An Empirical Study of Multi-Point Constructive Search for Constraint Satisfaction

Ivan Heckman & J. Christopher Beck

Department of Mechanical & Industrial Engineering
University of Toronto
{iheckman,jcb}@mie.utoronto.ca

Abstract. Multi-Point Constructive Search (MPCS) is a constructive search technique which borrows the idea from local search of being guided by multiple viewpoints. MPCS consists of a series of resource-limited backtracking searches: each starting from an empty solution or guided by one of a set of high quality, "elite" solutions encountered earlier in the search. This paper focuses on MPCS as applied to constraint satisfaction problems where elite solution quality is measured by the number of assigned variables in a partial solution. We systematically study different parameter settings including the size of the elite set, the probability of using an elite solution for guidance, and the use of chronological backtracking or limited discrepancy search. Experiments are performed on three constraint satisfaction problems: quasigroup-with-holes, magic squares, and multi-dimensional knapsack problems. Our results indicate that MPCS significantly out-performs both randomized restart and standard backtracking search on quasigroup-with-holes, performs about the same as randomized restart on the other problems, and is much worse than chronological on the multi-dimensional knapsack problems. The observed differences on two such similar problems (quasigroup-with-holes and magic square) suggests that these problems are a good testbed for future work to understand the reasons underlying the performance of MPCS.

1 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 population which is then filtered based on quality. In metaheuristics, such as advanced tabu search [7], a set of high quality solutions are maintained and revisited to intensify search in promising areas of the search space.

Multi-point Constructive Search (MPCS) [1] is an algorithm framework designed to allow constructive search to exploit multiple viewpoints. As with randomized restart techniques [4], search consists of a series of tree searches 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. Restarting from an elite solution

entails performing this fail-limited backtracking search starting from the guiding elite solution with a new randomized variable ordering. While preliminary experiments indicated that MPCS can significantly out-perform both standard chronological backtracking and randomized restart on optimization and satisfaction problems [1, 2], the one systematic study only addressed the former. Beck [3] applied MPCS to job shop scheduling problems with two different optimization criteria: makespan and weighted tardiness. The results indicated that MPCS significantly out-performs randomized restart and chronological backtracking on these problems. Yet, one of the best parameter settings found (i.e., maintaining only one elite solution) calls into question the exploitation of multiple viewpoints as the motivation for MPCS.

In this paper, we perform a similar systematic study of MPCS parameter values for constraint satisfaction problems. The primary modification to the optimization version of the algorithm is a change in the definition of an elite solution. Rather than comparing complete solutions using a cost function, we compare partial solutions based on the number of assigned variables. We vary the primary parameters of the MPCS algorithm in a detailed set of experiments using the quasigroup-with-holes problem. The results indicate that, unlike for the scheduling problems, maintaining more than one elite solution leads to stronger performance. The preliminary results in comparing MPCS with randomized restart and chronological backtracking on quasigroup-with-holes are confirmed. Interestingly, experiments on magic squares and a satisfaction variant of multi-dimensional knapsack problems show performance that is about the same as randomized restart. MPCS and randomized restart are significantly better than chronological backtracking for the magic square problems but significantly worse on the multi-dimensional knapsack problems.

In the next section, we present the MPCS framework as applied to constraint satisfaction and the parameter space that will be investigated in this paper. We then turn to the empirical studies, varying the parameters on each of the three satisfaction problems. We present detailed results for the quasigroup-with-holes experiments and, due to space restrictions, summary results for the other problems. Finally, we discuss our results and their implications for developing an understanding of why and how MPCS works.

2 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 5) or from a randomly selected elite solution (line 10). In the former case, if the best partial solution found during the search, s, is better than the worst elite solution, s replaces the worst elite solution. In the latter case, s replaces the starting elite solution, r, if s is better than r. Each individual search is limited by a fail bound: a maximum number of fails that can be incurred. The entire process ends when the problem is solved or proved insoluble within one of the iterations, or when some overall bound on the computational resources (e.g., CPU time, number of fails) is reached.

As the MPCS framework has been presented previously [3], we only briefly describe the algorithm details here.

Algorithm 1: MPCS: Multi-Point Constructive Search

```
   MPCS():
1  initialize elite solution set e
2  while not solved and termination criteria unmet do
3      if rand[0,1) < p then
4          set fail bound, b
5          s := search(∅, b)
6          if s is better than worst(e) then
7              replace worst(e) with s
       else
8          r := randomly chosen element of e
9          set fail bound, b
10         s := search(r, b)
11         if s is better than r then
12             replace r with s
```

– *Elite Solution Initialization* The elite solutions can be initialized by any search technique. We use independent runs of a standard chronological backtracking with a randomized heuristic. The search effort is limited by a maximum number of fails for each run.
– *Bounding the Search* Each individual search is bounded by a fail bound: a single search (lines 10 and 5) will terminate, returning the best solution encountered, after it has failed the corresponding number of times.
– *Searching From An Empty Solution* Searching from an empty solution (line 5) simply means using any standard constructive search with a randomized heuristic and a bound on the number of fails.
– *Searching from a Solution* To start constructive search from an elite solution, we create a search tree using any variable ordering heuristic and specifying that the value assigned to a variable is the one in the elite solution, provided it is still in the domain of the variable. Otherwise, any other value ordering heuristic can be used to choose a value. Formally, given a constraint satisfaction problem with n variables, a solution, s, is a set of variable assignments, $\{\langle V_1 = x_1\rangle, \langle V_2 = x_2\rangle, \ldots, \langle V_m = x_m\rangle\}, m \leq n$. When $m = n$, the solution is complete; when $m < n$, s is a partial 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. The variable ordering heuristic has complete freedom to choose a variable, V_i, to be assigned. If $\langle V_i = x_i\rangle \in s$ and $x_i \in dom(V_i)$, the choice point is made with $x = x_i$. Otherwise any value ordering heuristic can be used to choose $x \in dom(V_i)$.

2.1 Adapting Elite Solutions to Satisfaction Problems

In an optimization context, a solution can be defined as a complete, feasible assignment of all variables. Solutions can be compared based on their corresponding objective value

or cost. To adapt MPCS for satisfaction, we relax the need for a complete assignment and compare solutions based on the number of assigned variables.

The solution, s, returned from a single search is the partial solution with the most assigned variables encountered during the search. Either this is a complete solution satisfying all constraints and so search terminates or it is a dead-end. Clearly the partial solution encountered with the greatest number of assigned variables must be either a complete solution or a dead-end. When we encounter a dead-end, we make no attempt to further assign variables: as soon as there is a domain wipe-out, we evaluate the partial solution by counting the number of assigned variables and then backtrack, provided we have not reached the resource limit on the search.

2.2 MPCS Parameter Space

There are a number of parameter values that must be specified in order to implement the MPCS algorithm. We concentrate on three parameters that appear most interesting based on previous work with optimization problems [3]:

- *Elite Set Size* Previous studies of MPCS for satisfaction problems [1, 2] used an elite set size of 8. However, results for optimization problems point to an elite set size of 1 as performing best. 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. In this paper, we study $p = \{0, 0.25, 0.5, 0.75, 1\}$. Note that $p = 1$ is equivalent to randomized restart.
- *Backtrack Method* For a single search, we have a choice as to how the search should be performed. In particular, we experiment with using standard chronological backtracking or limited discrepancy search (LDS) [6]. In either case, the search is limited by the fail bound as described below.

There are a number of other parameters of the MPCS algorithm that are not experimented with here. Specifically:

- *Fail Sequence* The resource bound sets the number of fails allowed for each search. For all our algorithms we use the polynomially growing bound: 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 is increased by adding 32. The value 32 was chosen to give a reasonable increase in the fail limit on each iteration. The polynomial sequence is adopted here for its simplicity and is consistently among the best sequences in previous work.
- *Initialization* There are two parameters associated with the initialization of the elite set: the number of solutions that are found and the resource bound for each of the initial solutions. It has been shown in previous work [3] that simply initializing each elite solution can skew results that examine changing $|e|$. Therefore, we always create 20 initial solutions and then select the $|e|$ best for inclusion in the elite set. The resource bound is simply the number of fails we allow for each initialization. Previous experiments have shown no positive impact in terms of final solution quality

from a large initialization fail bound [3]. As preliminary experiments with quasigroup problems confirmed this trend, we do not experiment with the initialization fail bound here but rather set the bound to 1000 for all experiments.

– *Setting an Upper Bound on Solution Cost* In an optimization context, constraint programming methods typically place an upper bound on the cost function, redefining the set of feasible solutions to be those that are better than any solution seen so far. Such an approach is not meaningful in a satisfaction context because a bound on the number of assigned variables has no impact during the search (i.e., it does not propagate). Therefore, we place no bound on the cost function in the individual searches and solutions are admitted to the elite set by the criteria defined in Algorithm 1. This is the "local" bounding or mplb method in [3].

2.3 MPCS as Local Search

While the core search technique in MPCS is heuristic tree search, there are two ways in which MPCS can be viewed as a hybrid of constructive and local search. First, MPCS is based on a fundamental idea of local search: the use of sub-optimal solutions to guide search. Second, and more crucially, a single iteration of MPCS starting from an elite solution is an implicit search over a neighborhood of that solution. Given a variable ordering and a resource limit, a chronological backtracking[1] tree search is only able to search through a small subtree before the resource bound is reached. That subtree is, implicitly, a neighborhood of the starting solution. If a better solution is found in that neighborhood, it is accepted and inserted into the elite set where it will be later used as a starting solution for a new neighborhood search. If a better solution is not found in the neighborhood, a subsequent search with the same starting solution but a different variable ordering and/or resource limit will investigate a different neighborhood of that elite solution. From this perspective, heuristic tree search is used to implement the evaluation of neighboring solutions. While this is reminiscent of previous work [8], through an implicit definition of the neighborhood via the variable ordering, resource limit, and style of tree exploration, MPCS does not require sophisticated, often domain-dependent, neighborhood engineering.

3 Empirical Study

Due to time and space limitations we do not conduct a fully crossed experiment with all values of the parameters. For most of the experiments, one parameter is varied while the others were set to default values. Based on preliminary experimentation and past studies the following values are used as defaults:

3.1 Problems

The experiments were performed on three different satisfaction problems: quasigroup-with-holes, magic squares, and multi-dimensional knapsack. These problems were chosen because benchmark sets exist and randomized restart shows an interesting pattern

[1] This perspective is equally valid for other styles of tree exploration such as LDS. For clarity we concentrate on the chronological case.

Fail Seq.	$\|e\|$	p	Init. Fail Bound	Backtrack Method	Elite Replacement
poly	8	0.5	1000	chron	mplb

Table 1. Default parameter values for the experiments.

of performance: performing well on quasigroup problems [5] and poorly on multi-dimensional knapsack [9]. Magic square problems appear to bear similarities to quasi-group-with-holes. Our reason for focusing on problems with interesting performance of randomized restart is that MPCS can be interpreted as a form of guided randomized restart and therefore we are interested in its behaviour as randomized restart behaviour varies.

More specifically, the problems we use are as follows:

- *Quasigroup-with-Holes Completion Problem* An $n \times n$ quasigroup-with-holes (QWH) completion problem is a matrix where each row and column is a permutation of the first n integers 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. This problem is modeled by all-different constraints with extended propagation [10] on each row and column. Two sets of problem instances are used: (i) 100 order-30 instances used previously to study MPCS [1] divided into ten subsets according to the number of holes: $m \in \{315, 320, .., 360\}$ (ii) a set of existing benchmark instances [5, 9]. Each instance is solved ten times with different random seeds and a fail limit of 2,000,000 fails.
- *Magic Squares* An $n \times n$ magic square is a matrix of the numbers $1, .., n^2$ whose rows, columns, and diagonals all sum to $S = \frac{n \times (n+1)}{2}$. These problems are modeled with one all-different constraint and by constraining the sum of each row, column, and diagonal to be S. For $n = \{10, .., 15\}$ results were averaged over 20 independent runs with different random seeds with a limit of 10,000,000 fails.
- *Multi-Dimensional Knapsack* Given a knapsack with m dimensions such that each dimension has capacity, $c_1, \ldots, c_m$, a multi-dimensional knapsack problem requires the selection of a subset of the n objects such that the profit, $P = \sum_{i=1}^{n} x_i p_i$, is maximized and the m dimension constraints, $\sum_{i=1}^{n} x_i r_{ij} \leq c_j$ for $j = 1, \ldots, m$, are respected. Each object, i, has a individual profit, p_i, and a size for each dimension, r_{ij}. This problem can be posed as a satisfaction problem by constraining P to be equal to the known optimal value [9]. Six problems are used from the operations research library[2] which have 15 to 50 variables and 6 to 11 constraints. For each problem, results were averaged over 20 independent runs with different random seeds and a limit of 10,000,000 fails. For each problem, results were averaged over 20 independent runs with different random seeds and a limit of 10,000,000 fails.

3.2 Experimental Details

Each of the problem types and search methods used a minimum domain variable ordering with randomization on ties. The value ordering for each problem and technique,

[2] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

when not being guided by an elite solution, is random. All algorithms were implemented in ILOG Scheduler 6.0 and run on a 2.8GHz Pentium 4 with 512Mb RAM running either Fedora Core 2 or Red Hat Enterprise 4.

3.3 Initial Quasigroup Experiments

Our first set of experiments uses the set of order-30 QWH problems to examine the impact of various parameter settings. In the next section we examine the performance of a number of the best settings found on the second set of benchmark problem instances.

Elite Set Size The results for varying $|e|$, the number of elite solutions maintained, are shown in Figure 1. In contrast to previous results on scheduling problems [3], with QWH an elite size of one is worse than any of the other sizes, especially at $m = \{325, 330\}$. The best performance is achieved by $|e| = 4$ or $|e| = 8$. On the scheduling problems, the best performance was achieved, in some cases, by $|e| = 1$.

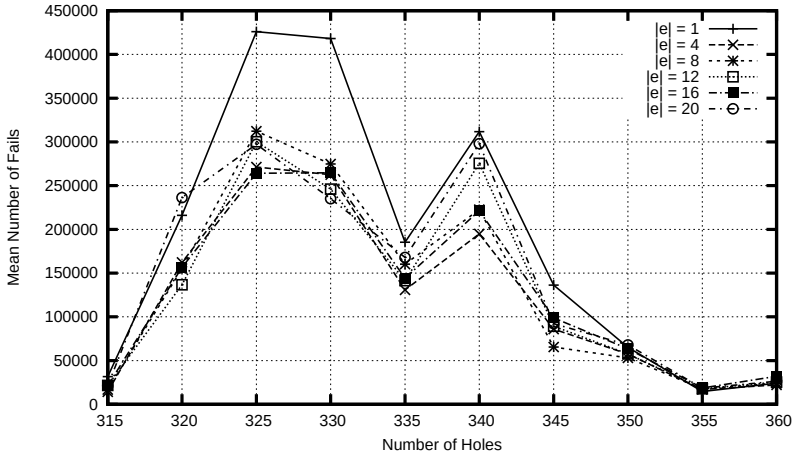

Fig. 1. Mean number of fails to solve order-30 problems in each subset for varying values of $|e|$.

The Probability of Searching from an Empty Solution The p parameter is the probability that search will be done from an empty solution vs. being guided with an elite solution. Figure 2 shows that the effort to solve the QWH problems monotonically increases with increasing p. The best result is achieved by always guiding the search with an elite solution ($p = 0$) while the worst performance is to always search from an empty solution ($p = 1$). These results agree somewhat with the scheduling results where it was shown that $p = 0.25$ delivered the best performance with decreasing performance for $p \geq 0.5$ and with $p = 0$ performing worse than $p = 0.25$.

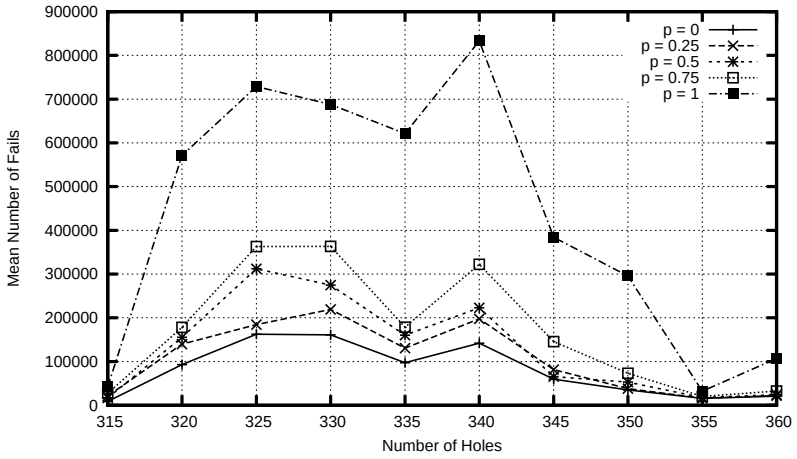

Fig. 2. Mean number of fails to solve order-30 problems in each subset for varying p-values.

The Interaction Between $|e|$ and p We believe it is likely that there is an interaction between the size of the elite set and the probability of searching from an elite solution. To examine possible interactions, a full cross of these two parameters was done. Figure 3 shows the mean results of all 1000 runs over all 100 problems for a given setting of p and $|e|$. It again shows that a lower probability of starting from an empty solution performs better and a elite size of 1 performs poorly. Overall, the combination $|e| = 8$ and $p = 0$ performed the best.

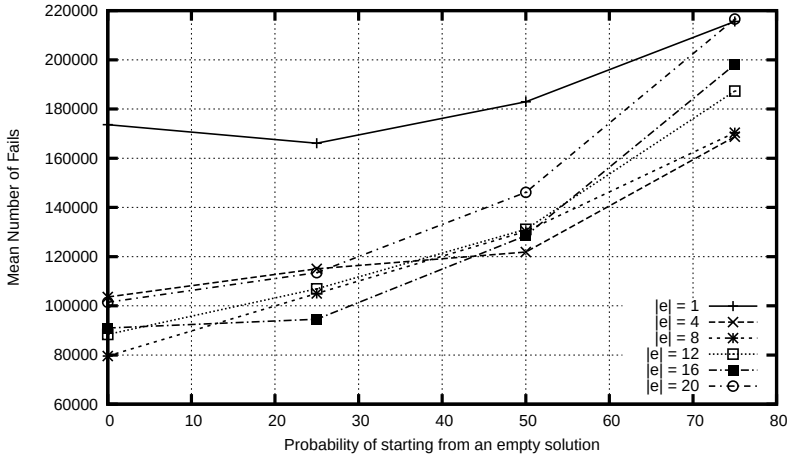

Fig. 3. Mean number of fails to solve order-30 problems for all values of $|e|$ and p.

Backtrack Method Finally, we examine the impact of using chronological backtracking or LDS for each individual search. Figure 4 shows that, in terms of the number of fails, LDS results in better performance.

However, as shown in Figure 5, despite incurring fewer fails, LDS has a significantly longer run time.[3] The time per fail is larger for LDS because it spends much less time deep in the tree than chronological backtracking. Therefore, the computational effort of the choice points high in the tree is not amortized over as many fails.

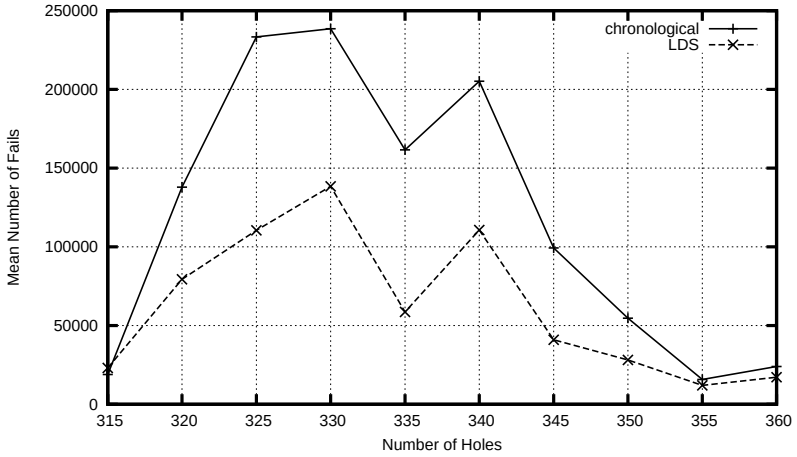

Fig. 4. Mean number of fails to solve order-30 problems in each subset for both backtracking methods.

Summary Overall, these experiments indicate that for the QWH problem the following parameter values perform best: $|e| = 8$, $p = 0$, and chronological backtracking. Recall that we did not vary the fail sequence, initialization fail bound, or method for bounding the cost of solutions.

3.4 Benchmark Quasigroup Problems

Using the best parameter settings from the initial QWH experiments, in this section we apply MPCS to an existing set of QWH benchmarks and compare the performance with standard chronological backtracking (*chron*) and randomized restart (*restart*). In all algorithms the same randomized minimum domain variable ordering and random value ordering are used.

The restart algorithm follows the same polynomial fail sequence as the MPCS methods and initializes and maintains a set of elite solutions. However, it always searches from an empty solution (i.e., it is equivalent of MPCS with $p = 1$).

[3] Unless specifically mentioned, the comparison of methods based on the mean number of fails is identical to the comparison based on mean run time.

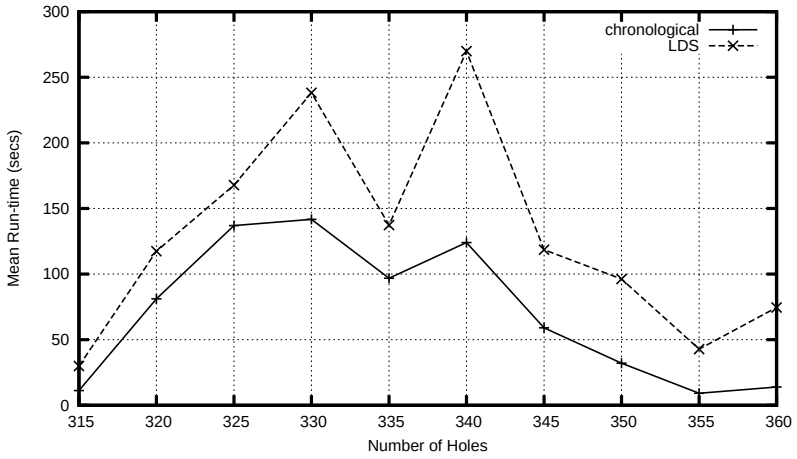

Fig. 5. Mean time to solve order-30 problems in each subset for both backtracking methods.

Each problem was run 10 times for each algorithm. The percent of runs that were able to find a solution in the 2,000,000 fail limit, the mean number of fails to find a solution, and the mean solve time in seconds are reported in Table 2. Bold entries indicate the best result (either lowest mean number of fails or lowest mean run-time) for each problem instance. When a run failed to find a solution, the fail limit is used in calculating the mean.

The pattern of bold entries in Table 2 is summarized in Table 3. MPCS achieves the lowest mean number of fails in 18 problem instances and the lowest run-time for 14. Furthermore, on 20 of the instances, all 10 runs of the algorithm found a solution within the global fail limit. Exceptions to this pattern arise at higher orders and with no filled in values. Here standard chronological search performed the best and MPCS the worst.

Table 4 compares our results with previous [9] results on the same benchmark problems using impact-based heuristics combined with restarts. Shown are the number of choice points reported by impact-based search along with the percentage of successful runs and mean number of choice points for MPCS to solve the same problems. A direct comparison is complicated by the fact that [9] limited the search to 1500 seconds where ours was limited by 2,000,000 fails. While there are several instances where the impact-based heuristic beats MPCS (as shown by the bold entries), MPCS clearly beats it on some of the hardest problems, and is able to reliably solve at least two more instances. Note however that MPCS incurs fewer choice points in only 6 of the 19 instances versus 10 of 19 for the impact-based heuristic. Given that impact-based heuristics could be used as a variable ordering heuristics within MPCS, it would be interesting to look at combining these approaches.

3.5 Magic Square Experiments

Despite the apparent similarities between quasigroup-with-holes and magic square problems, the performance of the MPCS algorithm is quite different. While space prevents

		chron			restart			MPCS-best		
order	holes	%sol	fails	time	%sol	fails	time	%sol	fails	time
30	316	100	6458	3.1	100	679	0.6	100	**289**	**0.2**
30	320	100	325	**0.2**	100	327	0.3	100	**267**	0.2
33	381	0	2000000	1244.7	0	2000000	1726.7	0	2000000	1376.2
35	405	40	1566506	979.0	50	1740792	1458.7	100	**185383**	**130.4**
40	1600	100	1	**2.8**	100	**0**	2.9	100	**0**	2.9
40	528	0	2000000	1469.9	0	2000000	1684.4	40	**1675930**	**1335.1**
40	544	0	2000000	1461.7	0	2000000	1671.9	80	**693720**	**558.9**
40	560	0	2000000	1359.2	0	2000000	1614.2	100	**132751**	**109.3**
50	2500	100	**5**	**8.6**	100	8	**8.6**	100	**5**	8.7
50	2000	100	12	**3.6**	100	**3**	**3.6**	100	12	**3.6**
50	825	0	2000000	2125.9	0	2000000	2619.3	0	2000000	2515.6
60	3600	100	**2**	**21.2**	100	21	23.3	100	11	21.5
60	1440	0	2000000	2047.6	60	1367260	2126.6	100	**51251**	**121.7**
60	1620	20	1627363	1574.1	80	1043824	1695.6	100	**63185**	**169.4**
60	1692	80	824779	748.3	100	82952	218.4	100	**13758**	**70.2**
60	1728	100	303789	259.5	100	35235	125.3	100	**11019**	**65.5**
60	1764	80	682079	646.3	100	26566	102.1	100	**4669**	**40.3**
60	1800	80	765919	665.6	100	25139	88.8	100	**5174**	**45.3**
70	4900	100	147	**46.0**	100	33	55.2	100	**31**	46.7
70	2450	40	1415351	1695.5	100	714045	2023.9	100	**50557**	**331.1**
70	2940	100	38578	**45.7**	100	3300	115.9	100	**1390**	77.1
70	3430	100	838	**10.9**	100	495	59.3	100	**190**	26.0
90	8100	100	**154**	**157.4**	100	168	367.9	100	289	593.4
100	10000	100	5429	**275.1**	100	**441**	1553.4	100	839	2437.5

Table 2. QWH benchmark comparison with other search algorithms.

a full presentation of the results as done with the QWH problems, we can summarize the results as follows:

- Elite Set Size: varying the elite set size has little effect on the algorithm performance.
- The Probability of Searches from an Empty Solution: Results for varying the probability of starting from an empty solution are shown in Figure 6. Unlike the QWH problems, MPCS does not out-perform randomize restart (i.e., $p = 1$) on the magic squares problems. In fact, $p = 1$ results in the best performance while $p = 0$, the best setting on the QWH problems, performs worst.
- Backtrack Method: Using LDS on the magic square instances led to extremely bad performance. On the easiest order-10 instances, no solutions were ever found using a global limit of 10,000,000 fails.

Displayed in Figure 7 is a comparison of MPCS with the best settings found in the quasigroup (*MPCS:qwh*) and magic squares (*MPCS:magic*) experiments, and the other search algorithms. For MPCS:magic the following parameters are used: $|e| = 8$, $p = 0.75$,[4] and the backtrack method is chronological. *Restart* and the MPCS variations are all better than chronological search. MPCS performs poorly compared to restart.

[4] The second best p is taken since it is not MPCS at $p = 1$.

	chron	restart	MPCS-best
# best fails	3	3	18
# best time	10	2	14
# 100% solved	12	16	20

Table 3. Summary Statistics for Table 2.

		MPCS		Impact-based
order	holes	%sol	choice pts.	choice pts.
18	120	100	11	**2**
30	316	100	358	**31**
30	320	100	310	**278**
33	381	0	2025235	-
35	405	100	**192720**	752779
40	528	40	1738612	-
40	544	80	**739356**	-
40	560	100	**161053**	289686
50	2000	100	1737	**1735**
50	825	0	2171697	-
60	1440	100	**154308**	-
60	1620	100	199879	**56050**
60	1692	100	**84941**	164048
60	1728	100	76625	**2333**
60	1764	100	**47068**	48485
60	1800	100	50487	**1934**
70	2450	100	246042	**43831**
70	2940	100	36737	**3732**
70	3430	100	7581	**3073**

Table 4. QWH comparison with Impact Based Search [9].

These results are intriguing, given the QWH results and the similarities between QWH and magic squares. We will return to these results in the discussion below.

3.6 Multi-Dimensional Knapsack Experiments

As with the magic squares experiments, due to space limitations, we focus on the comparison of MPCS with the best parameter values on the multi-dimensional knapsack problems, MPCS with the best parameter values on the QWH problems, restart, and chronological backtracking.

In the knapsack experiments, the best settings for MPCS were found to be the initial default settings, with $|e| = 8$, $p = 0.5$, and chronological backtracking performing slightly better than other parameter settings. Yet as seen in Table 5, both MPCS and randomized restart perform poorly in comparison to basic chronological search. Significantly better performance than chronological backtracking on these problems is presented in [9]. The best MPCS settings perform about the same as restart.

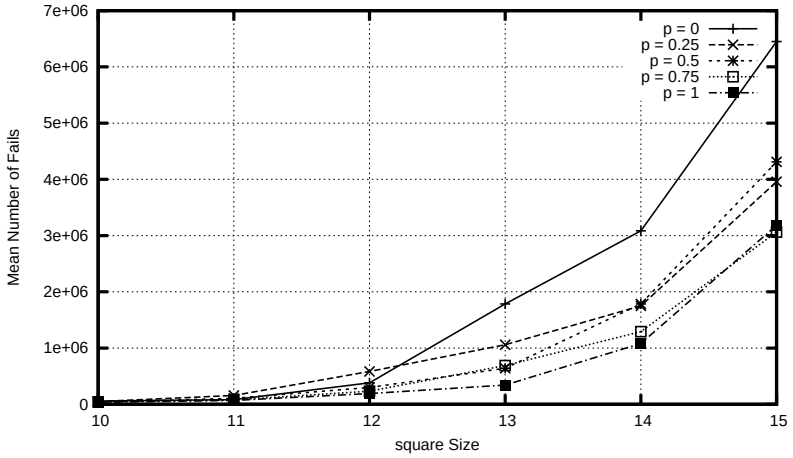

Fig. 6. Mean number of fails to solve magic squares problems with varying values for p.

	chron			restart			MPCS-qwh best			MPCS-knap best		
	%sol	fails	time	%sol	fails	time	%sol	fails	time	%sol	fails	time
mknap1-0	100	1	**0.0**	100	**0**	0.0	100	1	0.0	100	0	**0.0**
mknap1-2	100	26	0.0	100	**22**	**0.0**	100	24	0.0	100	23	**0.0**
mknap1-3	100	**363**	**0.0**	100	418	0.0	100	724	0.1	100	660	0.1
mknap1-4	100	**15551**	**1.1**	100	53938	4.8	100	30939	3.1	100	48219	4.4
mknap1-5	100	**2862059**	**148.3**	55	8141502	552.2	85	5035286	376.6	80	4156366	287.7
mknap1-6	0	10000000	**660.6**	0	10000000	843.7	0	10000000	938.5	0	10000000	857.8

Table 5. Comparison of multi-dimensional knapsack results for different algorithms and best MPCS parameter settings.

4 Discussion and Future Work

The goal of this paper was to apply multi-point constructive search to constraint satisfaction problems and to perform a systematic evaluation of the various parameter settings. As the extensive experiments on the quasigroup-with-holes problems indicate, MPCS is able to significantly out-perform both randomized restart and chronological backtracking on constraint satisfaction problems. While the best parameter values tended to agree with those found on scheduling problems [3] (i.e., low $|e|$ value, low p value), the QWH results showed that maintaining more than one elite solution improves search. This is an important finding as the scheduling results found very good performance with $|e| = 1$, calling into question the intuition that the observed performance gains could be due to exploiting multiple viewpoints. The QWH results are a proof of concept for MPCS on constraint satisfaction problems.

However, when the empirical results for the magic squares and multi-dimensional knapsack are considered, our conclusions are more nuanced and interesting. The significant change in the relative performance of randomized restart and MPCS when moving from the QWH to the magic squares problems is particularly interesting given the simi-

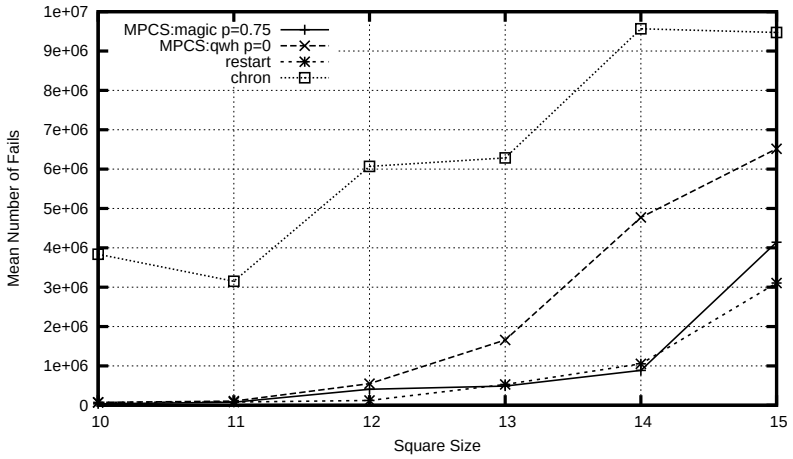

Fig. 7. Mean number of fails comparing different search techniques and MPCS with best parameters found for magic squares.

larities in the problems. What is it about the differences between the problems that lead to strong MPCS performance on QWH and weak performance on magic squares? We hope that answering this question will lead us to an understanding of the problem characteristics that influence MPCS performance and ultimately to an understanding of the reasons for MPCS performance. As a starting point, we believe it would be interesting to generate some magic-squares-with-holes problems to determine if the phase transition behaviour of QWH is seen and to evaluate the performance of randomized restart and MPCS.

Interestingly, it appears that the multi-dimensional knapsack problems create a particular challenge for MPCS. Detailed traces of the MPCS runs showed that very quickly all the elite solutions had an objective value of n: all of the variables were assigned but the solution does not satisfy all constraints. In other words, the linear constraint propagation tended not to result in empty domains but rather a fully assigned variables that broke one or more constraints. Further experiments modified our objective to be the sum of the number of assigned variables and the number of satisfied constraints. Under these conditions, in many situations, though not always, the elite solutions were soon populated with solutions that assigned all variables and only broke one constraint: the overall cost constraint: $\sum_{i=1}^{n} x_i p_i = C$.[5] Under these conditions, we speculate that MPCS gets poor heuristic guidance: the elite solutions do not differentiate between solutions that are close to the optimal cost and those far away and so the elite set quickly stagnates to the first $|e|$ solutions found that only break the cost constraint. Preliminary experiments using MPCS to solve the optimization version of the multi-dimensional knapsack problem show that it significantly out-performs all other techniques in finding the optimal solution. This implies that the actual cost of a solution provides a better criteria for inclusion in the elite set and much better heuristic guidance.

[5] Recall that we made the multi-dimensional knapsack a satisfaction problem, following Refalo, by requiring that the cost be equal to the (previously known) optimal cost, C.

5 Conclusion

This paper is the first systematic application of multi-point constructive search to constraint satisfaction problems. Our empirical results demonstrated that MPCS can perform significantly better than chronological backtracking and randomized restart on constraint satisfaction problems. In particular, our experiments with quasigroup-with-holes problems showed that a relatively small elite pool and a zero probability of searching from an empty solution lead to such strong results. In general, such results are in agreement with previous studies on optimization problems in the scheduling domain, however the results in this paper reinforce the intuition that exploiting multiple viewpoints can be of substantial benefit in heuristic search.

The types of problems that we experimented with revealed an interesting pattern. MPCS significantly out-performs chronological backtracking on the quasigroup-with-holes and magic squares problems but significantly under-performs on the multi-dimensional knapsack problems. Similarly, MPCS out-performed randomized restart on the quasigroup problems, performed slightly worse on the magic squares problems, and performed about the same on the multi-dimensional knapsack problems. This variety of results leads us to consider these three problem types as important for future work in understanding the reasons for the behaviour of MPCS. If we can explain these varied results, we will be substantially closer to explaining the behaviour of MPCS.

References

1. J. C. Beck. Multi-point constructive search. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP05)*, 2005.
2. J. C. Beck. Multi-point constructive search: Extended remix. In *Proceedings of the CP2005 Workshop on Local Search Techniques for Constraint Satisfaction*, pages 17–31, 2005.
3. J. C. Beck. An empirical study of multi-point constructive search for constraint-based scheduling. In *Proceedings of the Sixteenth International on Automated Planning and Scheduling (ICAPS'06)*, 2006.
4. 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.
5. 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.
6. W. D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Department of Computer Science, Stanford University, 1995.
7. E. Nowicki and C. Smutnicki. An advanced tabu algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.
8. G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.
9. 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.
10. 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.