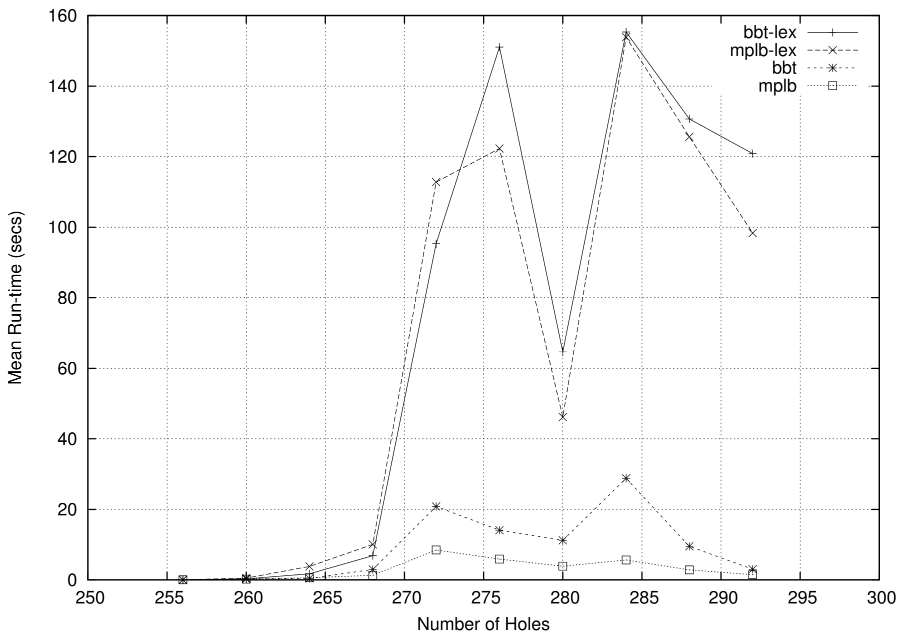

Fig. 5. Mean run-time to solve order-27 problems in each subset. Each point on the graph represents the mean of ten independent runs of the algorithm on ten problem instances.

5 Future Work

The use of sub-optimal solutions to guide constructive search is not limited to the algorithms presented in this paper. Inspired by work in the metaheuristics literature, we are pursuing a number of alternative algorithmic methods, such as path relinking [7], in which sub-optimal solutions can be used to guide constructive search.

One weakness in the current work is that no investigation of the different parameter settings has been done. Experiments are underway to evaluate varying the proportion of time spent searching from elite solutions (i.e., the p value), different sizes for the elite set, different fail-bound sequences, and using limited discrepancy search rather than chronological search. We hope to be able to develop a model of the behaviour of the MPCS algorithms and an understanding of the characteristics of the problems and search spaces where such algorithms perform well.

There are also a number of areas of future work in investigation of combinations of MPCS with recent advances and in applying it to other constructive search formalisms. In particular, the following directions seem promising:

- Recent work on adaptive variable ordering heuristics [18] notes that restarting results in more informed heuristics suggesting a good fit with the multi-point search.
- Large neighbourhood search [16, 8] and other iterative methods [4] already perform repeated searches. A multi-point extension of these techniques is straightforward.

- Hybrid techniques [3] that mix different constructive search algorithms and metaheuristics can easily be extended to maintain elite solutions.
- Techniques that adaptively allocate computational resources to different search algorithms have shown some impressive results [3]. Such techniques may be useful in adaptively setting the algorithm parameters such as the p value.
- Multi-point search techniques are naturally decomposable and therefore might be exploitable in a multi-processor architecture.
- Constructive tree search is used in other search formalisms such as SAT solving and mixed-integer programming. It would be interesting to see if the techniques introduced in this paper can also be exploited in these areas.

6 Conclusion

This paper introduces multi-point constructive search. The search technique builds on existing CP infrastructure, notably search heuristics, constraint propagation, and modeling techniques and operates by maintaining a small set of *elite* solutions: high quality solutions that have been encountered during the problem solving. The overall algorithm consists of a series of resource-limited constructive searches either from an empty solution or from an elite solution. Depending on the outcome of the searches, new solutions are inserted into the elite set, replacing existing solutions. Two sets of experiments are conducted and significant performance gains relative to chronological backtracking and bounded backtracking with random restart are observed both on constraint models of optimization problems (job shop scheduling) and satisfaction problems (quasigroup-with-holes completion). Finally, two conjectures are made that relate the performance of multi-point constructive search to the clustering of high quality solutions in a search tree and to the changes in this clustering with different variable orders. Though significant work remains, preliminary experimental evidence is consistent with the conjectures.

The contributions of the paper are:

1. The introduction of multi-point constructive search.
2. The introduction of a technique to restart constructive search from an existing solution that exploits existing variable ordering heuristics.
3. The demonstration of significant search performance gains resulting from multi-point search for both constraint optimization and constraint satisfaction problems.
4. Empirical evidence that is consistent with two conjectures regarding the relation between the clustering of solutions in search trees and the performance of multi-point constructive search.

References

1. D. Achlioptas, C.P. Gomes, H.A. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 256–261, 2000.
2. J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.

3. T. Carchrae and J.C. Beck. Low knowledge algorithm control. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI04)*, pages 49–54, 2004.
4. A. Cesta, A. Oddi, and S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.
5. P. R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass., 1995.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
7. F. Glover, M. Laguna, and R. Marti. Scatter search and path relinking: advances and applications. In G.C. Onwubolu and B.V. Babu, editors, *New Optimization Techniques in Engineering*. Springer, 2004.
8. D. Goddard, P. Laborie, and W. Nuijten. Randomized large neighborhood search for cumulative scheduling. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS05)*, 2005.
9. C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437, 1998.
10. C.P. Gomes, C. Fernàndes, B. Selman, and C. Bessiere. Statistical regimes across constrainedness regions. In *Proceedings of the Tenth International Conference on the Principles and Practice of Constraint Programming (CP2004)*, pages 32–46, 2004.
11. C.P. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *In Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.
12. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143:151–188, January 2003.
13. C. Le Pape. Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.
14. E. Nowicki and C. Smutnicki. An advanced tabu algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.
15. W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.
16. L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In *Proceedings of the Tenth International Conference on the Principles and Practice of Constraint Programming (CP2004)*, pages 468–481, 2004.
17. S.D. Prestwich. Combining the scalability of local search with the pruning techniques of systematic search. *Annals of Operations Research*, 115:51–72, 2002.
18. P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings of the Tenth International Conference on the Principles and Practice of Constraint Programming (CP2004)*, pages 557–571, 2004.
19. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 1, pages 362–367, 1994.
20. J. Singer, I.P. Gent, and A. Smaill. Backbone fragility and local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.
21. J.-P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123, 2002.
22. J.-P. Watson, J.C. Beck, A.E. Howe, and L.D. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143(2):189–217, 2003.