Multi-Point Constructive Search[⋆]

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 and quasi-group completion problems, multi-point constructive search performs significantly better than chronological backtracking and bounded backtracking with random restart.

1 Introduction

Metaheuristics such as path relinking [5] maintain a small number (e.g., five to ten) of "elite solutions," that are used to guide search into areas that appear promising. 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.

2 Multi-Point Constructive Search

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). The best solution found, s, is inserted into the elite set, in the former case, if it is better than the worst elite solution and, in the latter case, if it is better than the starting elite solution. Each search is limited by a fail-bound and the algorithm has some overall bound on the computational resources. The best elite solution is returned.

Searching from an Elite Solution Let a reference solution, r, be a set of variable assignments, $\{\langle V_1 = x_1\rangle, \ldots, \langle V_m = x_m\rangle\}, m \leq n$, where n is the number of variables. At a node in the search, any variable ordering heuristic can be used to choose a variable, V_i, to be assigned. If $x \in dom(V_i)$, where $\langle V_i = x\rangle \in r$, a choice point, $\langle V_i = x\rangle \vee \langle V_i \neq x\rangle$, is made. Otherwise, if $x \notin dom(V_i)$, any value ordering heuristic is used to choose $z \in dom(V_i)$ and a choice point is asserted using value z. An upper

⋆ This work has received support from ILOG, SA.

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($e$)
```

bound is placed on the cost function (line 10), and therefore a value assigned in the reference solution is not necessarily consistent with the current partial assignment. Note that our criterion for assigning a value from a reference solution covers the case where *no* value is assigned (i.e., when r is a partial solution and $m < n$).

Bounding the Cost Function Before each search (lines 6 and 11), we place an upper bound on the cost function. We experiment with three approaches. *Global bound*: Always set the upper bound on the search cost to the best cost found so far. *Local bound*: When starting from an empty solution, set the upper bound to be equal the cost of the worst elite solution. When starting from an elite solution, set the upper bound to be the cost of the reference solution. *Adaptive*: Use the global bound policy for $|e|$ searches whenever a new global best solution is found then revert to the local bound policy.

Elite Solution Initialization The elite solutions can be initialized by any search technique. In this paper, we use independent runs of 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.

Finding a Solution From Scratch A solution is found from scratch (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 a fail bound (lines 5 and 11). We associate a fail bound, initialized to 32, with each elite solution. Whenever search from an elite solution does not find a better solution, the corresponding fail bound is doubled. When an elite solution is replaced, the bound for 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 if a better solution is not found.

Adaptations for Constraint Satisfaction To adapt the approach to a satisfaction context, we rate *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.

3 Empirical Studies

Three variations of MPCS are used, corresponding to the different ways to set the cost bound: multi-point with global bound, *mpgb*; multi-point with local bound, *mplb*; and multi-point with adaptive bound, *mp-adapt*. We set $p = 0.5$ and $|e| = 8$. The effort to initialize the elite solutions is included in the results. For comparison, we use standard chronological backtracking (*chron*) and bounded backtracking with restart (*bbt*) following the same fail-bound sequence used for the multi-point techniques. The *bbt* algorithm is Algorithm 1 with line 12 is replaced by a copy of line 6. 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. 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.

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, has a pre-defined duration and a resource 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. We are interested in the optimization version of the problem: given a limited CPU time, return the solution with the smallest makespan found. Ten 20×20 JSPs were generated with randomly selected job routings and the activity durations independently and randomly drawn from $[1, 99]$ [8].

Randomized texture-based heuristics [3] and the standard constraint propagation techniques for scheduling [2] are used. To initialize the elite solutions, $|e|$ independent runs of a randomized algorithm that produces semi-active schedules is used. Each run is limited to 1000 fails. We compare algorithms based on mean relative error (MRE) as shown in Equation (1) where K is a set of problem instances, R is a set of independent runs, $c(a, k, r)$ is the lowest cost found by algorithm a on instance k in run r, and $c^*(k)$ is the lowest cost known for k.

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

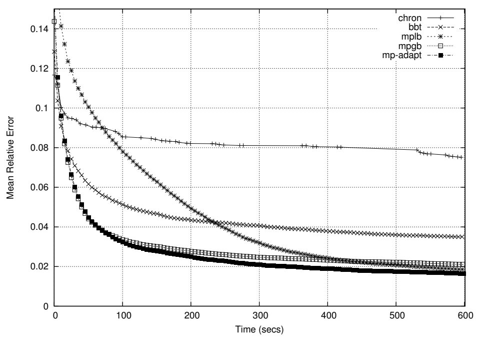
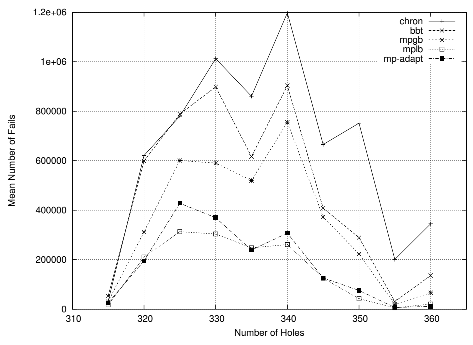


Fig. 1. Mean relative error to the best known solutions for each algorithm over ten independent runs of ten problem instances.



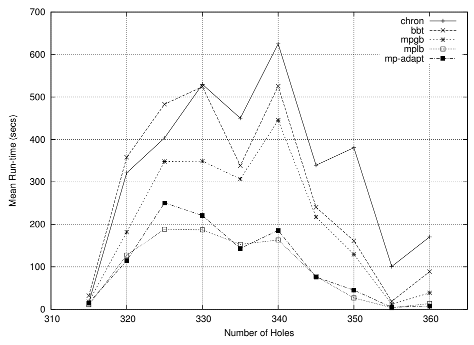


Fig. 2. Mean number of fails (left) and mean run-time (right) for the order-30 problems in each subset. Each point on the graph is the mean of ten independent runs on ten problem instances.

Figure 1 demonstrates that multi-point search is a significant improvement over both *chron* and *bbt*. 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 $t \geq 300$.

The Quasigroup-with-Holes Completion Problem An $n \times n$ quasigroup-with-holes (QWH) is a partially completed matrix where each row and column is required to be a permutation of the first n integers. A solution requires that all the empty cells ("holes") are consistently filled. The problem is NP-complete and bounded backtracking with randomized restart has been shown to be a strong performer [6]. We generated 100 balanced, order-30 QWH problems (i.e,. $n = 30$) using a generator that guarantees satisfiability [1]. Ten sets with problem instances each are generated with the number of

[1] A randomized paired-t test [4] and a significance level of $p \leq 0.005$ were used.

holes, $m = \{315, 320, \ldots, 360\}$. These values were chosen to span the difficulty peak identified in the literature. 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 above was used, implemented in ILOG Solver 6.0 on the same machine. The fail-limit to initialize each elite solution was set to 100 fails. The variable ordering heuristic randomly chooses a variable with minimum domain size while the value ordering is random. All-different constraints with extended propagation [7] are placed on each row and column.

Figure 2 presents the mean number of fails and mean run-time for each subset and algorithm. The MPCS variants *mplb* and *mp-adapt* perform significantly better than all other techniques for $m \geq 325$ for both the mean number of fails and the mean run-time.

4 Conclusion and Future Work

This paper introduces multi-point constructive search. The search technique maintains a small set of *elite* solutions that are used to conduct a series of resource-limited constructive searches. Depending on the outcome, 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 and satisfaction problems. Experiments are underway to systematically evaluate different parameter settings.

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. P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based Scheduling*. Kluwer Academic Publishers, 2001.
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. P. R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass., 1995.
5. 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.
6. 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.
7. 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.
8. 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.