# Empirical Analysis of Solution Guided Multi-Point Constructive Search

by

Ivan Heckman

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

# Abstract

Empirical Analysis of Solution Guided Multi-Point Constructive Search

Ivan Heckman
Master of Science
Graduate Department of Computer Science
University of Toronto
2007

Solution Guided Multi-Point Constructive Search (SGMPCS) is a constructive search technique inspired by local search algorithms that guide search from multiple viewpoints. SGMPCS 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. The thesis of this dissertation is that SGMPCS works, in part, because of two factors: the benefit of revisiting the areas near good solutions, and through the exploitation of heavy-tails. We provide a detailed analysis of the various parameters of SGMPCS on a variety of constraint satisfaction problems. We show evidence of heavy-tailed distributions for only the sets of problems where SGMPCS performs well. We show how performance is correlated with how well the evaluation of guiding solutions predicts actual distance to a solution. Finally, we explore various static cost models of SGMPCS performance.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Solution Guided Multi-Point Constructive Search (SGMPCS) is a complete, constructive search technique that has been shown to out-perform standard constructive search techniques on a number of constraint optimization and constraint satisfaction problems. The central thesis of this dissertation is that, as first speculated by Beck [7], there are at least two, non-mutually exclusive factors that influence SGMPCS performance: the exploitation of heavy-tails and the impact of revisiting elite solutions. The body of this dissertation consists of empirical investigations toward gaining a greater understanding of SGMPCS behaviour, focusing on these two factors.

In particular, in this dissertation:

- We perform a systematic investigation on how the various parameters of SGMPCS affect performance in solving three satisfaction problems.
- We investigate if and when heavy-tailed distributions appear in job-shop scheduling optimization, and their relation to SGMPCS performance.
- We investigate the impact of the evaluation function used by SGMPCS through fitness-distance correlation analysis.
- We evaluate static cost models of SGMPCS, which attempt to predict performance from static search space features of an instance.

## 1.1 Motivations

This dissertation is focused on gaining a better understanding of SGMPCS performance. As such, its motivations are the past performance SGMPCS, and previous empirical studies of SGMPCS, heavy-tails, and descriptive models of algorithm behaviour.

1. **Past Performance and Studies of SGMPCS:** While previous results [4, 5] indicate that SGMPCS can significantly out-perform both standard chronological backtracking and randomized restart on optimization and satisfaction problems, the one existing systematic study of the parameters of SGMPCS [6] only addressed optimization problems. Beck [6] showed that SGMPCS significantly out-performs randomized restart and chronological

backtracking on two different types of job-shop scheduling problem. 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 SGMPCS. One goal of this dissertation is to perform a similar systematic study of SGMPCS parameter values for constraint satisfaction.

2. **Heavy-Tailed Distributions in Constructive Search:** The studies of Gomes et al. [29] found that constructive search can exhibit tremendous variability in the cost of finding a solution, with distributions in search cost that can be characterized as heavy-tailed. These empirical studies led to a randomized rapid restart technique that can greatly improve performance of constructive search algorithms. As SGMPCS can itself be considered a randomized restart algorithm, one goal of this dissertation is to show theoretically and empirically that SGMPCS can also exploit heavy-tailed distributions.

3. **Descriptive Models of Algorithm Behaviour** A descriptive model of algorithm behaviour is a tool used to understand why an algorithm performs as it does on a particular class or instance of a problem. There has been considerable work over the past 15 years in developing models of problem hardness [21, 61] as well as work that has focused more directly on modeling the behaviour of specific algorithms or algorithm styles. This dissertation employs various descriptive models of SGMPCS performance as a means to better understand how and when it outperforms other techniques.

## 1.2 Outline

The outline of this dissertation is as follows:

In Chapter 2, we first define constraint satisfaction and optimization problems and the two main categories of algorithms to solve them. We then introduce Solution Guided Multi-Point Constructive Search, and various other algorithms that influenced its development or share common features. We then give an overview of empirical research into search algorithms and search spaces.

In Chapter 3, we vary the primary parameters of the SGMPCS algorithm in a detailed set of experiments on three satisfaction problems: quasigroup-with-holes, magic square, and a satisfaction version of the multi-dimensional knapsack problem. The results confirm previous results in comparing SGMPCS with randomized restart and chronological backtracking on quasigroup-with-holes and indicate that maintaining more than one elite solution leads to stronger performance. Interestingly, experiments on magic square and a multi-dimensional knapsack problems show performance that is about the same as randomized restart.

In Chapter 4, we review past research on heavy-tailed distributions in backtracking search including how to find the distributions and how to theoretically and empirically boost performance with a rapid-restart strategy. We argue that SGMPCS should theoretically also benefit the same way. Empirical investigations are then performed to confirm the theoretical argument: using job-shop scheduling optimization problems, we show that SGMPCS performance improves over basic backtracking at precisely the same sizes of problem at which heavy-tailed distributions appear in backtracking search.

In Chapter 5, we return to the interesting results of the multi-dimensional knapsack problems of Chapter 3 to investigate the conjecture that SGMPCS performance, unlike that of randomized restart and chronological backtracking, is partially affected by the quality of the heuristic that is used to select the guiding partial solutions. This conjecture is explored by developing a descriptive model of SGMPCS performance using fitness-distance analysis. It is demonstrated that SGMPCS search performance is partially dependent upon the correlation between the heuristic evaluation of the guiding solutions and their distance to the nearest satisfying solution

In Chapter 6, we explore how SGMPCS performance may be related to other search space features by an evaluation of various static cost models used previously by Watson et al. [70] on job-shop scheduling problems and tabu search.

Chapter 7 concludes this dissertation with a summary of contributions and suggestions for further work.

## 1.3 Summary of Contributions

The contributions of this dissertation are as follows:

- We perform the first systematic application of SGMPCS search to constraint satisfaction problems confirming good results on quasigroup-with-holes problems, and otherwise interesting results on two other satisfaction problems.

- We show that the distribution in run-times of the backtracking search on JSP instances can be heavy-tailed. We also show that SGMPCS and randomized restart techniques perform better at the same instance sizes in which heavy tails start to appear. Guided runs used by SGMPCS were also shown to exhibit heavy tails.

- Empirical results, both in an artificial context and using three different heuristic evaluation functions, demonstrate that the correlation between the heuristic evaluation of a state and its proximity to the satisfying solution has a strong impact on search performance of SGMPCS.

- We are able to take search space features, which were previously only used in analyzing local search procedures, and show interesting relations between them and SGMPCS performance. Surprising relations with the performance of standard chronological backtracking and randomized restart were also found.

# Chapter 2

# Literature Review

In this chapter, we first define constraint satisfaction and optimization problems and the two main categories of algorithms to solve them. We then introduce Solution Guided Multi-Point Constructive Search, and other algorithms that influenced its development or share common features. We then give an overview of empirical research into search algorithms and search spaces.

## 2.1 Constraint Satisfaction and Optimization Problems

Many interesting and important problems in Artificial Intelligence, from industrial scheduling and routing to protein folding [8, 17, 2], can be modeled and solved as Constraint Satisfaction or Optimization Problems. In a Constraint Satisfaction Problem (CSP), the goal is to assign values to a set of variables in such a way that a set of constraints between them hold. Constraint Optimization Problems (COP) have the added task of finding an optimal satisfying assignment of the variables. Formally, a CSP can be represented as a three-tuple $(V, D, C)$, where $V$ is a set of variables $V = \{v_1, v_2, \cdots, v_n\}$ each with corresponding domains from $D = \{D_1, \cdots, D_n\}$. $C$ is a set of $m$ constraints, $C = \{c_1, \cdots, c_m\}$ where each $c_i$ defines, over some subset of $V$, $(v_i, \cdots, v_j)$, the acceptable combination of assignments to these variables from the Cartesian product $D_i \times \cdots D_j$. The goal of a CSP is to find an assignment to all variables $\alpha = \{\langle v_1 = x_1\rangle, \langle v_2 = x_2\rangle \cdots \langle v_n = x_n\rangle\}$ that satisfies all constraints. A constraint optimization problem is a tuple $(V, D, C, f)$, where $(V, C, D)$ is a CSP and $f$ is a function which maps solutions to the CSP to a numeric value. The goal of a COP is to find a solution to the CSP with an optimal (minimal or maximal) value for $f$ [18].

Algorithms for solving constraint optimization and satisfaction problems can be divided into two main groups [36]: constructive or systematic search which systematically searches through the space of all solutions and local search methods which work from a full assignment of variables and then make small, local, repairs to move towards a closer to optimal or satisfying solution.

In constructive search, the search space of all possible solutions is systematically and completely explored. This is usually accomplished through an incremental adding of decisions to a partial solution until a full solution is constructed or a conflict is detected. Once a conflict is detected, decisions are removed (backtracking), and others are asserted in a systematic way that ensures every possible combination of values to variables is implicitly or explicitly explored. One main problem encountered in constructive search is that wrong decisions made early in the procedure may cause the algorithm to fruitlessly keep searching in a branch of the search tree where no solution can occur, repeatedly finding the same conflicts (thrashing). Improvements to constructive search can be categorized in two main areas, look-ahead and look-back techniques. Look-ahead techniques attempt to improve the area yet to be explored through filtering techniques that can prune areas of the search space which can be proved to have no solution [44, 55], and heuristics that order which value or variable to assign next [31]. Look-back techniques attempt to exploit the area already searched by jumping back to the cause of a dead-end [63], or learning the reason for the current conflict and adding it as a new constraint [23]. For a survey of the systematic constraint satisfaction techniques see Kumar [39] or Dechter [18].

Local search techniques are free from the structural confines of constructive search, with the freedom of making any move which can lead to a better solution. But with this freedom, one usually loses the completeness of constructive search. There is also no guarantee that a local search procedure will ever find a solution in a finite amount of time. In situations where there are no solutions, local search would just keep searching forever. Local search algorithms are also at risk of becoming stuck in local optima. Local optima are sub-optimal states of the search space to which improving moves can move to, and no improving moves can leave. Local search techniques use a variety of meta-heuristics in order to avoid and escape from these local optima. A good overview of the variety of local search algorithms is provided by Hoos and Stüzle [36].

### 2.1.1 Job-Shop Scheduling

In this section, we provide an example of a constraint optimization problem, job-shop scheduling, and how it can be solved with both constructive and local search algorithms.

An $n \times m$ job-shop scheduling problem (JSP) contains $n$ jobs each composed of $m$ completely ordered activities. Each activity, $a_i$, has a predefined duration, $d_i$, and a resource, $r_i$, that it must have unique use of during its duration. There are also $m$ resources and each activity in a job requires a different resource. The processing of activity on a machine is called an *operation*, and denoted $o_{ij}$ for an activity of job $i$ running on machine $j$.

A solution $s$ to a JSP instance specifies, for each machine, the ordering which the activities of each job are processed by that machine. This ordering also implicitly defines the earliest start time $est(x)$ and earliest completion time $ect(x)$ for each operation $x$ [70]. The usual goal of JSP optimization is to find a solution with a minimal makespan. The makespan $C_{max}(s)$ of a solution $s$ is the maximum earliest completion time of any operation.

A solution feature used by many search techniques is the *critical path*. A critical path is a sequence of operations of a solution which defines its makespan. Specifically, a critical path of a solution $s$ is a sequence of operations $o_1, o_2, \cdots, o_l$ such that (1) $est(o_1) = 0$, (2) $ect(o_l) = C_{max}$, and (3) $est(o_i) = ect(o_{i-1})$ for $1 \geq i \geq l$, where $ect(o_0) = 0$ [70]. A solution may have more than one critical path. The related term *critical block* refers to a subsequence of operations on a critical path which all run on the same machine. Figure 2.1 displays a solution to a $10 \times 10$ JSP with its sole critical path, composed of seven critical blocks, highlighted.

To solve a JSP with constructive search, a partial solution can be built up by using the orderings of activities on each resource as the *decision variables*, and assigning values to them

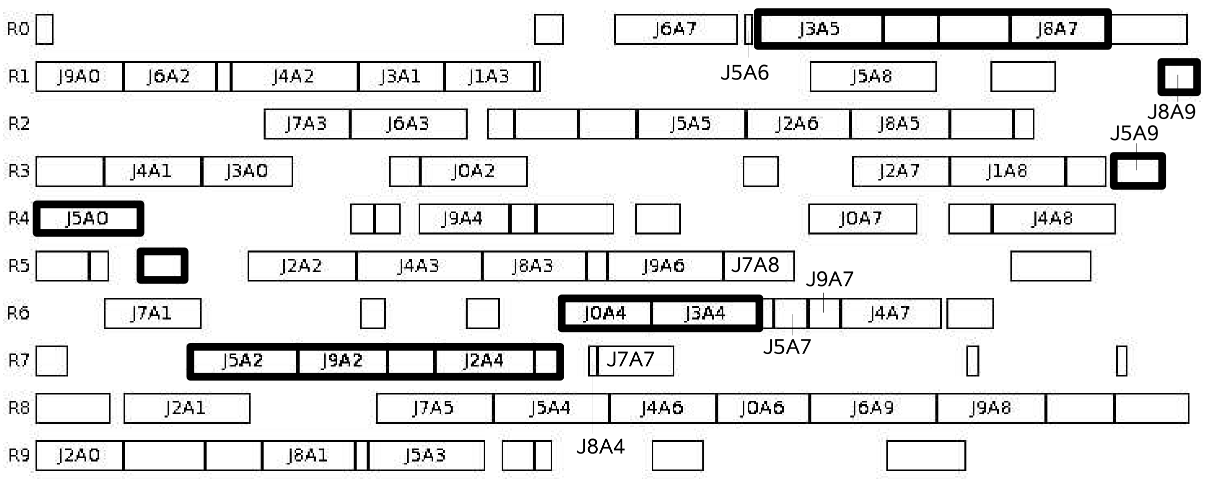

Figure 2.1: A $10 \times 10$ JSP example with the critical path highlighted by the thick borders. $J_iA_p$ represents $p^{th}$ operation of job $i$.

one at a time. Decision variables refer to variables that define the search space a constructive algorithm explores; a full, satisfying, assignment to these variables defines a solution to the problem. Between asserting assignments to these variables, look ahead techniques can be used to find new start time constraints and orderings that are implied by the current partial solution. These new constraints can then lead, or propagate, to the implication of more constraints. A variety of advanced look-ahead, or propagation techniques as they are called, exist for job-shop scheduling [51, 41, 42]. If the propagation techniques ever remove all possible values from the domain of a variable, or a constraint becomes unsatisfied, a *fail* occurs. At this time, a decision—usually the last one made—is removed, along with all propagated constraints caused by it, and another decision is asserted. Once a solution—a full instantiation of the variables that satisfies all constraints—is found, the bounds on when the last activity can end are set to one less than the makespan of the solution found and search continues. Eventually a solution with the optimal makespan will be found, and the bounds will be set to one less than this. The constructive search algorithm can then exhaustively explore the search space with this added constraint and prove that no solution with such a low makespan exists, and that the best makespan found so far is optimal.

In local search algorithms for job-shop scheduling optimization, search begins with a randomized ordering of activities on each resource that obeys all ordering and resource constraints on the activities. Such a random solution is very likely to have a sub-optimal makespan. Then with each iteration, a move, or repair, is made by altering this ordering. A usual technique [65, 49] is to take two activities on a critical block, as described earlier, and invert their order. One state-of-the-art local search algorithms for the JSP, TSAB [49], is described in depth in Section 2.3.3.2.

## 2.2 Solution Guided Multi Point Constructive Search

The topic of this dissertation, Solution Guided Multi Point Constructive Search (SGMPCS), is a recent CSP solving algorithm which attempts to incorporate ideas and methods from local search within a constructive search framework. SGMPCS consists of a sequence of many resource limited, randomized backtracking search iterations. Within each iteration a pool of the best solutions encountered so far during the search is maintained. With every iteration, a solution may be taken from this elite pool and used to guide the search. The initial studies of SGMPCS [5, 6, 7] have found it to outperform standard backtracking and randomized restart (see Section 2.3.1) on both job shop scheduling optimization and the quasi-group with holes completion constraint satisfaction problem.

Pseudocode for the basic SGMPCS 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, 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.

**Elite Solution Initialization** The elite solutions can be initialized by any search technique. The search effort is limited by a maximum number of fails for each run. As elite set diversity has been found to be important to metaheuristics [68], independent initialization runs are done to ensure that the initial solutions are diverse.

**Limiting Search** Each individual search is bounded by an evolving fail bound: a single search (lines 5 and 10) will terminate, returning the best solution encountered, after it has failed the corresponding number of times. How this bound grows over time to make the search complete depends on the fail sequence used (see Section 3.1).

**Searching From An Empty Solution** With some probability, $p$, search is started from an empty solution (line 5). Searching from an empty solution simply means using any standard constructive search with a randomized heuristic and a bound on the number of fails. Initially the $p$ parameter was used in hopes of helping to diversify the search process, but was later shown to do the opposite (see Section 2.2.2). The probability of starting from scratch can still be seen as a way to parameterize the extent to which SGMPCS is guided by elite solutions as opposed to simply being a randomized restart method.

**Setting Bounds on the Cost Function** To exploit constraint propagation an upper bound of the cost function must be set at each iteration. Beck [6] defines two possible techniques: local and global bounding. In global bounding, the upper bound on search is always set to one less than the lowest cost found so far. For local bounding, the bound depends on whether search is being guided by an elite solution. When search is started from an empty solution, the upper

**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 fail limit, $b$
5          $s$ := search($\emptyset$, $b$)
6          if $s$ is better than worst($e$) then
7              replace worst($e$) with $s$
       else
8          $r$ := randomly chosen element of $e$
9          set fail limit, $b$
10         $s$ := search($r$, $b$)
11         if $s$ is better than $r$ then
12             replace $r$ with $s$
```

bound on the cost function is set to one less than the cost of the worst current elite solution. When being guided by a solution, the upper bound is set to one less than the cost of the guiding solution.

### 2.2.1 Searching from a Solution

To start constructive search from an elite solution, a search tree is created 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, but possibly infeasible; 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 $x \in dom(V_i)$.

### 2.2.2 Adapting Elite Solutions to Satisfaction Problems

In an optimization context, a solution can be defined as a complete, feasible assignment of all variables. Solutions were compared based on their corresponding objective value or cost. To adapt SGMPCS for satisfaction, the need for a complete assignment is relaxed and solutions are compared 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 a dead-end is encountered, no attempt is made to further assign variables: as soon as there is a domain wipe-out, the cost of the partial solution is evaluated by counting the number of unassigned variables and then search backtracks, provided the resource limit on the search has not been reached. There are two main reasons for using this evaluation of partial solutions. Firstly, it is a very simple method that should be tried before anything more complex. The second is the intuition that partial solutions with more assignments are closer to being a complete assignment and are therefore should be preferred

### 2.2.3 SGMPCS as Local Search

While the core search technique in SGMPCS is heuristic tree search, there are two ways in which SGMPCS can be viewed as a hybrid of constructive and local search. First, SGMPCS is based on a fundamental idea of local search: the use of sub-optimal solutions to guide search. In fact, as mentioned in [7], one of the main motivations for SGMPCS was the local search algorithm TSAB [49] (see Section 2.3.3.2) which maintains multiple viewpoints, in the form of sub-optimal solutions, to guide search. Second, and more crucially, a single iteration of SGMPCS 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 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.

### 2.2.4 Past Empirical Studies of SGMPCS

Past studies have empirically studied the performance of SGMPCS on job shop scheduling problems [7, 6]. One interesting finding from [6] was that SGMPCS with smaller elite sets performed better on JSP optimization, with an elite set of one performing the best. This brought into question one of the main motivations of SGMPCS, that of maintaining multiple viewpoints during search. Low probabilities of starting from an empty solution ($p$) were also found to perform better. In a later empirical study [7], it was found that the $p$ value was having the opposite than expected effect on elite pool diversity. Because of the differing replacement rules, repeatedly starting from scratch had the effect of filling the elite pool with similar solutions. Further experiments controlling for different expected diversity levels found SGMPCS performed best with replacement rules that reduced the diversity of the elite set. This trend of better performance for lower diversity goes against intuitions and experience in local search [71].

## 2.3 Related Algorithms

### 2.3.1 Randomized Restart Search

SGMPCS is similar to the randomized restart techniques proposed by Gomes et al. [26] in which a randomized backtracking search is stopped if a solution is not found within a certain resource limit and then restarted with a new random seed. This technique came out of studies of heavy-tailed distributions in the run-time of backtracking search (see Section 2.4.2 for an overview of the empirical studies into heavy tails). By stopping search at a given limit, one avoids the non-trivial probability that the given stochastic run will be extremely long. Also, by using a small limit on search, one can exploit a related phenomena of heavy left hand tails: the non-trivial probability of the next stochastic run being very quick. Randomized restart search has been shown to be able to outperform standard chronological backtracking on various constraint satisfaction problems [26]. Randomized restarts have also been implemented in SAT solvers to boost performance on many practical problems such as planning and hardware verification [48].

Luby et al. [43] showed that the optimal restart limit for these type of restart algorithms is the value $r$ that optimizes Equation 2.1. $E(r)$ is the expected run-time of such a restart technique with a fixed restart limit $r$, where $q(t)$ is the probability the next run will take less than $t$ units of time (or units of search cost, when another measure of search effort is being used). To obtain this value, complete information of the run-time distribution of the randomized search is needed.

$$E(r) = \frac{r - \sum_{t<r} q(t)}{q(r)} \tag{2.1}$$

For the situation where there is no information of this distribution, Luby et al. also formulates a universal sequence $S^{\text{univ}}$ of limits which is only a log factor away from the true optimal fixed limit, and only a constant factor away from any other universal sequence. The universal sequence proceeds as follows:

$$S^{\text{univ}} = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \cdots)$$

Formally, $S^{\text{univ}} = (t_1, t_2, t_3, \cdots)$, where

$$t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1; \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \leq i \leq 2^k - 1. \end{cases} \tag{2.2}$$

### 2.3.2 Constructive Search in the Area of Good Solutions

A number of other algorithms implement a similar form of iterative randomized construction of solutions combined with some way of using the solutions found in past iterations to help guide search.

#### 2.3.2.1 Adaptive Probing

The Adaptive Probing algorithm of Ruml [56] consists of repeated probes into the search space. Each probe consist of constructing a solution using a randomized heuristic which is also a function of a weight attributed to each decision. At the end of a probe, the weights for each decision are updated under the assumption that the cost of the solution is based on an additive model of all decisions, or components, that made up that solution. As it was applied to satisfaction problems, Adaptive Probing used the same technique as SGMPCS of using the number of unassigned variables when a conflict was detected as the cost of the dead-end. Pseudocode for Adaptive Probing is shown in Algorithm 2. Search starts by first giving equal weight to all solution components. With each iteration, a new solution $s$ is constructed using a randomized heuristic that is biased by the weights $w$. The weights are then updated based on the solution quality of $s$ and the cycle repeats. Ruml [56] found that this algorithm can be competitive with systematic techniques, and can outperform them in the case where the heuristic makes many mistakes. We mention some further empirical studies involving Adaptive Probing in Section 2.4.3.

**Algorithm 2:** Adaptive Probing [56]

```
  AP():
1 initialize weights $w$
2 while termination criteria unmet do
3     $s$:= construct($w$)
4     $w$:= adaptWeights($s$,$w$)
```

### 2.3.2.2 Ant Colony Optimization

The development of the ant colony optimization (ACO) algorithm was inspired by the collective behaviour of ants which can find the shortest path to a food source through the use of distributed local communication in the form of pheromone trails [20]. In ACO, a population of artificial ants are created, and each independently, and stochastically, builds a solution to the problem. After each ant finds a solution they deposit *pheromone trails* along the components of their solution (the amount of pheromone usually depends on the quality of solution found). When the next wave of ants search, the randomized choices they take in building a solution are influenced by the pheromone deposited earlier. To keep the search process from stagnating, *pheromone evaporation* is employed so that the pheromone deposited on each component decreases with time. Reduced to its basic form, ACO can be seen to be quite similar to Adaptive Probing [36]. The difference being that with each step a population of $k$ new solutions is generated. These $k$ solutions are then taken together to update the weights used in the randomized heuristic.

As shown in Algorithm 3, the weights, or trails, $\tau$, are first initialized to some base level. $sp$ denotes the population of solutions: the results of each ant's search. With each iteration, each ant creates a new solution based in part by the current trails $\tau$. The trails are then updated based on the solutions found by each ant, and reduced due to evaporation, and the cycle repeats. ACO was first applied to the Traveling Salesperson Problem, and while results were somewhat promising, it is outperformed by other state-of-the-art techniques. One area where more recent Ant Colony algorithms are prominent is in dynamic problems. A dynamic problem is one which can change while solving it, such as telecommunications routing [19].

**Algorithm 3:** Ant Colony Optimization algorithm.

```
   ACO():
1  initialize trails $\tau$
2  $sp$ := {}
3  while termination criteria unmet do
4      sp:= constructPopulation($\tau$)
5      w:= updateTrails($sp$,$\tau$)
```

#### 2.3.2.3 Large Neighborhood Search

Large Neighborhood Search (LNS) [53, 59] is another search algorithm which combines constructive and local search techniques to find optimal, or high quality solutions to constraint optimization problems. LNS tries to iteratively find a better solution than it currently has by freezing a set of variables to the values they had in the last best solution and applying constructive search to the rest of the problem with an updated optimization bound and resource limit on search. Pseudocode for the basic architecture of LNS appears in Algorithm 4. At line 1, a solution $s$ is found through a standard constructive search technique, then a while loop is entered to attempt to improve the quality of the solution. With each iteration, a neighborhood $N$ is selected. This represents the variables that will be searched through to find a better solution. At line 4, a new solution $s'$, is found by freezing the values of variables not in $N$ to their values in the last solution $s$, and applying constructive search on the rest. The best solution so far, $s$, is updated, and the cycle continues until a solution of adequate quality is found, or some other termination criteria are met.

A difficult problem in LNS is choosing the large neighborhood to search through at each iteration. While there are more generic methods for defining these neighborhoods at each iteration [53], most implementations use problem specific methods [59]. Randomly chosen neighborhoods have been found to not to perform well [59]. If SGMPCS always guides from an elite solution ($p = 0$), and has an elite size of 1, it can be seen as quite similar to LNS with random neighborhoods. In SGMPCS, the variables high in the tree will be fixed to the value in the guiding solution. If $|e| = 1$, the guiding solution will always be the last best solution found, as in LNS. The variables which the randomized variable ordering places at the bottom of the search tree defines the neighborhood SGMPCS searches through at that iteration. SGMPCS typically uses a fail sequence that grows with time in order to be complete. An analogous increasing in the size of the neighborhoods is not typically used in LNS.

**Algorithm 4:** Large Neighborhood Search [53].

```
   LNS():
1  initialize initial solution $s$
2  while termination criteria unmet do
3      Select Neighborhood $N$
4      $s'$ = Constructive Search, Freezing variables $\notin N$ to values in $s$
5      if $s'$ is better than $s$ then
6          $s = s'$
```

### 2.3.3 Multiple Viewpoints

Other than ant colony optimization, most population-based techniques have been used in local search. Two examples of this are genetic algorithms and the tabu search algorithm TSAB which was one of the main inspirations for SGMPCS [7].

#### 2.3.3.1 Genetic Algorithms

In genetic algorithms (GAs), search proceeds through a population of solutions that *evolve* over time [34, 24]. A genetic algorithm (pseudocode shown in Algorithm 5) starts by generating an initial random set of candidate solutions. Each member of the population is then evaluated and assigned a cost, or as the inverse is called in GA, a fitness. Some subset of the population with the highest fitness then *reproduce*. This is accomplished through cross-over operators that combine components of two solutions together into a new solution, and by mutation which takes a solution and randomly changes some of its components. After reproduction the older generation dies—in some algorithms letting higher fitness members survive—and are removed from the population. Pseudocode for GA is shown in Algorithm 5. At line 1, a population of candidate solutions $sp$ is generated through some randomized means. At line 3, fitness values are assigned to each solution through some evaluation function. At line 4, new solutions $sp'$ are generated by mutating and recombining the fitter solutions from $sp$. At line 5, older and less fit solutions are removed from the population and the process continues until some termination criteria are met.

Many cross-over techniques exist for combining old solutions into new ones. The most common is the one-point crossover [36]. In this technique, as in most, solutions are represented as strings of components. In the one point cross over, two solutions are split into four at some random point on each solution, and recombined by swapping parts into two new solutions. One problem with most cross-over techniques is ensuring that the new recombined solutions are still well-formed. This can necessitate problem specific cross-over operators, or local search repair [46].

**Algorithm 5:** Genetic Algorithm.

```
GA():
1 initialize initial population $sp$
2 while termination criteria unmet do
3     assignFitness($sp$)
4     $sp'$ := reproduceFit($sp$)
5     $sp$ := $sp'$ + surviving($sp$)
```

#### 2.3.3.2 TSAB

Taboo Search Algorithm with Back Jump Tracking (TSAB) is a tabu search algorithm for job-shop optimization by Nowicki and Smutnicki [49]. In traditional tabu search, as local search progresses by making local moves that improve the quality of a solution, a tabu list is used as a type of short-term memory to keep track of the recent history of the search. The tabu list consists either of a list of the recent moves or states the search has recently made or visited, or as some collection of attributes of the recent few moves or states. When choosing the next move from the neighborhood of the current solution, any moves or states which match with the tabu list are forbidden. An aspiration criterion lets the algorithm overrule the tabu list if, for example, a forbidden move results in a solution with a cost lower than ever previously found. TSAB adds a long-term memory device to tabu search in the form of a list $S$ of the best $maxl$ solutions found so far. The solutions in $S$ are referred to as *elite solutions*, and $S$ is referred to as the *elite set*. When the tabu search procedure goes for a certain number of iterations without finding a better solution, search is restarted around one of the solutions in $S$. The tabu list is also updated at this time so the same moves away from that old position made earlier are not repeated.

Pseudocode of TSAB is shown in Algorithm 6. The algorithm first initializes an empty elite set $S$, and tabu list $T$. An initial solution $\pi$ is then generated. Search then progresses through two nested while loops. The inner loop (line 3) performs standard tabu search where search continuously moves to the best of the valid neighbors of $\pi$. The valid neighbors $N$ are defined by the current solution $\pi$, the aspiration criterion $a$, and the tabu list $T$ (line 6). As tabu search proceeds, the elite set $S$ of the best solutions found so far, and the tabu list $T$ of tabu recent moves/attributes are kept up-to-date. Once the tabu search has been found to stagnate—more than $maxiter$ iterations without finding a better solution—search is restarted around one of the elite solutions in $S$ (line 12). This is accomplished by setting $\pi$ to this solution, and resetting $T$ to its state when this solution was found with the original move made away from it also made tabu. The cycle then repeats until some final termination criteria are met.

TSAB was one of the best algorithms for solving job-shop scheduling problems. Subsequent techniques, that build on TSAB through more complex manipulations on the elite set, have resulted in even better algorithms [50].

**Algorithm 6:** Taboo Search Algorithm with Back Jump Tracking [49].

```
TSAB():
1  $S$:= {} ; $T$:= {}
2  initialize solution $\pi$
3  while termination criteria unmet do
4      iter = 0
5      while iter++ < maxiter do
6          $N$ = validNeighbours($\pi$,$T$,$a$)
7          $\pi$ = Best($N$)
8          update $T$
9          if $\pi$ is a new best solution then
10             iter =0
11             update $S$
12     Resume search around a solution in $S$
       reseting $T$ and $\pi$ appropriately
```

## 2.4 Empirical Analysis of Search Algorithms

Empirical analysis of search algorithms refers to the study of these algorithms through the use of reproducible computational experiments. From experiments, hypotheses about how and why an algorithm behaves the way it does can be made, and tested empirically through further experiments. The true value of empirical analysis in study of search algorithms may not have been always realized. Usually empirical results only appear near the end of papers to confirm what was already proved by the theory. Yet as argued by Hooker [35], as algorithms become more complex with many interacting components, empirical research of search algorithms becomes as necessary as it is to the study of any other complex system. In the next section, we provide an overview of various empirical studies applied to search algorithms for constraint satisfaction and optimization.

### 2.4.1 Problem Difficulty

One main topic of empirical research into satisfaction problems is in attempts to explain the wide variability in search cost for algorithms to solve randomly generated problem instances. One earlier finding was the easy-hard-easy pattern in search costs [47, 12, 21] as randomly generated instances go from being under-constrained to being over-constrained. In random instances with few constraints, each instances in this under-constrained area is most likely to be solvable. As more constraints are added randomly generated over-constrained instances become almost all unsolvable. Search cost for under-constrained problem is low since assigning almost any values to the variables is likely to be a solution. Search cost is also low for over-constrained instances since search can make use of all the constraints to quickly prove there is no solution. Where problems are the hardest is where they transition from being almost all solvable to almost all unsolvable. Cheeseman et al. [12] were able to observe this phase transition, and the associated easy-hard-easy pattern by adjusting one of the parameters in the instance generators for Hamiltonian circuit and graph coloring problems. Mitchel et al. [47] investigated this pattern in 3-SAT problems, satisfiability problems which consist of a conjunction of disjunctive clauses where each clause has three literals. By changing the clause to variable ratio ($c/v$) in randomly generated problems, a phase transition from nearing 100% satisfiable to 100% unsatisfiable is produced. The instances that were the hardest for a Davis-Putnam [16] systematic search algorithm to solve occurred where 50% of the problems are satisfiable, with a $c/v$ ratio of near 4.3. Gent et al. [21] developed a more generic parameter $\kappa$ to measure this *constrainedness*, where $\langle Sol \rangle$ is average number of solutions in a given population of instances, and $N$ is the number of bits needed to define the problem.

$$\kappa = 1 - \frac{log_2(\langle Sol \rangle)}{N} \tag{2.3}$$

$\kappa$ is bounded by the range $[0, \infty)$. An ensemble of instances—by definition $\kappa$ refers to an ensemble of instances—is under-constrained when $\kappa < 1$, and over-constrained when $\kappa > 1$. The phase transition, and critically constrained region, occurs near $\kappa \approx 1$. There has been a considerable amount of research into finding and explaining this phase transition [33, 61, 66, 72].

One motivation for past work on constrainedness was to find a way to consistently generate hard random instances [47, 72]. Some of the earlier benchmarks used to test various algorithms were found to be trivially solvable. By generating instances in the critically constrained region, there is more assurance that the problems will be consistently hard for most algorithms. Beyond just creating good random problem generators, constrainedness and phase transition research gives us a better understanding of the big picture of what makes hard problems hard. This research has also lead to new search techniques, such as the minimize $\kappa$ heuristic [21].

### 2.4.2 Heavy Tails

While most hard problems occur at the phase transition, some past studies [62, 22] have found that the under-constrained area contained some instances which were also exceptionally hard. Gomes et al. [26] used a partially randomized backtracking algorithm to solve these exact same instances. They found that with some random seeds, search took an exceptionally long time. Yet, with other random seeds, search became easy again. The original search algorithm just happened to make enough unlucky early decisions to make it take a long time to solve. As proposed by Gomes et al. [26], the cost of solving an instance can be measured as the distribution of search costs over all possible runs of a randomized algorithm. In certain cases, the distribution of search cost can be modeled by a distribution where all moments are infinite, which they refer to as a heavy-tailed distribution. For more information on the definition of heavy tails, how it is measured, and how a rapid restart technique removes them see Section 4.2.

Williams et al. [73] relate this heavy-tailed phenomenon with the back-door variables of an instance. The back door variables of an instance are a set of variables that once assigned make the rest of the problem solvable by a polynomial time algorithm. Randomized restart search can be seen as repeatedly trying to find this back-door set. Their theoretical model of backtracking search shows that the lower bounds on search cost can be modeled as a heavy tailed distribution when the back-door set is of a small enough size.

A later empirical investigation by Gomes et al [29] found some interesting results on where and why heavy-tailed behaviour occurs. For one, they found that heavy-tailed distributions disappear in randomly generated problems near the phase transition where all stochastic runs seem to become homogeneously hard for backtracking search. They were also able relate heavy tails to the thrashing behaviour of backtracking search by showing a correlation and mathematically proving a relation between the distribution of the depths of inconsistent sub-trees experienced during search and the appearance heavy-tails: exponentially distributed inconsistent sub-tree depth, along with exponential growth of the search tree as tree depth increases implies heavy tailed distributions in runtime.

### 2.4.3 Search Space Features of Local Search

A variety of empirical work has also been done in attempting to understand and explain local search behaviour by correlating performance with a variety of search space features such as: fitness-distance correlation [64], number of local optima [74], backbone size [52] and backbone fragility [60].

Local search primarily proceeds by making local moves to solutions that the algorithm evaluates to be better. So one would expect the performance of local search at finding optimal solutions would depend on how well this evaluation function does in predicting the true distance of the solution to an optimal solution. If moves to a *locally* better solution, also tends to move the search closer to the globally optimal solution, then search performance should be very strong. Because the first work investigating the correlation between the evaluated cost of a solution and its distance to an optimal solution was done in the area of genetic algorithms, it is given the term *fitness-distance correlation* (FDC). Early investigations on genetic algorithms found that FDC values could generally predict whether an instance would be easy or hard for GA to solve [64, 15]. Watson and Beck [10] investigated the relation between FDC and search performance for Adaptive Probing (AP) and Ant Colony Optimization (ACO). In an idealized problem where expected FDC values were controlled, they were able to see a strong effect of FDC on AP and ACO performance. Yet on the real job-shop scheduling problem, the correlation between the FDC of an instance and ACO search cost was weak.

Given only satisfiable instances, the easy-hard-easy pattern mentioned earlier can still be seen in constructive search algorithms. This pattern can be attributed to an algorithm's ability to use the many constraints in the over-constrained instances to find dead-ends early, and quickly prune down the search tree to the areas where a solution occurs. What may be surprising is that this easy-hard-easy pattern can also be seen with local search algorithms on satisfiable instances [74, 60]. Past empirical studies have looked into how different search space features correlate with local search cost in order to understand this pattern and to help gain deeper understanding of local search behaviour. Yokoo [74] sought to explain the diminishing search cost in the over-constrained region through the observed reduction in the number of local minima. Local minima are rarer in the over-constrained region, so a hill climbing procedure should have a higher likelihood of starting at a point that can reach a solution. Parkes [52] correlated problem difficulty with the backbone size of an instance in Boolean satisfiability problems. The backbone size of an instance is the proportion of variables that always have the same value in all solutions to the instance. When the backbone is large, all solutions will naturally form one cluster in the search space, making it hard for a local search procedure to find any solutions quickly. Singer et al. [60] introduced the concept of backbone fragility for SAT problems, which refers to how much smaller the backbone becomes as a clauses are removed from the problem instance. They relate this backbone fragility to the quasi-solution area of an instance, where a quasi-solution is defined as a solution to the original problem instance with 5 of its 100 clauses removed. Singer et al. propose that backbone fragility can be seen as a measure of how attractive the quasi-solution area is, since instances with small backbones are easy to solve. From its definition, backbone-fragility also measures the distance of this quasi-solution area to true solutions.

Watson et al. [70] performed a study correlating a variety of the search space features with the search cost of tabu search on job-shop scheduling optimization. They found the best predictor of performance was the average distance of a local optima to its nearest optimal solution. In later work [67], Watson showed that an even better predictor was the average distance between solutions encountered during search and optimal solutions. This led to an even more accurate dynamic model of tabu search as a random walk through a smaller subspace of a problem's search space induced by the algorithm's meta-heuristics. We return to Watson's search space features in Chapter 6.

## 2.5 Conclusion

The aim of this dissertation is to better understand SGMPCS behaviour and performance. Partly from the various algorithms and empirical analysis that influenced its development, Beck [7] hypothesized three main factors that influence the performance of SGMPCS: the benefit of maintaining a diverse set of multiple viewpoints, the benefit of guiding search in the area of past good solutions, and the exploitation of heavy tails in randomized backtracking. In the remainder of this dissertation, we investigate the last two of these claims through empirical analyses of SGMPCS.

# Chapter 3

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

As described in Chapter 2, Solution Guided Multi-Point Constructive Search (SGMPCS) is a recent constructive search algorithm for constraint optimization and satisfaction that borrows techniques from local search of maintaining multiple viewpoints and revisiting the space of good solutions. While previous results [4, 5] indicate that SGMPCS can significantly out-perform both standard chronological backtracking and randomized restart on optimization and satisfaction problems, the one existing systematic study of the parameters of SGMPCS [6] only addressed optimization problems. Beck [6] showed that SGMPCS significantly out-performs randomized restart and chronological backtracking on two different types of job-shop scheduling problem. 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 SGMPCS. The purpose of this chapter is to perform a similar systematic study of SGMPCS parameter values for constraint satisfaction.

In this chapter, we vary the primary parameters of the SGMPCS algorithm in a detailed set of experiments on three satisfaction problems: quasigroup-with-holes, magic square, and a satisfaction version of the multi-dimensional knapsack problem. The results confirm previous results in comparing SGMPCS with randomized restart and chronological backtracking on quasigroup-with-holes and indicate that maintaining more than one elite solution leads to stronger performance. Interestingly, experiments on magic square and a multi-dimensional knapsack problems show performance that is about the same as randomized restart. Both SGMPCS and randomized restart are significantly better than chronological backtracking for the magic square problems but significantly *worse* on the multi-dimensional knapsack problems. Further experimentation, changing the way that the elite solutions are compared by using the original multi-dimensional knapsack optimization function, shows that in solving the satisfaction problem with this extra information, SGMPCS exhibits a substantial increase in problem solving performance, and out-performs both chronological backtracking and randomized restart.

In the next section, we present the parameter space that will be investigated. We then turn to the empirical studies, varying the parameters on each of the three satisfaction problems.

Finally, we discuss our results and their implications in terms of developing an understanding of why and how SGMPCS works.

## 3.1 SGMPCS Parameter Space

There are a number of parameter values that must be specified in order to implement the SGMPCS algorithm. The main purpose of this chapter is to examine how the choice of each parameter value impacts performance on CSPs.

**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 chapter, we study $p = \{0, 0.25, 0.5, 0.75, 1\}$. Note that $p = 1$ is equivalent to randomized restart as it always searches from an empty solution.

**Elite Set Size** Previous studies of SGMPCS for satisfaction problems [4, 5] 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 chapter, we experiment with elite sizes of $\{1, 4, 8, 12, 16, 20\}$.

**Backtrack Method** For a single search, we have a choice as to how the tree search should be performed. That is, with each iteration of guided or non-guided search, we do not necessarily have to use chronological backtracking to explore the tree. In particular, we experiment with using chronological backtracking or limited discrepancy search (LDS) [32]. In both cases, the search is limited by the fail bound as described below.

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

- Luby - the fail bound sequence follows the optimal sequence when there is no knowledge about the solution distribution: 1,1,2,1,1,2,4,1,1,2,1,1,2,4,8, ... [43].

- Geometric (Geo) - the fail bound is initialized to 1 and reset to 1 when a new best solution is found. When a search fails to find a new best solution, the bound is doubled. In the satisfaction context, a new best solution is a solution with more assigned variables than any solution previously encountered.

- Polynomial (Poly) - the fail bound is initialized to 32 and reset to 32 whenever a new best solution is found. Whenever a search fails to find a new best solution, the bound grows polynomially: 32 is added to the fail limit. The value 32 was chosen to give a reasonable increase in the fail limit on each iteration.

**Initialization Fail Bound** There are two parameters associated with the initialization of the elite set: we need to decide how many solutions should be initially produced and we need to decide how much effort we should spend on producing each initial solution. It has been previously shown that generating $|e|$ initial solutions (i.e., one for each element of the elite set) can skew results that examine the impact of changing $|e|$ [6]. Therefore, we always create 20 initial solutions and then select the $|e|$ best for the initial elite set. The resource bound is simply the number of fails we allow to find each initial solution. The following initialization fail limits are tested in this chapter: 1,10,100,1000,10000.

## 3.2 Empirical Study

In the following experiments, a fully crossed experimental design is not implemented, as the focus is on how each parameter setting individually affects performance. For all but one of the experiments, one parameter is varied while the others are set to their default values. Based on preliminary experimentation and previous studies the values shown in Table 3.1 are used as defaults.

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

Table 3.1: Default parameter values for the experiments.

### 3.2.1 Problems

The experiments are performed on three different satisfaction problems: quasigroup-with-holes, magic square, and multi-dimensional knapsack. These problems were chosen because benchmark sets exist and randomized restart shows an interesting pattern of performance: performing well on quasigroup problems [30] and poorly on multi-dimensional knapsack [54]. Magic square problems are similar in form to quasigroup-with-holes and so may present an interesting variation. We focus on problems with interesting performance of randomized restart because of the similarity between randomized restart and SGMPCS. SGMPCS can be interpreted as a form of guided randomized restart and so one of the questions we are interested in is if SGMPCS is simply exploiting the heavy-tails phenomenon like randomized restart [30].

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 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. This problem is modeled by all-different constraints with extended propagation [55] placed on each row and column. Two sets of problem instances are used. (i) 100 order-30 instances divided into ten subsets according to the number of holes: $m \in \{315, 320, .., 360\}$. These instances are created using an existing generator [1] and were used previously to study SGMPCS [4]. (ii) A set of existing benchmark instances [30, 54]. In all QWH experiments, each instance is solved ten times with different random seeds and a fail limit of 2,000,000 fails for each run.

- *Magic Square* 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 and a limit of 10,000,000 fails.

- *Multi-Dimensional Knapsack* Given a knapsack with $m$ dimensions such that each dimension has capacity, $c_1, \dots, c_m$, 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 $j = 1, \dots, m$, 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 [54]. Two sets of six problems are used from the operations research library[1] which have 15 to 50 variables and 2 to 30 dimensions. For each problem, results were averaged over 20 independent runs with different random seeds and a limit of 10,000,000 fails.

### 3.2.2 Experimental Details

Each of the problem types and search methods used a minimum domain variable ordering with randomization on ties. When variables are binary, as in the multi-dimensional knapsack problem, the variable ordering is completetly random. The value ordering for each problem and technique, when not being guided by an elite solution, is also random. All algorithms were implemented in ILOG Solver 6.0 and run on a 2.8GHz Pentium 4 with 512Mb RAM running either Fedora Core 2 or Red Hat Enterprise 4.

### 3.2.3 Quasigroup-with-Holes Experiments

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

#### 3.2.3.1 Initial QWH Experiments

**The Probability of Searching from an Empty Solution** The $p$ parameter is the probability that search will be done from an empty solution (i.e., a standard restart) vs. being guided with an elite solution. Figure 3.1 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 [6] where it was shown that $p = 0.25$ delivered the best performance with monotonically decreasing performance for $p \geq 0.5$ and with $p = 0$ performing worse than $p = 0.25$.

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

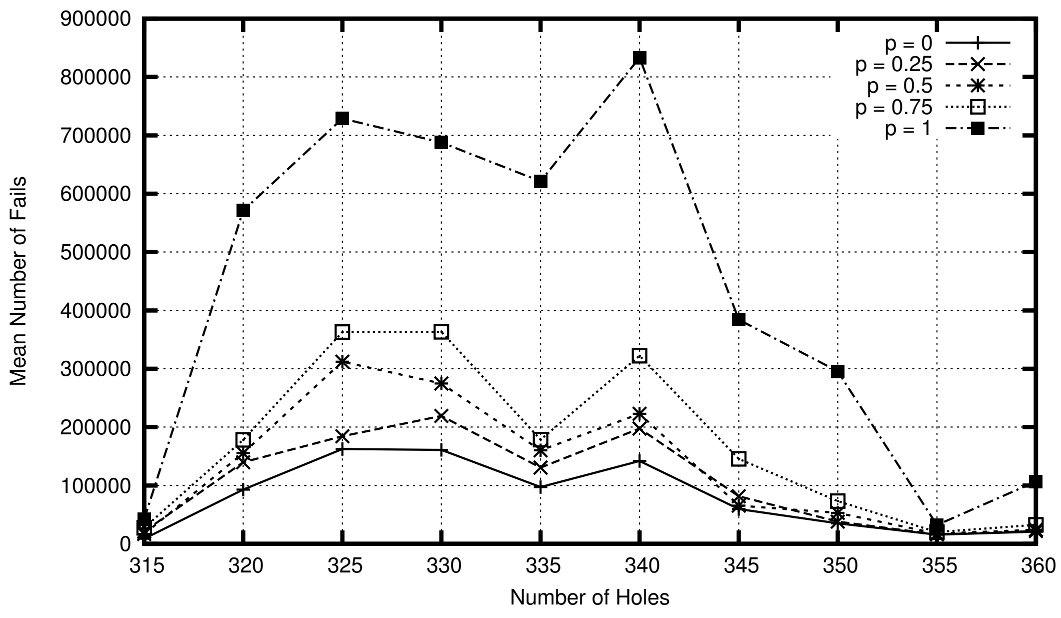

Figure 3.1: Mean number of fails to solve order-30 QWH problems in each subset for varying $p$-values.

**Elite Set Size** The results for varying $|e|$, the number of elite solutions maintained, are shown in Figure 3.2. In contrast to previous results on scheduling problems [6], 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 $|e| = 8$. However there appear to be only small differences for all $|e| > 1$.

**The Interaction Between $|e|$ and $p$** Maintaining more than one elite solution and starting from scratch with some probability were both included in SGMPCS to add diversity to the search process. To examine possible interactions between the size of the elite set and the probability of searching from an empty solution, a full cross of these two parameters is done. Figure 3.3 shows the mean results of all 1000 runs over all 100 problems for a given setting of $p$ and $|e|$. The results again demonstrate that a lower probability of starting from an empty solution performs better, and a elite size of 1 performs poorly. While no clear evidence of an interaction exists between settings of these two parameters, the combination $|e| = 8$ and $p = 0$ does perform the best. As mentioned in Section 2.2.4, it was found in [7] that higher $p$ values, contrary to expectations, actually decreased diversity of the elite set. So both the poor performance of $|e| = 1$ and the better performance of lower $p$ values support the need for diversity in the search. Yet this is contrary to the same study on scheduling optimization problems [7] which found techniques that further reduced diversity performed better.

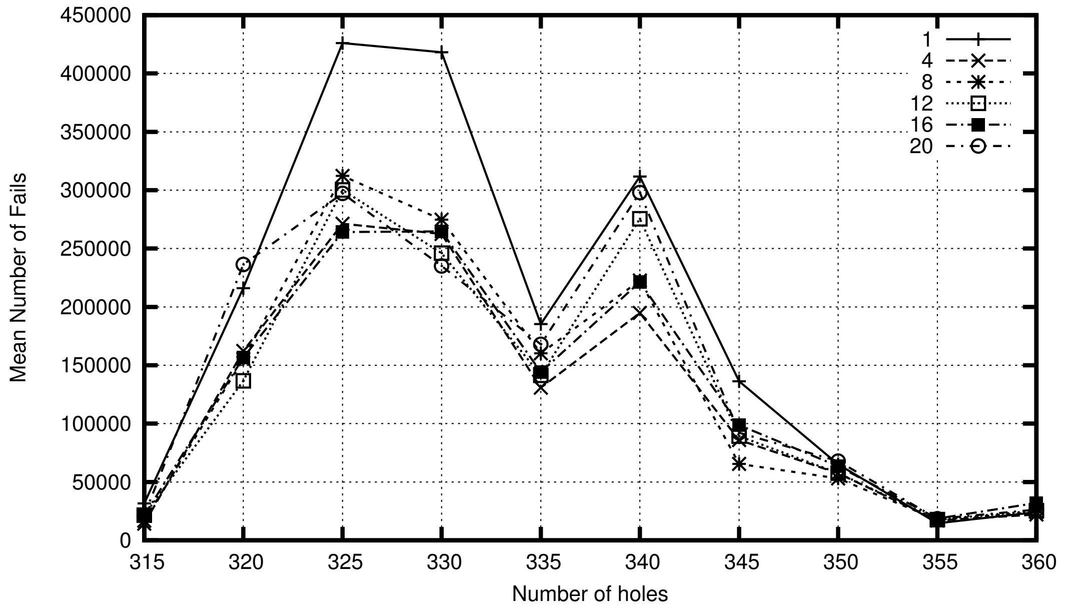

Figure 3.2: Mean number of fails to solve order-30 QWH problems in each subset for varying values of $|e|$.

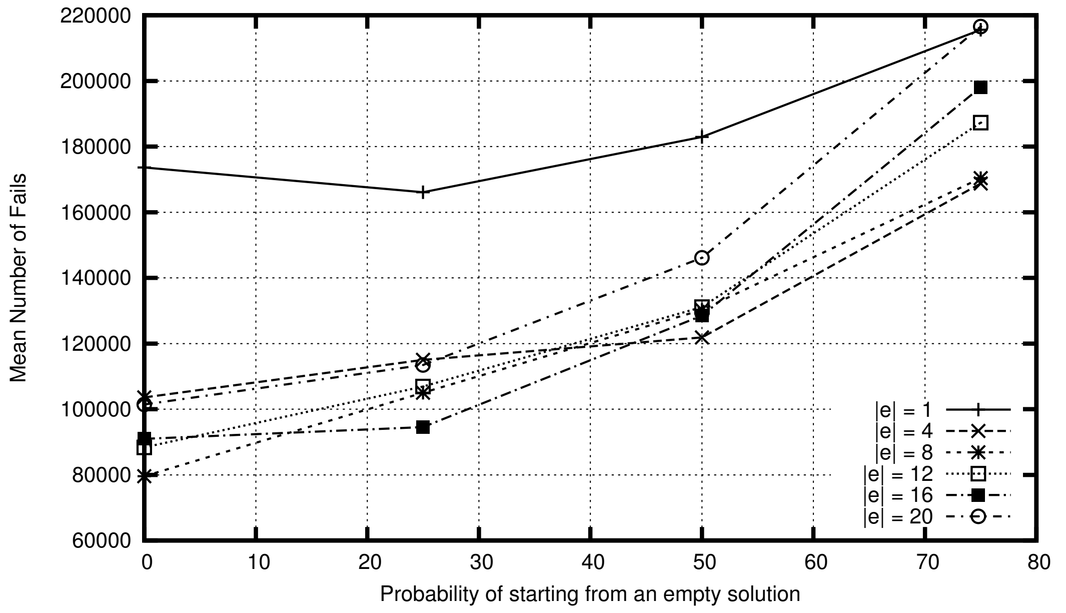

Figure 3.3: Mean number of fails to solve order-30 QWH problems for all values of $|e|$ and $p$.

**Backtrack Method** Next, we examine the impact of using chronological backtracking or LDS for each individual search. Figure 3.4 shows that, in terms of the number of fails, LDS results in better performance.

However, as shown in Figure 3.5, LDS takes significantly longer to solve the problems even though it incurs fewer fails.[2] Our intuitive explanation for these results rests on the behaviour of LDS. In chronological backtracking, much of the search is at the bottom of the search tree. Therefore, a single fail will tend to result in few additional choice points. In contrast, a single fail in LDS will more often result in a large “jump” requiring many additional choice points. The computational effort of the choice points high in the tree is not amortized over as many fails. Apparently, on these QWH problems, while LDS finds a solution in fewer fails, the improvement is not sufficient to make up for the greater computational time per fail.

**Fail Sequence** Figures 3.6 and 3.7 show the results of running SGMPCS with each fail sequence in terms of mean number of fails and mean solve time, respectively. As was observed with LDS, the time per fail is much higher in the Luby fail sequence. While the Luby sequence results in the lowest number of fails, it also exhibits some of the highest run-times. We believe this can be attributed to the relatively slow growth of the Luby sequence: it has a relatively large number of low fail limits which result in more complete restarts that the other techniques. The geometric sequence shows poor performance in both measures.

[2]For all our other experimental results, unless specifically mentioned, the relative performance of methods based on the mean number of fails is identical to the comparison based on mean run-time.

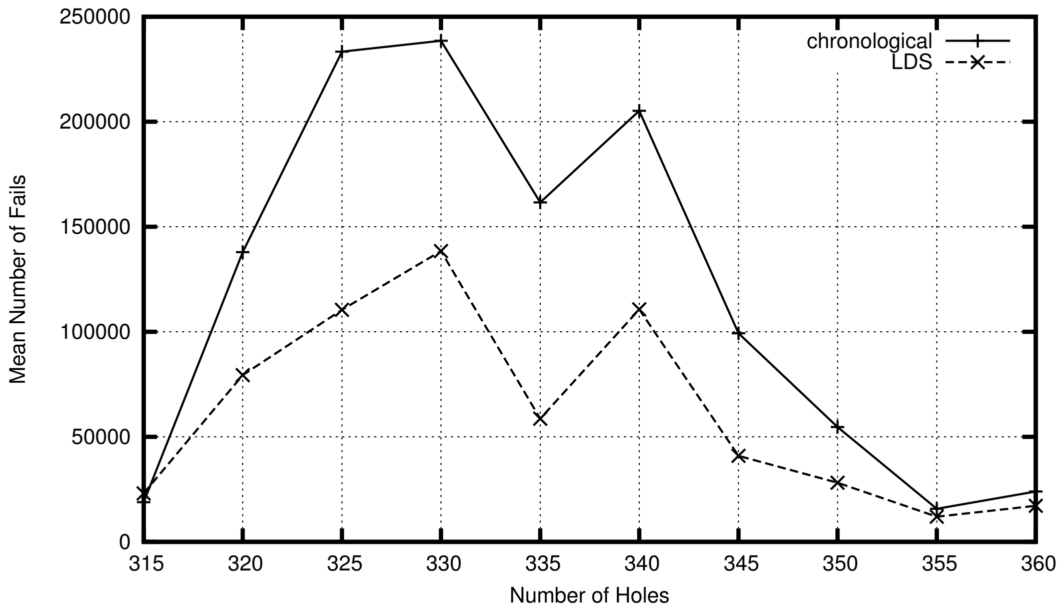

Figure 3.4: Mean number of fails to solve order-30 QWH problems for both backtracking methods.

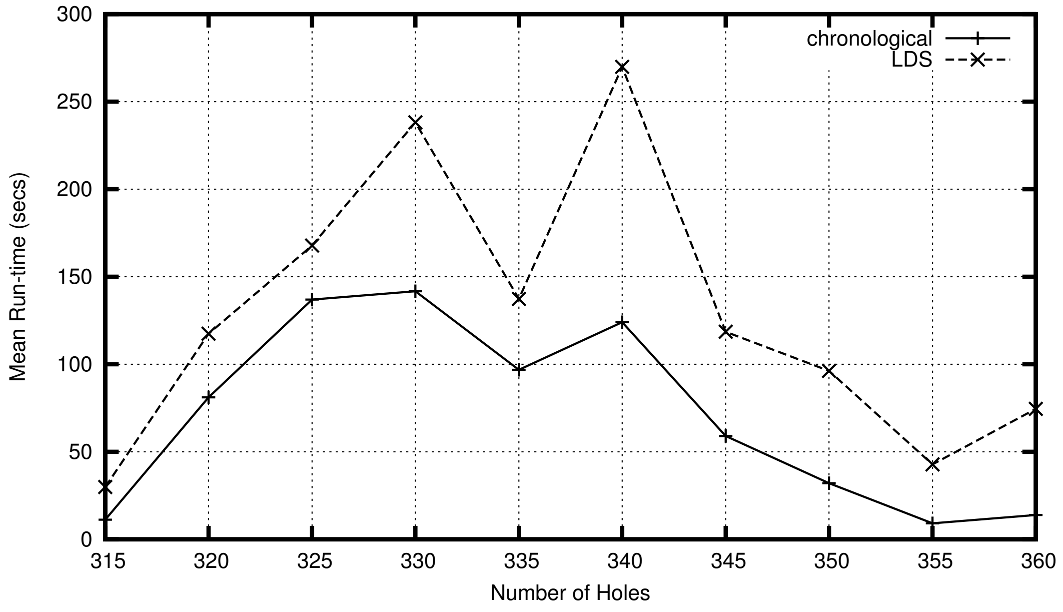

Figure 3.5: Mean time to solve order-30 QWH problems for both backtracking methods.

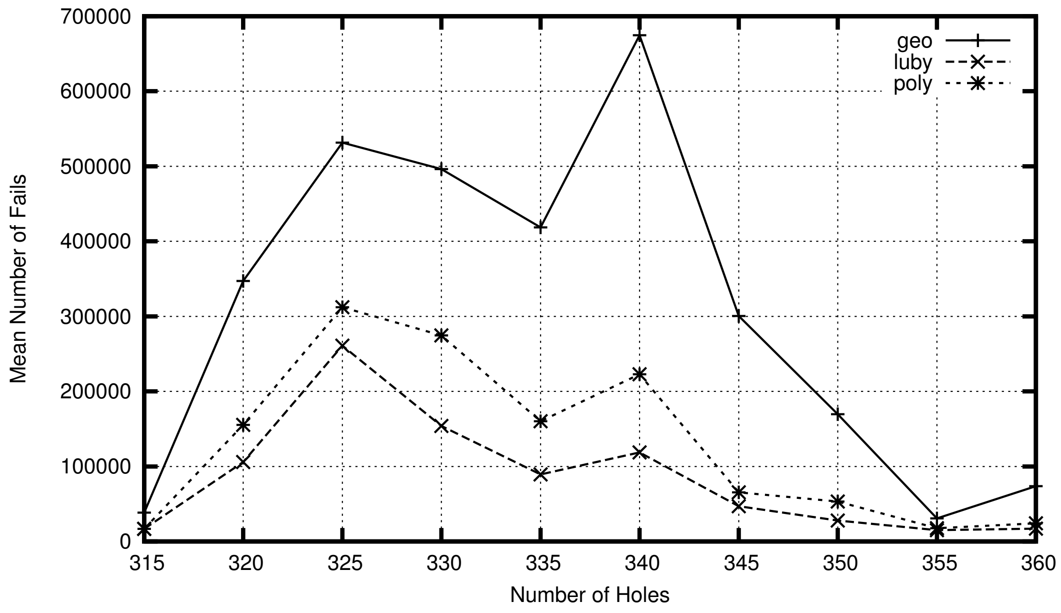

Figure 3.6: Mean number of fails to solve order-30 QWH problems in each subset for each fail sequence.

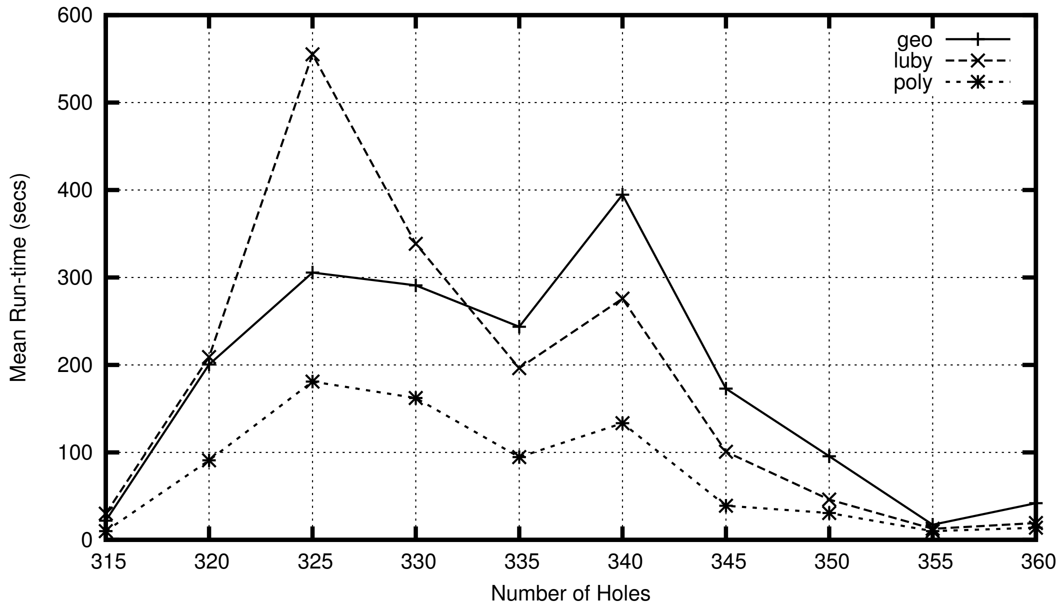

Figure 3.7: Mean time to solve order-30 QWH problems in each subset for each fail sequence.

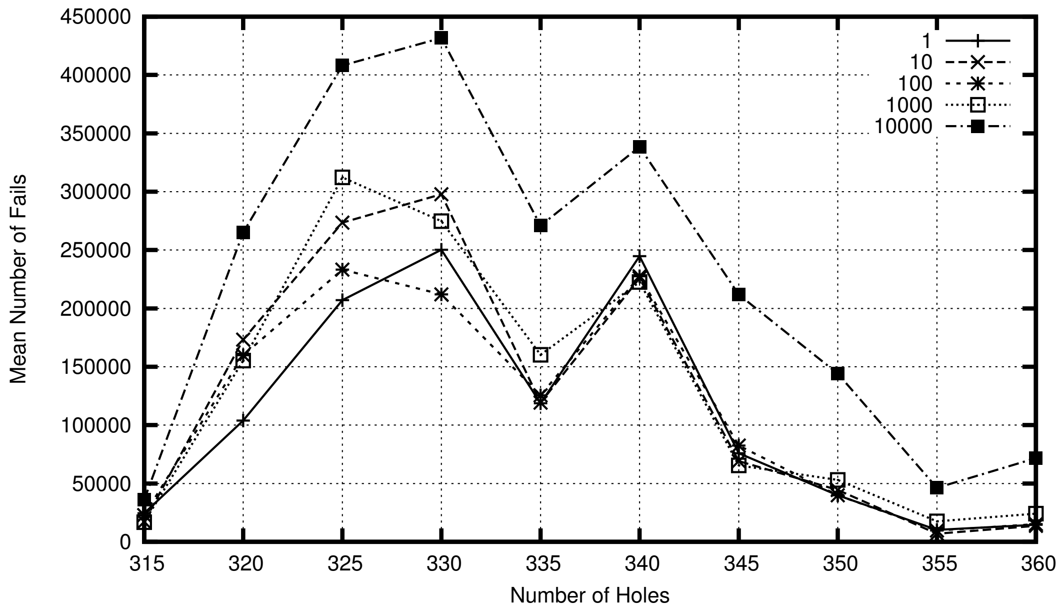

Figure 3.8: Mean number of fails to solve order-30 QWH problems for varying initialization bounds.

**Initialization Bound** Finally, the five different bounds on the initialization of the elite set are examined: 1, 10, 100, 1000, 10000. Figure 3.8 shows that initialization bounds other than 10,000 have little effect on overall performance. With an initialization limit of 10,000, the 200,000 fails used to build the elite set are about equal to the total number of fails needed to find a solution when smaller initialization limits are used.

**Summary** Overall, these experiments indicate that, for the QWH problem, the following parameter values perform best: $|e| = 8, p = 0$, chronological backtracking, the polynomial fail sequence, and an initialization fail bound of 100.

#### 3.2.3.2 Benchmark QWH Experiments

Using the best parameter settings from the QWH experiments above, we apply SGMPCS to an existing set of QWH benchmarks and compare the performance with standard chronological backtracking (*chron*), randomized restart (*restart*), and results from the literature. 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 SGMPCS methods and initializes and maintains a set of elite solutions. However, it always searches from an empty solution (i.e., it is equivalent of SGMPCS with $p = 1$). Therefore, it has a small run-time overhead to maintain the elite set as compared with standard randomized restart. This overhead does not effect the number of fails.

Each problem was run 10 times for each algorithm. The percentage 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 3.2. 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.

| | | chron | | | restart | | | SGMPCS-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 3.2: QWH benchmark comparison with other search algorithms.

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

Table 3.3: Summary Statistics for Table 3.2.

| | | SGMPCS | | 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 | - | - |
| 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 | - | - |
| 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 3.4: QWH comparison with Impact Based Search [54].

The pattern of bold entries in Table 3.2 is summarized in Table 3.3. SGMPCS 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.

Table 3.4 compares our results with previous [54] results on the same benchmark instances 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 SGMPCS. A direct comparison is complicated by the fact that [54] 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 SGMPCS (as shown by the bold entries), SGMPCS clearly beats it on some of the hardest problems, and is able to reliably solve at least two more instances. However, SGMPCS 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 and value ordering heuristic within SGMPCS, it would be interesting to look at combining these approaches.

### 3.2.4 Magic Square Experiments

In this section, the same experiments performed on the QWH problems are done on the apparently similar magic square problem.

**The Probability of Searching from an Empty Solution** Results for varying the probability of starting from an empty solution are shown in Figure 3.9. Unlike on the QWH problems, SGMPCS does not out-perform randomized restart (i.e., $p = 1$) on the magic square problems. In fact, $p = 1$ results in the best performance while $p = 0$, the best setting on the QWH problems, performs worst.

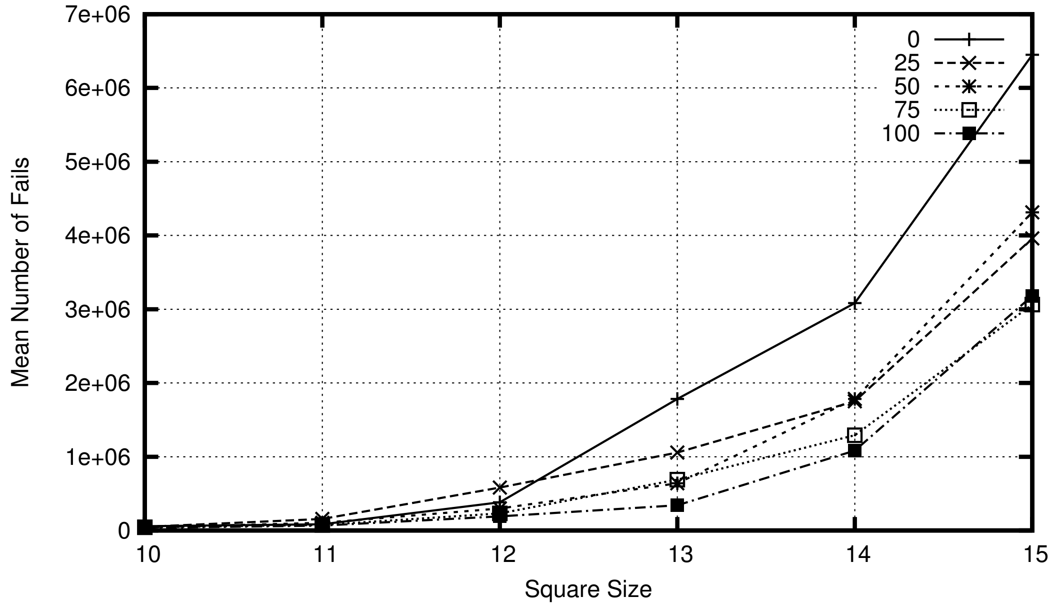

Figure 3.9: Mean number of fails to solve magic square problems with varying values for $p$.

**Elite Set Size** As seen in Figure 3.10, varying the elite set size has little effect on the performance of SGMPCS on the magic square problems. All results are worse than $p = 1$ from the previous experiment which never uses the elite set to guide search.

**Backtrack Method** Using LDS on the magic square instances led to extremely bad performance. On the order-10 instance, no solutions were found on any run using a global limit of 10,000,000 fails. As a result, we do not display the results here.

**Fail Sequence** Figures 3.11 and 3.12 show the results of SGMPCS on magic square using the three fail sequences. The differing result in terms of fails and time observed for the QWH problems are magnified here. Unlike the QWH results, the polynomial sequence is slightly better than Luby in terms of fails as seen in Figure 3.11. In terms of overall search time, the Luby sequence is orders of magnitude worse than the polynomial limit as seen in the log scale Figure 3.12. Overall, the polynomial sequence is the best performer.

**Initialization Bound** The results for varying the fail bound used to initialize the elite set are shown in Figure 3.13. There seems to be no effect of varying the fail bound. While an initialization limit of 1 appears better at order 15, the results are affected by 15% of the runs hitting the 10,000,000 global limit, and the difference is insignificant.

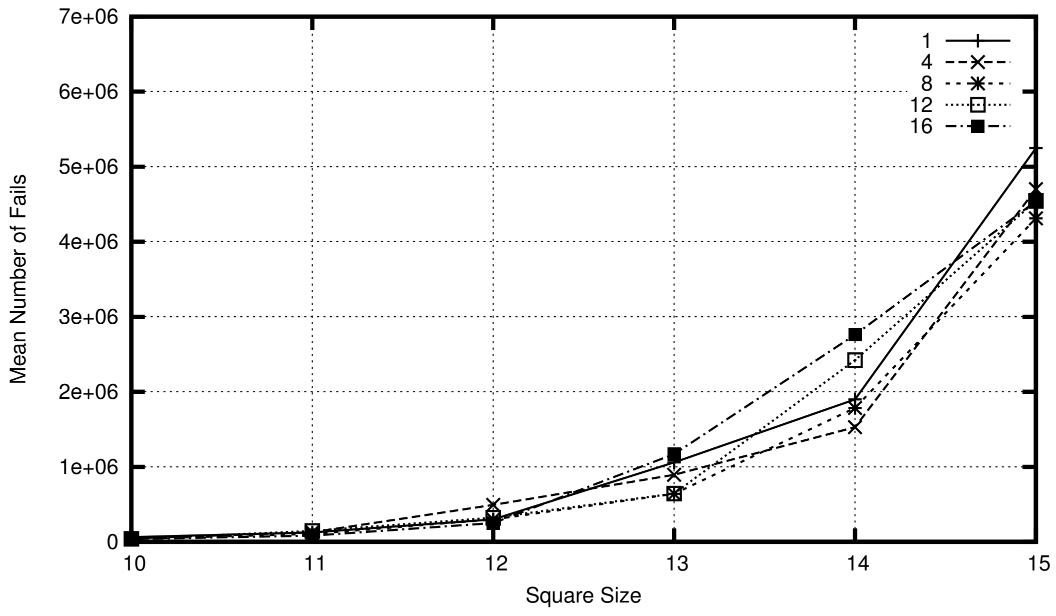

Figure 3.10: Mean number of fails to solve magic square problems with varying elite sizes.

**Comparison with Other Techniques** Figure 3.14 displays the comparison of SGMPCS with the best settings found in the QWH (*SGMPCS:qwh*) and magic square (*SGMPCS:magic*) experiments with the other search algorithms. For *SGMPCS:magic* the following parameters are used: $|e| = 8, p = 0.75,$[3] and the backtrack method is chronological. *Restart* and the SGMPCS variations are all better than chronological search but SGMPCS performs slightly worse than restart.

Given the QWH results and the similarity in form between QWH and magic square problems, the fact that there seems to be no benefit from guiding search with elite solutions is intriguing. We return to these results in Section 3.3.

[3]The second best $p$ is used since $p = 1$ is equivalent to randomized restart.

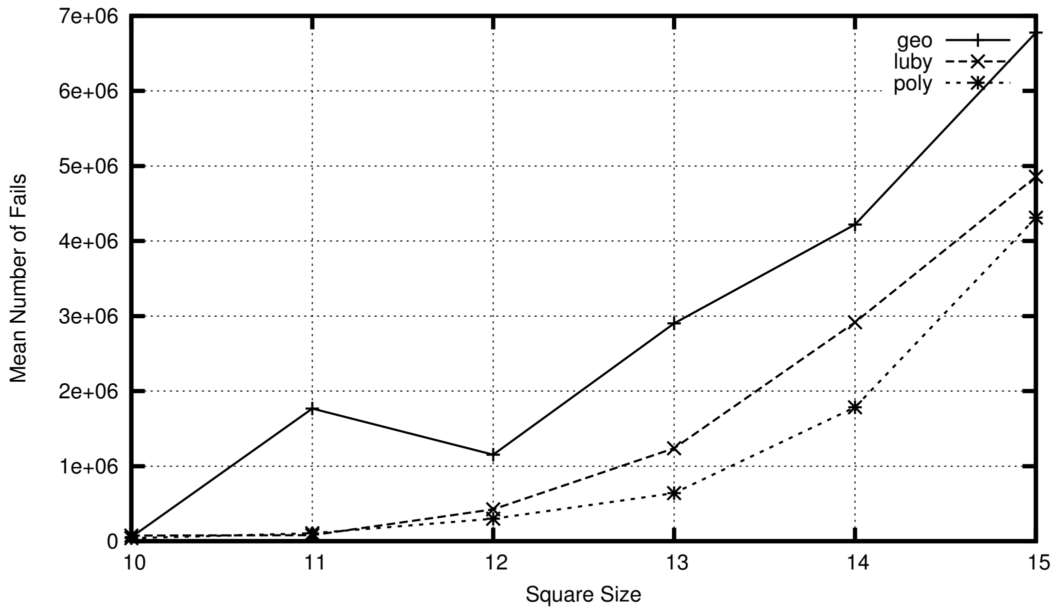

Figure 3.11: Mean number of fails to solve magic square problems with each fail sequence.

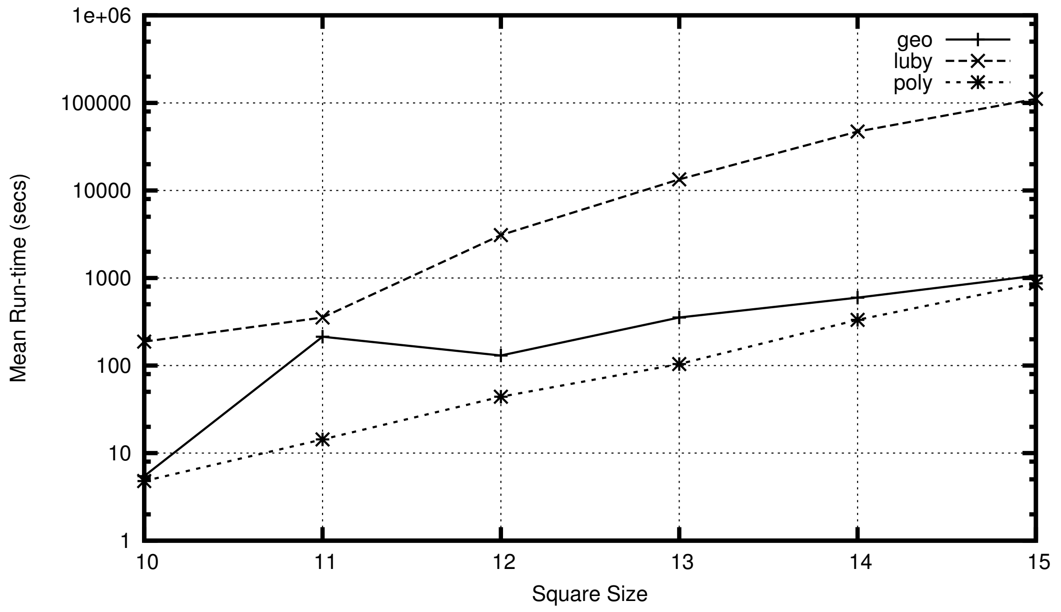

Figure 3.12: Mean time to solve magic square problems with each fail sequence.

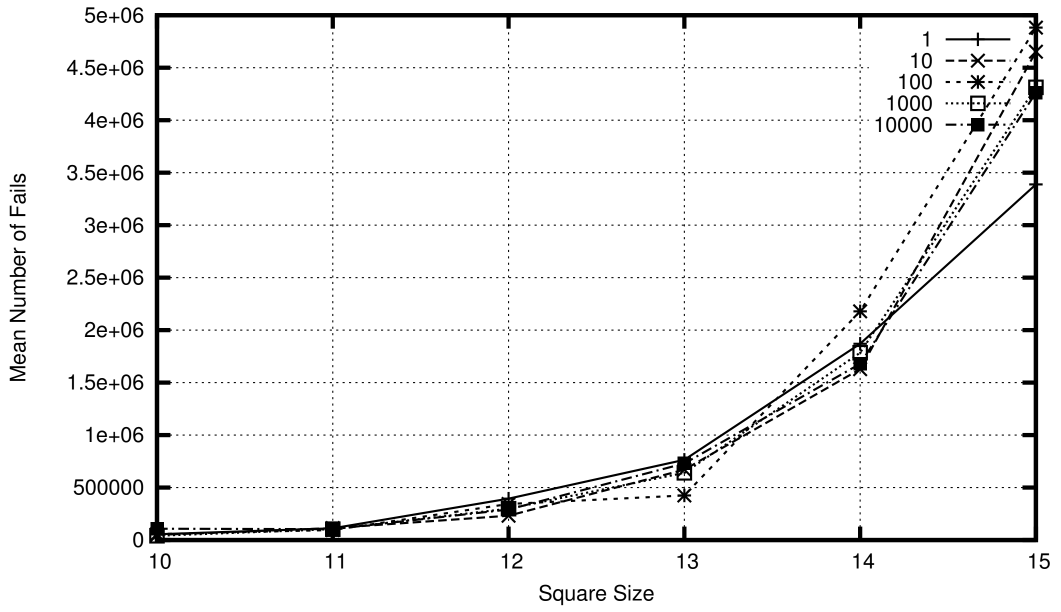

Figure 3.13: Mean number of fails to solve magic square problems with varying initialization bounds.

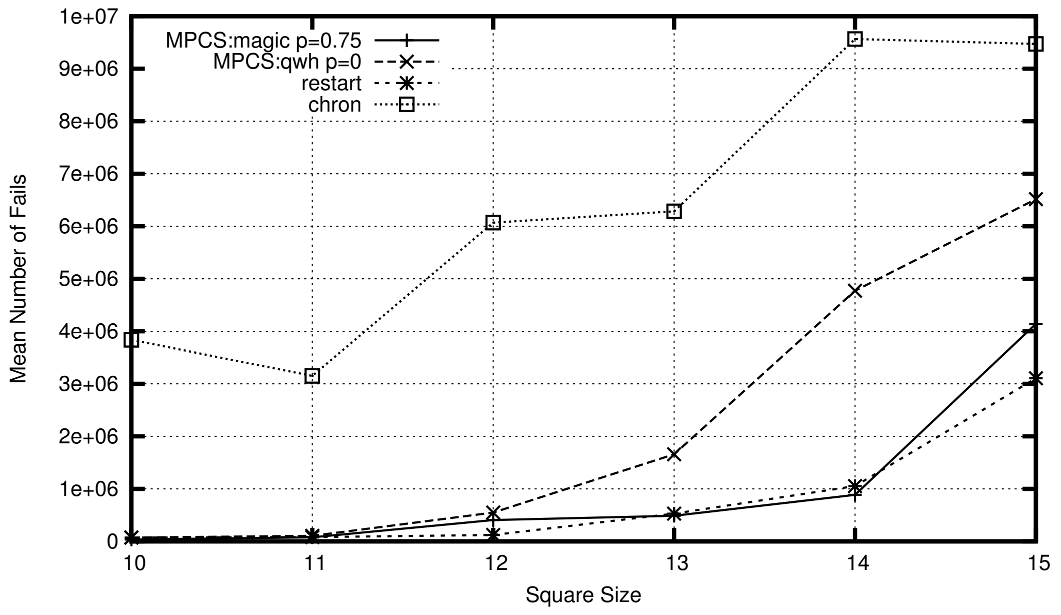

Figure 3.14: Mean number of fails comparing different search techniques and SGMPCS with best parameters found for magic square.

### 3.2.5 Multi-Dimensional Knapsack Experiments

#### 3.2.5.1 Initial Experiments

We performed an identical set of experiments as above to evaluate the different parameter settings on SGMPCS performance on the multi-dimensional knapsack problems. Most of the parameters have no discernible effect. The two exceptions, shown in Tables 3.5 and 3.6, are the backtracking method, where chronological backtracking is better, and the fail sequence, where the geometric fail limit shows a clear pattern of the best performance.

The best settings for SGMPCS on the knapsack problems—the geometric fail limit along with the other default settings—are used in a final comparison with the other search techniques and SGMPCS with the best settings found from the QWH experiments. As seen in Table 3.7, both SGMPCS and randomized restart perform poorly in comparison to basic chronological search. SGMPCS is far from state-of-the art on these problems, significantly better performance than all of these results is reported by Refalo [54] using impact-based heuristics. The best SGMPCS settings (other than $p = 1$) perform about the same as restart.

Given that chronological backtracking achieves the best results on these problems, there is a relatively straightforward interpretation of the results of the parameter settings on SGMPCS. The geometric fail limit grows the fastest of all fail sequences tested. It appears that the fail bound simply grows until a single chronological search can be done. In other words, the use of restarts and elite solutions is simply a distraction.

| | chron | | | lds | | |
|---|---|---|---|---|---|---|
| | %sol | fails | time | %sol | fails | time |
| mknap1-0 | 100 | **0** | **0.0** | 100 | 1 | **0.0** |
| mknap1-2 | 100 | **23** | **0.0** | 100 | 46 | 0.0 |
| mknap1-3 | 100 | **660** | **0.1** | 100 | 1287 | 0.1 |
| mknap1-4 | 100 | **48219** | **4.4** | 100 | 98633 | 15.6 |
| mknap1-5 | 80 | **4156366** | **287.7** | 60 | 5661855 | 800.5 |
| mknap1-6 | 0 | **10000000** | **857.8** | 0 | **10000000** | 2422.0 |
| mknap2-PB1 | 100 | **32621** | **1.8** | 100 | 40384 | 3.9 |
| mknap2-PB2 | 95 | **3419627** | **186.5** | 75 | 5050786 | 933.5 |
| mknap2-PB4 | 100 | **40711** | **1.7** | 100 | 65314 | 8.0 |
| mknap2-PB5 | 100 | **10813** | **0.9** | 100 | 21824 | 5.4 |
| mknap2-PB6 | 100 | **31013** | **11.0** | 100 | 43158 | 48.9 |
| mknap2-PB7 | 85 | **4446171** | **1361.1** | 75 | 5815284 | 5442.2 |

Table 3.5: Multi-dimensional knapsack results for both backtracking methods.

| | geo | | | luby | | | poly | | |
|---|---|---|---|---|---|---|---|---|---|
| | %sol | fails | time | %sol | fails | time | %sol | fails | time |
| mknap1-0 | 100 | 1 | 0.0 | 100 | 1 | 0.0 | 100 | **0** | **0.0** |
| mknap1-2 | 100 | **22** | 0.0 | 100 | 29 | **0.0** | 100 | 23 | **0.0** |
| mknap1-3 | 100 | 465 | 0.0 | 100 | **448** | **0.0** | 100 | 660 | 0.1 |
| mknap1-4 | 100 | **27401** | **2.5** | 100 | 57524 | 8.6 | 100 | 48219 | 4.4 |
| mknap1-5 | 90 | 4293937 | 288.2 | 65 | 5912766 | 759.7 | 80 | **4156366** | **287.7** |
| mknap1-6 | 0 | 10000000 | **850.7** | 5 | **9946401** | 1770.6 | 0 | 10000000 | 857.8 |
| mknap2-PB1 | 100 | **26625** | **1.5** | 100 | 36670 | 3.2 | 100 | 32621 | 1.8 |
| mknap2-PB2 | 100 | **3018792** | **162.7** | 95 | 3811687 | 417.0 | 95 | 3419627 | 186.5 |
| mknap2-PB4 | 100 | **27860** | **1.2** | 100 | 38678 | 2.6 | 100 | 40711 | 1.7 |
| mknap2-PB5 | 100 | 11776 | 1.0 | 100 | **7378** | **0.6** | 100 | 10813 | 0.9 |
| mknap2-PB6 | 100 | **20590** | **7.4** | 100 | 33980 | 19.9 | 100 | 31013 | 11.0 |
| mknap2-PB7 | 100 | **1542933** | **596.8** | 75 | 4619805 | 2730.6 | 85 | 4446171 | 1361.1 |

Table 3.6: Multi-dimensional knapsack results with varying fail-sequences.

| | chron | | | restart | | | SGMPCS-qwh best | | | SGMPCS-knap best | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %sol | fails | time | %sol | fails | time | %sol | fails | time | %sol | fails | time |
| mknap1-0 | 100 | 1 | **0.0** | 100 | 1 | **0.0** | 100 | **1** | 0.0 | 100 | 1 | **0.0** |
| mknap1-2 | 100 | 26 | 0.0 | 100 | **18** | 0.0 | 100 | 24 | 0.0 | 100 | 26 | **0.0** |
| mknap1-3 | 100 | **363** | **0.0** | 100 | 510 | 0.0 | 100 | 724 | 0.1 | 100 | 571 | 0.0 |
| mknap1-4 | 100 | **15551** | **1.1** | 100 | 32601 | 3.0 | 100 | 30939 | 3.1 | 100 | 32305 | 2.9 |
| mknap1-5 | 100 | **2862059** | **148.3** | 75 | 5500578 | 400.2 | 85 | 5035286 | 376.6 | 100 | 3341859 | 229.9 |
| mknap1-6 | 0 | **10000000** | **660.6** | 0 | **10000000** | 913.2 | 0 | **10000000** | 938.5 | 0 | **10000000** | 856.2 |
| mknap2-PB1 | 100 | **15200** | **0.7** | 100 | 26568 | 1.6 | 100 | 34679 | 2.0 | 100 | 26324 | 1.5 |
| mknap2-PB2 | 100 | 3087894 | 124.7 | 80 | 5705325 | 326.1 | 100 | **1914350** | **105.7** | 95 | 2969530 | 159.2 |
| mknap2-PB4 | 100 | **11870** | **0.4** | 100 | 25608 | 1.2 | 100 | 27202 | 1.2 | 100 | 16382 | 0.7 |
| mknap2-PB5 | 100 | **6138** | **0.4** | 100 | 10924 | 0.9 | 100 | 14836 | 1.3 | 100 | 16920 | 1.4 |
| mknap2-PB6 | 100 | **11789** | **3.3** | 100 | 33030 | 11.1 | 100 | 29635 | 10.4 | 100 | 25017 | 8.9 |
| mknap2-PB7 | 100 | **1469050** | **344.1** | 100 | 2128190 | 584.1 | 75 | 4543196 | 1310.6 | 100 | 2295825 | 659.5 |

Table 3.7: Comparison of multi-dimensional knapsack results for different algorithms and best SGMPCS parameter settings.

### 3.2.5.2 Multi-Dimensional Knapsack Optimization

It appears that the multi-dimensional knapsack problems create a particular challenge for SGMPCS. Detailed traces of the SGMPCS runs show that early in the search all the elite solutions have an objective value of 0: all of the variables are assigned but the solution does not satisfy all constraints. The propagation of the linear constraints tends not to result in domain wipe-outs but rather fully assigned variables that break one or more constraints. In a subsequent set of experiments, we modified our objective to be the sum of the number of assigned variables and the number of satisfied constraints. This change did not improve the problem solving performance. Analysis indicated that, in many situations, though not always, the elite solutions were quickly populated with solutions that assigned all variables and only broke one constraint: the overall cost constraint: $\sum_{i=1}^{n} x_i p_i = C$.[4] The poor performance, therefore, appears to be based on poor heuristic guidance. The number of assigned variables and the number of satisfied constraints does not appear to be a useful way to compare elite solutions because there seem to be many solutions with the same value of these measures. Under such conditions, the elite set quickly stagnates to the first $|e|$ solutions found that only break the cost constraint.

To investigate this intuition, we modify the satisfaction model of multi-dimensional knapsack to include cost information. We believe this will provide a better way to discriminate among potential elite solutions. The change in the model is to remove the constraint requiring that the profit be equal to a previously known optimal value and to replace it with an optimization function of the form: $f(x) = C - \sum_{i=1}^{n} x_i p_i$, where $C$ is the previously known optimal cost, $x$ is the vector of decision variables, and the initial bounds on the value of $f(x)$ are $[0, C]$. We then solve the problem to minimize $f(x)$. Note that this model is a hybrid satisfaction/optimization model because different feasible solutions have different costs but we know that $f(x) = 0$ is a globally satisfying solution and, as a result, there is no need to prove optimality once such a solution is found. The constraint propagation from this model is weaker than from Refalo's satisfaction model because the constraint on the profit is looser. To be clear, the main purpose of this change is not to find a useful way to solve multi-dimensional knapsack problems. Rather, the main purpose is to learn more about the behaviour of SGMPCS by

[4]Recall that we made the multi-dimensional knapsack a satisfaction problem, following Refalo [54], by requiring that the cost be equal to the (previously known) optimal cost, $C$.

investigating how this change affects performance.

To apply SGMPCS to our new multi-dimensional knapsack model, there are two changes. First, solutions are complete solutions (i.e., when $m = n$, see Section 2.2.2) and they are compared based of their cost, with smaller cost being preferred. Second, when performing an individual search, to exploit constraint propagation, we must specify the upper bound of the the cost function. Here, we use the local bounding method introduced in Beck [6]:[5]

- When search is started from an empty solution, the upper bound on the cost function is set to one less than the cost of the worst current elite solution.

- When search is started from an elite solution, the upper bound on the cost function is set to one less than the cost of the starting elite solution.

The variable and value ordering heuristics and other experimental details used above are maintained.

Table 3.8 displays the results of the three optimizing algorithms on the same twelve multi-dimensional knapsack problems. Table 3.9 presents the number of problem instances for which each algorithm clearly performed better either in the satisfaction model or the optimization model based on the mean number of fails and the mean run-time. For example, chron incurred lower mean fails on 10 of the instances when modeled as a satisfaction rather than as an optimization problem. Similarly, chron incurred a lower run-time on 5 of the instances when they were modeled as satisfaction problems.

Both chron and restart perform better on the satisfaction model. This can likely be attributed to the stronger propagation from the cost constraint. In contrast, in its optimization form, SGMPCS performs remarkably well. SGMPCS was able to find a solution with the optimal cost for mknap1-6 in an average of 28 seconds, a feat it was unable to accomplish in its 10,000,000 fail limit (in an average of 857.8 seconds) when posed as a satisfaction problem. SGMPCS performs better in the optimization model on 5 instances, in terms of the number fails and on 6 instances based on run-time. As nothing else was modified, this improvement must be attributed to the stronger heuristic guidance produced by the cost-based comparison of elite solutions. Overall, SGMPCS on the optimization model solves 5 instances better (in terms of both fails and run-time) than any other technique tested here on any model.

## 3.3 Discussion

The primary goals of this chapter were to apply Solution Guided Multi-Point Constructive Search to a selection of constraint satisfaction problems and to systematically investigate the impact of the various parameter settings. As the extensive experiments on the quasigroup-with-holes problems indicate, SGMPCS 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 [6] (i.e., low $|e|$ value, low $p$ value), the QWH results demonstrate that maintaining more than one elite solution

[5]Experiments with the global bounding method did not exhibit significant differences from using the local bound.

| | chron | | | restart | | | SGMPCS-poly | | |
|---|---|---|---|---|---|---|---|---|---|
| | %opt | fails | time | %opt | fails | time | %opt | fails | time |
| mknap1-0 | 100 | 1 | **0.0** | 100 | **0** | **0.0** | 100 | 1 | **0.0** |
| mknap1-2 | 100 | 55 | **0.0** | 100 | 60 | **0.0** | 100 | **46** | **0.0** |
| mknap1-3 | 100 | **708** | **0.0** | 100 | 1150 | 0.1 | 100 | 994 | 0.1 |
| mknap1-4 | 100 | 29632 | **2.1** | 100 | 51098 | 3.8 | 100 | **27927** | **2.1** |
| mknap1-5 | 100 | 3986153 | 211.7 | 80 | 14681817 | 789.0 | 100 | **159198** | **8.7** |
| mknap1-6 | 0 | 22938204 | 1500.0 | 0 | 21945243 | 1501.0 | 100 | **407050** | **27.9** |
| mknap2-PB1 | 100 | **20236** | **0.9** | 100 | 46467 | 2.1 | 100 | 32189 | 1.5 |
| mknap2-PB2 | 100 | 3931370 | 164.5 | 100 | 7531632 | 316.5 | 100 | **89037** | **4.1** |
| mknap2-PB4 | 100 | **20615** | **0.7** | 100 | 32111 | 1.0 | 100 | 30167 | 1.0 |
| mknap2-PB5 | 100 | **8339** | **0.5** | 100 | 34797 | 2.3 | 100 | 22061 | 1.5 |
| mknap2-PB6 | 100 | **23496** | **6.6** | 100 | 33351 | 10.1 | 100 | 26730 | 7.6 |
| mknap2-PB7 | 100 | 2294518 | 532.0 | 55 | 4808054 | 1140.1 | 100 | **89958** | **21.2** |

Table 3.8: Multi-dimensional knapsack optimization results for three optimization algorithms.

| | # fails | | run-time | |
|---|---|---|---|---|
| | SAT | OPT | SAT | OPT |
| chron | 10 | 0 | 5 | 0 |
| restart | 10 | 1 | 2 | 0 |
| SGMPCS | 5 | 5 | 0 | 6 |

Table 3.9: The number of multi-dimensional knapsack problem instances (out of 12) where an algorithm performed better on the the satisfaction model (SAT) or the optimization model (OPT) based on mean number of fails and mean run-time.

can contribute to improved performance. 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 SGMPCS on constraint satisfaction problems. Further work is necessary to understand why QWH problems benefit from a larger elite size and scheduling problems do not.

When the empirical results for the magic square are considered, our conclusions are more nuanced and interesting. The significant change in the relative performance of randomized restart and SGMPCS when moving from the QWH to the magic square problems is particularly interesting given the similarities in the problems. What is it about the differences between the problems that lead to strong SGMPCS performance on QWH and weak performance on magic square? We hope that answering this question will lead us to an understanding of the problem characteristics that influence SGMPCS performance and ultimately to an understanding of the reasons for SGMPCS performance. We believe it would be interesting to generate some magic-square-with-holes problems to determine if the phase transition behaviour of QWH is seen and to evaluate the performance of randomized restart and SGMPCS.

The multi-dimensional knapsack experiments also provide interesting results. While SGMPCS performed poorly on the satisfaction model, redefining the elite solution cost and adding a cost function to the model allowed SGMPCS to perform very well. Both chronological backtracking and restart perform worse on the optimization model than they did with the satisfaction model. These results support our intuition that the criteria for comparing elite solutions is critical for the performance of SGMPCS.

We believe that SGMPCS will perform well when there is a "path" of good solutions such that given one solution, a better solution can be found within a small number of backtracks for some variable ordering. Revisiting solutions with different variable orderings, essentially defines the neighbourhoods SGMPCS moves between. This model of SGMPCS search suggests that we may be able to adapt the fitness-distance analysis tools developed in local search [36] to SGMPCS. On problems where SGMPCS performs well, we expect to observe a strong correlation between the distance between solutions and the quality difference between solutions: solutions of similar quality will tend to be close to each other in the search tree. We explore such models of search behaviour in Chapters 5 and 6.

## 3.4 Conclusion

This chapter is the first systematic application of Solution Guided Multi-Point Constructive Search to constraint satisfaction problems. Our empirical results demonstrate that SGMPCS 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, our results are in agreement with previous studies on optimization problems in the scheduling domain, however the results reinforce the intuition that exploiting multiple viewpoints can be of substantial benefit in heuristic search.

The types of problems that we experimented with reveal an interesting pattern. SGMPCS significantly out-performs chronological backtracking on the quasigroup-with-holes and magic square problems but significantly under-performed on the multi-dimensional knapsack problems. Similarly, SGMPCS out-performed randomized restart on the quasigroup problems, performed slightly worse on the magic square problems, and performed about the same on the multi-dimensional knapsack problems. Evaluating the elite solutions by their cost instead of the number of assigned variables results in orders of magnitude speed-up for SGMPCS on the multi-dimensional knapsack problems.

In the next chapter, we investigate one main factor which we believe influences this varying behaviour of SGMPCS: the exploitation of heavy-tailed behaviour in backtracking search.

# Chapter 4

# Heavy Tails in Solution Guided Multi-Point Constructive Search

## 4.1 Introduction

It has been shown that constructive search algorithms can be improved through a randomized restart technique which exploits the heavy-tailed nature of a randomized algorithm's run-time distribution [26]. By demonstrating that SGMPCS can take advantage of these heavy tails in the same way, we can incorporate all work on heavy tails in backtracking search into our understanding of SGMPCS.

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

## 4.2 Background

Given a single problem instance and a backtracking algorithm with some degree of randomness, the cost of a single random run of the algorithm can be highly variable. Sometimes a solution is found quickly, while with other random seeds, the time to find a solution is so high that it becomes practically unsolvable. Past studies have found these distributions to have heavy tails [26, 25, 27]. More specifically, the probability that the cost of the next stochastic run (random variable $X$) is greater than some value $x$ can be modeled by a Pareto-Levy Distribution:

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

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

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

As shown theoretically and empirically by Gomes et al. [26], if randomized backtracking algorithm is repeatedly restarted after some resource limit, the expected run-time of such a rapid randomized restart search will no longer be heavy-tailed. Shown in Equation 4.1 is the expected tail of the runtime distribution of a randomized restart strategy, the probability that the random variable representing the cost of this strategy $S$ will be greater than some cost $s$. $A$ is the random variable representing the original randomized backtracking search, and $c$ is the *fixed* restart cut-off used. The form Equation 4.1 follows an exponential distribution, so we expect heavy tails to be eliminated (exponential distributions have finite mean and variance).

$$P[S > s] = (1-p)^{\lfloor s/c \rfloor} \;\; P[A > s \mod c] \tag{4.1}$$

Instead of a fixed limit, the sequence of limits at each successive restart can increase to give a complete search procedure (the limit will eventually be large enough to exhaust the whole tree). As discussed in Chapter 2, Luby et al. [43] have formulated a universal sequence of limits that is optimal given no prior information of the original distribution, and only a log factor away from the optimal fixed cut-off computed from complete information of the search cost distribution.

Much research has been done on when and how these heavy-tailed distributions come about (see Chapter 2 for a more detailed overview). Williams [73] has shown how they are related to the idea of back-door variables. A related paper has shown a theoretical model of backtracking search which predicts heavy-tailed behaviour when search is unbalanced [13]. Gomes [28] correlates heavy tails with thrashing behaviour in inconsistent subtrees.

By arguing that SGMPCS uses a randomized backtracking search that should experience heavy tails, and a restart strategy that removes them, we can inherit all of this research into our understanding of SGMPCS.

## 4.3 SGMPCS as a Randomized Restart Model

When the probability of starting a non-guided iteration in SGMPCS is 1, the algorithm becomes identical to randomized restart. If the underlying randomized backtracking search has heavy tails, each run will sample from it, and as discussed in Section 4.2, heavy tails will be eliminated and performance will benefit over standard chronological search. The question then becomes:

[1] Of course constructive search is complete, the size of the full search space is finite, so none of the moments can actually be infinite. Yet, some of the runs will take so long as to be practically infinite, in that we will never wait until they are finished. The distributions are actually "truncated heavy tails" [27]. The infinite model works well for the range of search costs investigated here.

when SGMPCS is guided by elite solutions, will heavy tails still be eliminated? The main difference is that instead of sampling from the distribution of backtracking runs, SGMPCS will be sampling from a distribution of guided runs. If we let $G$ be a random variable representing this distribution, the number of backtracks needed for the next guided run, we get a similar formula for the probability mass of the tail as Equation 4.1, with $p = P[G \leq c]$, and a final term of $P[G > c \mod s]$. As long as $P[G \leq c] > 0$, the distribution will still have an exponential form, and SGMPCS should still eliminate heavy tails, getting a similar performance advantage over basic backtracking as randomized restart does. Hence, this guided distribution, $G$, does not even have to be heavy-tailed for SGMPCS to perform better than chronological search. It suffices that the original backtracking distribution, $A$, is heavy-tailed for SGMPCS to perform better than chronological search. Yet, if $G$ is heavy-tailed, then a rapid restart technique is just as necessary in SGMPCS as it is for randomized restart.

In the following sections, we attempt to empirically confirm the theoretical argument made here. In Experiment 1, we find problem instances—we will focus on job-shop scheduling optimization—in which the search cost distribution of chronological backtracking search is heavy-tailed. Secondly, we examine how search performance of SGMPCS and a randomized restart algorithm changes as the instances start to exhibit heavy-tailed distributions. Finally in Experiment 3, we examine the distribution of *guided* runs.

## 4.4 Experiment 1: Heavy Tails in Job-Shop Scheduling

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

### 4.4.1 The Job-Shop Scheduling Problem

As described in Section 2.1.1 an $n \times m$ job-shop scheduling problem (JSP) contains $n$ jobs each composed of $m$ completely ordered activities. Each activity, $a_i$, has a predefined duration, $d_i$, and a resource, $r_i$, that it must have unique use of during its duration. There are 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.

All job-shop problems experimented with here are square, in that the number of jobs is equal to the number of resources $n = m$. Ten problems of each order (13x13,14x14,15x15,16x16) are generated using an existing generator [69]. The routings of the jobs through the machines are randomly generated and the activity durations are independently and randomly drawn from $[1, 99]$.

### 4.4.2 Experimental Details

For all algorithms, texture-based heuristics [9] are used to identify a resource and time point with maximum competition among the activities. This resource and time point are then used to 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. The standard constraint propagation techniques for scheduling [51, 41, 42] are also used in all experiments and algorithms.

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

### 4.4.3 Measuring Run-Time Distributions of Chronological Search

To determine the run-time distribution of search cost in our job-shop scheduling problems each problem instance is run 1000 times with different random seeds. Due the size of the problems of interest, we only measure the search until a near optimal ($4\%$ from optimal) solution is found. Given an optimal makespan, $C^*$, we only search for a makespan, $C = 1.04 * C^*$. This goal of $4\%$ from optimal was chosen to allow most runs to finish within the 10,000,000 choice point limit, but small enough that the improved performance of randomized restart and SGMPCS can be seen in the graph of mean relative error over time at the largest problem size (see Fig 4.2). A similar technique has been used in investigating heavy tails in optimization problems [25].

Ideally no global limit would be put on each overall search cost. But considering how many runs were needed, and that given the size of the problems certain runs could take days, a high limit of 10,000,000 choice points is used. Although this affects the strength of the statistics (some extreme values expected by heavy-tailed distributions would be missed), a modified maximum likelihood estimate of the index of stability has already been used for such a case [26].

### 4.4.4 Generating Random Solutions

A random solution is generated at the start of each run in order to set the initial upper-bound, and for guiding the solutions in Experiment 3. These are generated through a *schedule-or-postpone* [58] technique that attempts to assign start times to all activities. At each step it chooses randomly from all activities which can be scheduled next and assigns the activity its earliest start time. An activity can be scheduled next if the preceding activity in its job has been assigned (or if it is the first activity), and if the activity has not been by postponed. An activity becomes postponed if assigning it its earliest start time leads to a failure. It stays postponed until this earliest start time is removed due to the propagation that is applied after each assignment.

### 4.4.5 Results

Displayed in Figure 4.1 are log-log survival functions for a selection of instances of each order. The y-value corresponds (log scale) the the proportion of the 1000 runs that still could not find a good solution (within 4% of optimal) after creating x (log-scale) choice points. The overlaid line is the fitted Pareto distribution of the tail using a modified maximum likelihood estimate of $\alpha$ formulated in Gomes et. al [26]. Estimates of $\alpha$ for each instance are shown in Table 4.1. For heavy tails, we would expect the log-log plot of the tail to be linear, and have and $\alpha$ value that is associated with all infinite moments $\alpha < 1$. From the graphs and figures, we see that heavy-tailed behaviour starts to emerge by size 16.

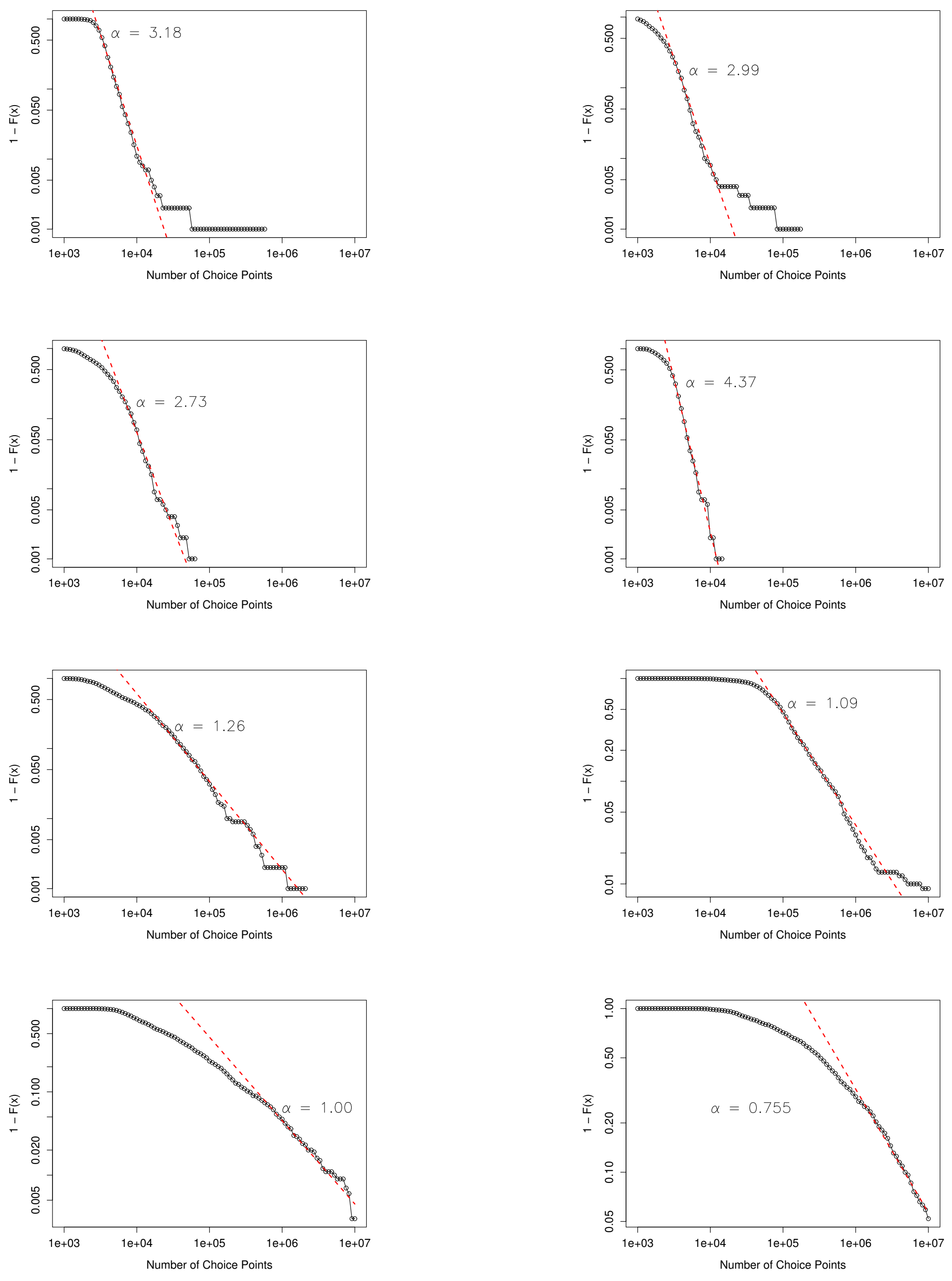

Figure 4.1: Log-Log plot of the survival function of backtracking search to find 4% of optimal for two random JSP instances for sizes, from top to bottom, 13, 14, 15, 16. Also plotted, is the fitted Pareto distribution of the tail with the estimated parameter $\alpha$.

## 4.5 Experiment 2: Comparing Algorithms

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

### 4.5.1 Experimental Details

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

Three algorithms are tested:

- Standard chronological backtracking (Chron): Similar to the runs from the last experiment except the heuristic is not randomized, and the texture heuristic is used in the value ordering. Since it is not randomized, one run is performed for each instance.
- Randomized Restart (Restart): A randomized restart algorithm using the randomized texture heuristic from Experiment 1. The fail limit is polynomial with a step of 32 (see Section 3.1). Each instance is run 10 times. As in SGMPCS, on each successive restart, the best solution found so far is used as the new upper bound on the makespan.
- Solution Guided Multi-Point Constructive Search (SGMPCS): To simplify the algorithm, and from good performance found in [6] the following parameters are used: $|e| = 1$, $p = 0.25$, initialization limit : 100, fail sequence: polynomial with a step of 32. The same randomized texture heuristic as in randomized restart is used, except, when guiding from a solution, the activity ordering in the guiding solution is always asserted. Each instance is run 10 times.

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

### 4.5.2 Results

Displayed in Figure 4.2 are, for each problem size, the mean relative errors (MRE) relative to the known optimal solution at 10 second intervals. The MRE is the mean of the relative error of each 10 problems at a given size $n$:

$$MRE(a, K_n, R, t) = \frac{1}{|R||K_n|} \sum_{r \in R} \sum_{k \in K_n} \frac{c(a, k, r, t) - c^*(k)}{c^*(k)} \tag{4.2}$$

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

While standard chronological search performs much better on the smaller problems, by order 16, it becomes the worst in terms of average performance. Table 4.1 displays the average number of choice points to find a 4% of optimal solution (what we used to measure heavy tails), along with the estimated $\alpha$ value from Experiment 1. Chronological search can still perform best on some larger instances, but it is also highly variable between instances, which we would expect from heavy tails. Also from the table we can see, in contrast to graphs in Figure 4.2, that simple restart always performs better than SGMPCS. This may seem contradictory, but what was plotted in Figure 4.2 was the mean relative error at a set time (about half the runs will have a larger relative error at this time). In contrast, Table 4.1 measures the mean cost of each run to get to this 4% relative error.

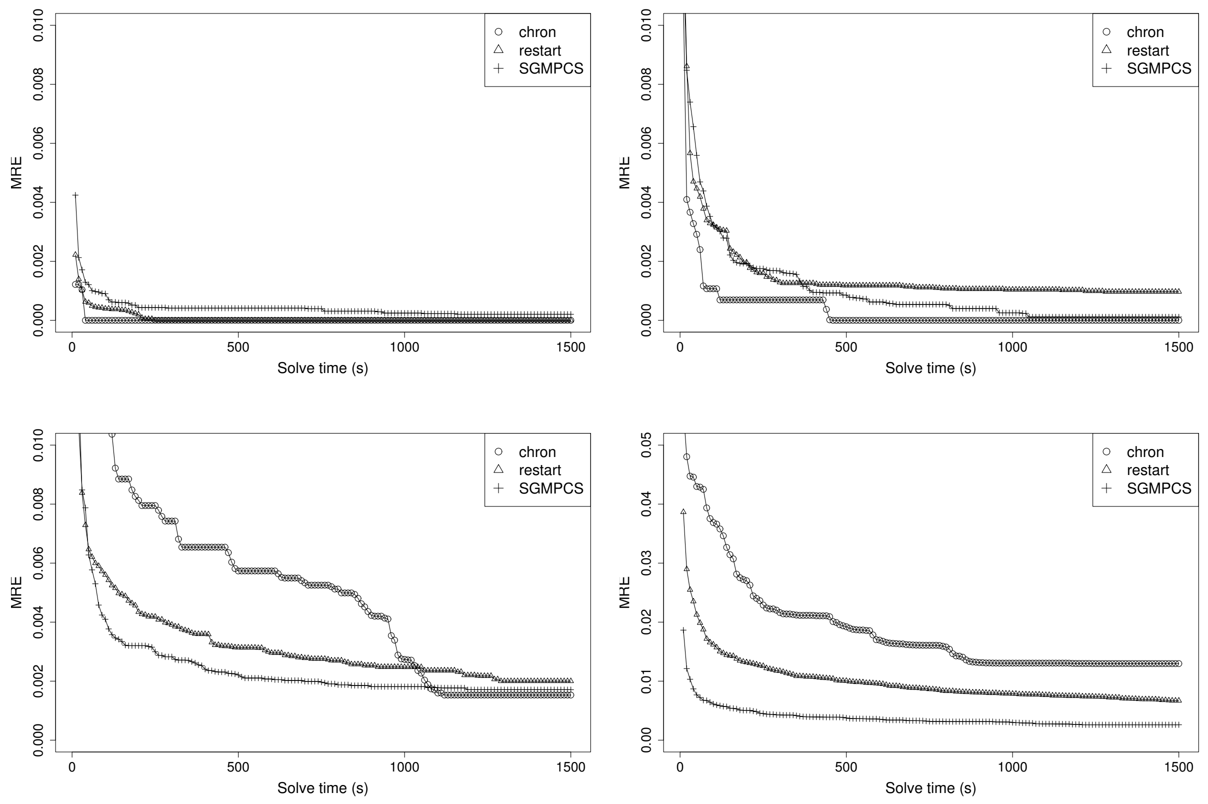

Figure 4.2: Mean relative error of best solutions found over time for JSP problems of sizes 13 (top left), 14 (top right), 15 (bottom left), and 16 (bottom right). The y-axis of size 16 is different from the others since the chronological line would not appear otherwise.

The graph in Figure 4.3 attempts to show how average performance relative to chronological search changes as the distributions become heavy-tailed, measured through the average estimate of $\alpha$. Plotted on the x-axis are the means of the 10 $\alpha$ values for each size. On the y-axis is average performance relative to chronological search for each size $n$:

$$\bar{R_n} = \frac{(\bar{A_n} - \bar{C_n})}{\bar{C_n}} \tag{4.3}$$

where $\bar{A_n}$ is the mean number of choice points to find a 4% of optimal solution on problems of size $n$ for either SGMPCS or Restart, and $\bar{C_n}$ is the mean performance for standard chronological search on the same problems. It can be observed that as the $\alpha$ value approaches 1 from above, the performance scores for SGMPCS and restart both become negative signifying improved mean performance relative to chronological search.

| | | Chron | | Restart | | SGMPCS | |
|---|---|---|---|---|---|---|---|
| size | $\bar{\alpha}$ | mean | sd | mean | sd | mean | sd |
| 13 | 3.46 | 1532 | 447 | 2694 | 1889 | 11081 | 2120 |
| 14 | 2.02 | 3939 | 3327 | 4365 | 1479 | 18364 | 25374 |
| 15 | 1.35 | 21292 | 35336 | 8050 | 7349 | 20934 | 8185 |
| 16 | 1.09 | 92694 | 127361 | 17062 | 11551 | 19585 | 3553 |

Table 4.1: Mean and standard deviation of the number of choice points to to find a 4% from optimal solution to JSP instances of varying size along with mean $\alpha$ values of their tails measured in Experiment 1.

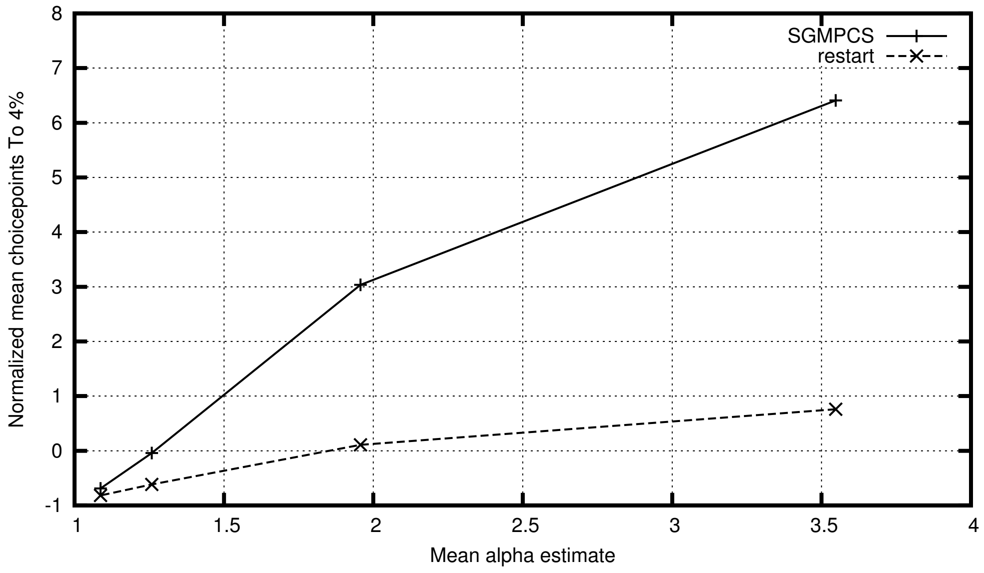

Figure 4.3: Mean relative performance from randomized restart and SGMPCS for problem sizes of changing average $\alpha$ estimates.

## 4.6 Experiment 3: Finding Run-Time Distributions of Guided Search

We have seen that heavy-tailed behaviour can occur in JSP instances and that randomized restart and SGMPCS can both benefit from these search cost distributions. The purpose of our final experiment to look at how SGMPCS may change this underlying distribution. The main question is whether this distribution is still heavy-tailed. Even if the guided distribution is heavy-tailed, the extremely long guided runs will still always be avoided because of the restarts SGMPCS uses, and the overall search cost distribution of SGMPCS will not be heavy-tailed. If the guided distribution is not heavy-tailed, then perhaps there is a way to guide from solutions without the need of restarts. The purpose of this experiment is to see if a rapid restart technique is critical for the performance of SGMPCS (i.e., to avoid to long runs).

### 4.6.1 Experimental Details

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

### 4.6.2 Results

The log-log plots of the survival function over all instances are shown in Fig. 4.4, where the y-axis represents the proportion of runs over all instances of size 16 that have found a 4% from optimal solution after x choice points have been created. From the plots, we can see the search cost distributions for guided and non-guided runs both exhibit heavy tails, except that in the guided case, the heavy tails occur sooner and are less steep. The maximum likelihood estimates of the $\alpha$ parameter are 0.728 and 0.547 for the randomized and guided distributions respectively, confirming both *infinite* means and variances. Plots focusing on the left-hand tail show little evidence of heavy tails as seen in Fig. 4.5, with $\alpha$ estimates for the left hand Pareto distribution of 5.09 and 4.17 for the random and guided distributions respectively.

Heavy tails over multiple instances is of little use to randomized restart or SGMPCS trying to solve a single instance. As shown in Table 4.2 and Fig. 4.6 the results when broken down per instance are more varied, but still tend to confirm heavy tails for both cases. The column labeled ‘all’ represents fitting a distribution of all runs over all problems, as shown in Fig 4.4.

| instance | non-guided $\alpha$ | guided $\alpha$ |
|---|---|---|
| all | 0.728 | 0.547 |
| 0 | 0.586 | 1.00 |
| 1 | 0.61 | 0.755 |
| 2 | 0.676 | 0.579 |
| 3 | 0.532 | 1.62 |
| 4 | 1.14 | 1.78 |
| 5 | 0.483 | 0.718 |
| 6 | 0.714 | 1.43 |
| 7 | 0.658 | 3.42 |
| 8 | 0.239 | 0.194 |
| 9 | 0.582 | 1.09 |

Table 4.2: Estimates of $\alpha$ for guided and non-guided runs on ten 16x16 JSP instances.

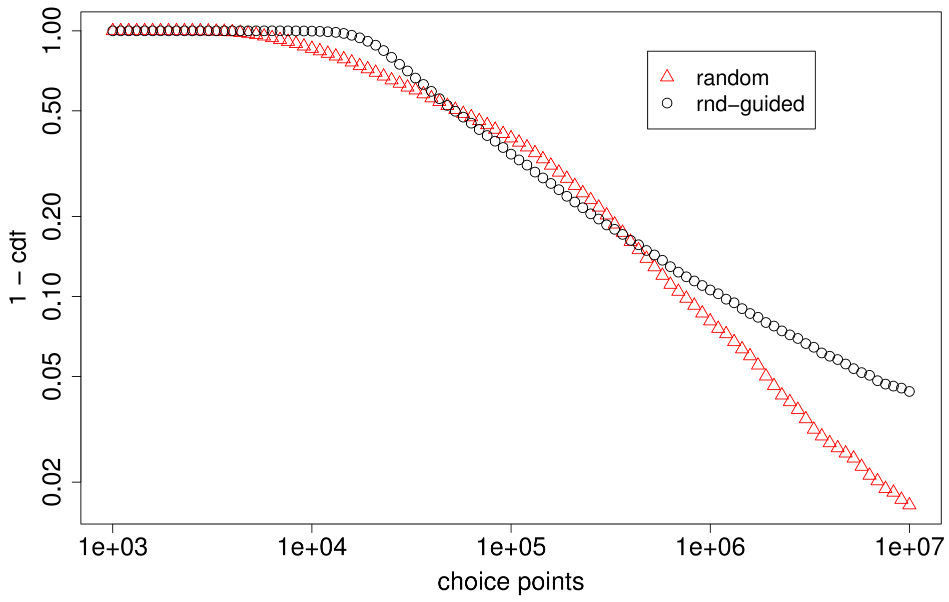

Figure 4.4: Right hand heavy tail plots of guided and non-guided (random) backtracking search on ten 16x16 JSP instances.

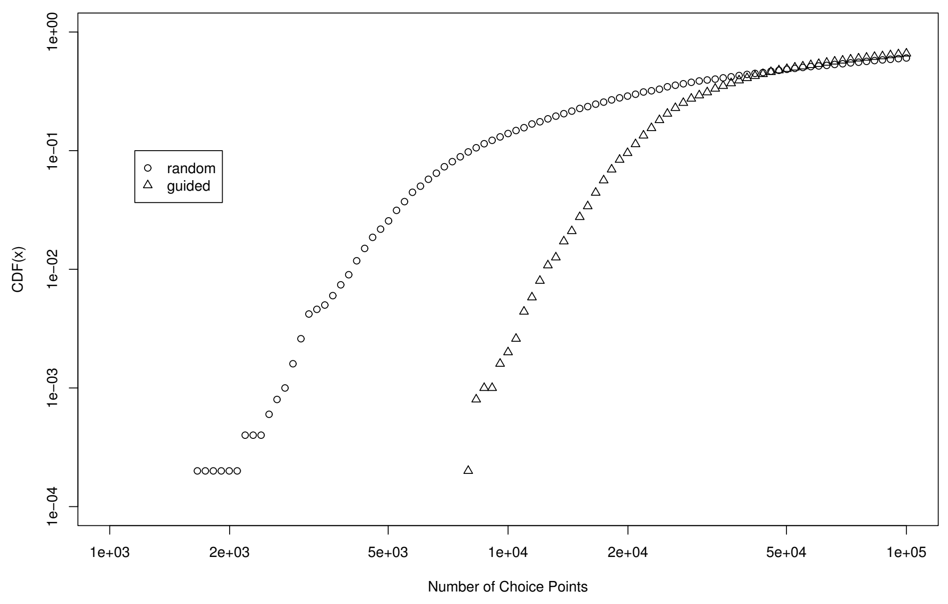

Figure 4.5: Left hand plots of guided and random backtracking search on ten 16x16 JSP instances.

## 4.7 Discussion

The aim of this chapter was to explore how heavy-tailed distributions in randomized backtracking search cost could help explain SGMPCS behaviour. We have shown that heavy-tailed behaviour can occur in job-shop scheduling optimization. As expected, the average performance of SGMPCS and randomized restart search both start to improve over standard chronological search at the size of problems where heavy-tailed behaviour appears. Finally we found the distribution of guided runs used by SGMPCS would also be heavy-tailed, suggesting a rapid restart strategy is just a necessary for SGMPCS as it is for the standard randomized restart algorithm.

### 4.7.1 Limitations

While we believe we have proved our case, various points regarding our study's limitations can be raised. These include experimental details added due to time constraints, complexities added by studying optimization, and our simplified definition of all guided runs.

Our study may have been weakened by two details added due to time constraints: only searching to a suboptimal solution and censoring measurements at 10,000,000 choice points. This limit censored less than 5% of all 10,000 guided runs on the largest size problem. Less than 2% of all non-guided runs on the size 16 instances were censored. Regarding searching to a suboptimal solution, a previous paper by Gomes and Selman [25] employ a similar technique in studying heavy tails in optimization. In retrospect, we should have chosen a percent to which the mean performance of SGMPCS clearly outperforms restart.

Restart techniques in optimization add extra benefits as well as complications to modeling their performance. At each restart, more is known about the bounds on the optimization function as the upper bound is reduced to the best found on the last iteration. Starting at the top of the search tree, this lower upper bound can by itself help find a better solution sooner through increased propagation. In satisfaction, there is nothing to optimize so each restart is the same as the last.[2] In contrast, the distribution of run-times that a restart technique samples from is likely to change as the bounds become more constrained.

How we defined the distribution of guided runs may also be a bit simplistic. A randomized run is defined over every possible guiding solution. Yet, as we have seen in SGMPCS, a single elite solution may remain in the elite set for a large portion of the overall search time. As well, not just any solution can be guiding solution, only better quality solutions are allowed into the elite set. Regarding the comparison of distributions for guided and non-guided runs, we may ask how different these heuristics really are? Either values are randomly chosen once at the start of search when the guiding solution is generated, or dynamically chosen at each choice point. This may alone explain for the differing results. When guided by a randomly generated solution, the same possibly bad decision will be made whenever search comes to a particular variable whereas in the random heuristic it has another chance each time. These heavier tails for guided runs may perhaps be explained by the formal model of heavy tails of Chen et al. [13]: the static random variable ordering when guiding by a solution leads the search tree to be more unbalanced, and hence more likely to exhibit heavy tails.

## 4.8 Conclusion

These experiments have confirmed that the distribution in run-times of the backtracking search on JSP instances can be heavy-tailed. It was shown that SGMPCS and randomized restart techniques perform better at the same instance sizes in which heavy tails start to appear. Guided runs used by SGMPCS were also shown to exhibit heavy tails.

While the tails for guided runs were heavier, this only means that using a rapid restart technique is even more necessary when guiding from a solution. The smaller probability mass on the left hand side, suggests that SGMPCS does not benefit more than the standard restarting technique from exploiting heavy tails. While the heavy tailed distribution explains why SGMPCS and randomized restart perform better than basic backtracking, we have not yet explained why SGMPCS can perform better than randomized restart.

One main claim of our thesis is that SGMPCS can perform better than restart by the benefit of being guided by good solutions. In the next chapter, this is investigated through a fitness-distance correlation study on the multi-dimensional knapsack satisfaction problem.

[2]This does not account for techniques which combine restarts with no good learning [3].

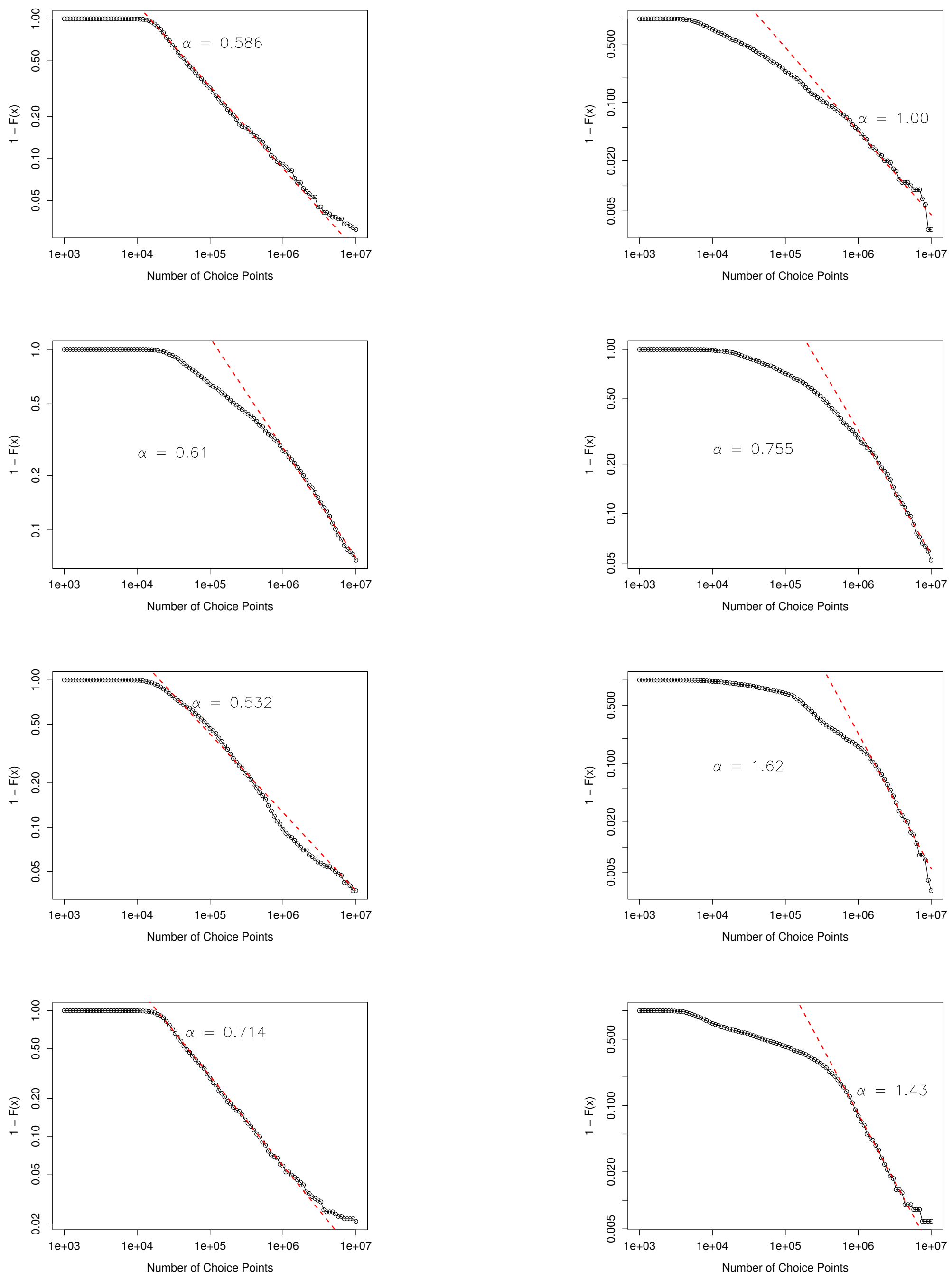

Figure 4.6: Log-log plot of the survival functions of backtracking search to find a 4% of optimal solution, **Left** guided by a random solution and **Right** using a random variable ordering for randomly generated order 16 JSP instances 0,1,3 and 6. Also plotted is the fitted Pareto distribution of the tail with the estimated parameter $\alpha$.

# Chapter 5

# Fitness-Distance Correlations in Solution Guided Multi-Point Constructive Search

According to our thesis, two factors have an impact on the performance of SGMPCS: the exploitation of heavy-tails, and the impact of revisiting elite solutions. In an attempt to explain the poor performance of SGMPCS on the multi-dimensional knapsack satisfaction problems of Chapter 3, in this chapter, we build a descriptive model of SGMPCS performance based on this second premise.

The core of the chapter is the investigation of the conjecture that SGMPCS performance, unlike that of randomized restart and chronological backtracking, is partially affected by the quality of the heuristic that is used to select the guiding partial solutions. When we artificially control the quality of the heuristic evaluation, we observe substantial performance differences. We then investigate two new heuristics for the multi-dimensional knapsack problem. The better heuristic results in significant gain in search performance and, more importantly, the observed performance differences among the three heuristics are consistent with the descriptive model. Approximately 44% of the variation in search performance can be accounted for by the quality of the heuristic.

In the next section, we discuss the goal of this chapter: to develop a descriptive model of SGMPCS based on fitness-distance correlation. Then, in Section 5.2, we revisit and discuss the initial empirical studies, demonstrating the poor performance of SGMPCS. Section 5.3 develops our descriptive model of SGMPCS performance. Section 5.3.4 proposes two new heuristic evaluation functions and empirically evaluates them. We discuss the implications and limitations of our study in Section 5.4.

## 5.1 Descriptive Models of Algorithm Behaviour

As discussed in Chapter 2, a descriptive model of algorithm behaviour is a tool used to understand why an algorithm performs as it does on a particular class or instance of a problem. There has been considerable work over the past 15 years in developing models of problem hardness [21, 61] as well as work that has focused more directly on modeling the behaviour of specific algorithms or algorithm styles. The work on heavy-tailed phenomenon [29, 38] models the dynamic behaviour of constructive search algorithms while local search has been addressed in a number of models–see [36] for a detailed overview.

In this chapter, we develop a static cost model with the goal of correlating problem instance features to algorithm performance. Our primary interest is to understand why SGMPCS outperforms or fails to outperform, other constructive search techniques. Specifically, we focus on multi-dimensional knapsack satisfaction problems, which as shown in Chapter 3, are solved just as poorly by SGMPCS as by randomized restart search. The approach we adopt is *fitness-distance analysis* [36], an *a posteriori* approach traditionally applied to local search algorithms. Local search algorithms move through the search space based on an evaluation of the quality of (suboptimal) "solutions" in the neighbourhood of the current solution. Neighbouring solutions are evaluated and, typically, the lowest cost solution is selected to be the next solution. In fitness-distance analysis, the quality of a solution (i.e., its *fitness*) is compared against its distance to the nearest optimal solution. Distance is measured as the minimum number of steps it would take to move from the solution in question to the nearest optimal solution. In problem instances where the search space and neighbourhood function induce a high *fitness-distance correlation* (FDC), the standard behaviour of moving to a solution with higher fitness will therefore tend to move the search closer to an optimal solution.

Standard constructive search techniques such as chronological backtracking, limited discrepancy search, and randomized restart do not exploit the fitness of sub-optimal solutions that are found during search. Even when there is a notion of sub-optimality, as in optimization problems, these techniques do not attempt to search in the "neighbourhood" of high quality solutions. There are, however, some algorithms that are based on constructive search such as ant colony optimization [20] and adaptive probing [57] that have been shown to be sensitive to FDC on optimization problems [10].

We test the hypothesis that SGMPCS is sensitive to fitness-distance correlation and that, therefore, its search performance can be partially understood by the FDC of a problem instance.

## 5.2 Initial Experiment

In this section, we present the details and results of our initial experiments of re-running the benchmark multi-dimensional knapsack problems of Chapter 3 with new default parameters and newer hardware and software. These results will be used as the basis of comparison for further experiments.

### 5.2.1 Experimental Details

We compare three search techniques: chronological backtracking (*chron*), randomized restart (*restart*) [29], and SGMPCS. In all algorithms the variable ordering is random. The value ordering for each algorithm, when not being guided by an elite solution, is also random. Any restart-based technique needs some randomization. The use of purely random variable and value ordering serves to simplify the experimental set-up.

*Restart* follows the same fail sequence as SGMPCS (see below) and initializes and maintains a set of elite solutions. However, it always searches from an empty solution (i.e., it is equivalent of SGMPCS with $p = 1$). Therefore, it has a small run-time overhead to maintain the elite set as compared with standard randomized restart.

All algorithms were implemented in ILOG Solver 6.3 and run on a 2GHz Dual Core AMD Opteron 270 with 2GB RAM running Red Hat Enterprise Linux 4.

**Parameter Values for SGMPCS** Chapter 3 examined the impact of different parameter settings. Here, we are interested in SGMPCS performance in general, and, therefore, adopt the parameters listed in Table 5.1 for all experiments. These parameters were chosen based the experiments of Chapter 3 that showed little performance variation for SGMPCS for different settings on multi-dimensional knapsack problems. Since we do not want the fail limit used to depend on the evaluation function—the original default polynomial sequence reset when a new best solution was found—we use the Luby fail sequence for these experiments. Due to the poor performance, in terms of run-time, of the original Luby sequence we follow [37] and multiply each limit by a constant, in our case 32 (i.e. 32, 32, 64, 32, 32, 64, 128, 32, ...).

| Fail Seq. | $\|e\|$ | p | Init. Fail Bound | Backtrack Method |
|---|---|---|---|---|
| luby32 | 8 | 0.5 | 1 | chron |

Table 5.1: Default parameter values for the experiments.

**Problem Instances** The same two sets of six problems from the operations research library[1] are used as in Chapter 3. The instances range from 15 to 50 variables and 2 to 30 dimensions.

For each problem instance, results are averaged over 1000 independent runs with different random seeds and a limit of 10,000,000 fails per run. For each run of each problem instance, we search for a satisfying solution.

**Heuristic Evaluation for SGMPCS** Following the idea of trying simple approaches before more complex ones, our initial heuristic evaluation, as was used in Chapter 3, is the number of unassigned variables. As described in Section 2.2.2, our elite solution candidates are dead-ends that either have one or more variables with an empty domain or break a constraint. When the solver encounters a dead-end, we simply count the number of unassigned variables and use that as the heuristic evaluation: the fewer unassigned variables, the better the dead-end. We make no attempt at a dead-end to assign any of the unassigned variables that have non-empty domains. We refer to this heuristic evaluation as $H_1$. This is the exact same heuristic used in Chapter 3.

### 5.2.2 Results

Table 5.2 compares the performance of chronological backtracking, randomized restart, and SGMPCS as defined above. Both SGMPCS and randomized restart perform poorly when compared to chronological backtracking. There does not seem to be a large difference between the performance of SGMPCS and randomized restart.

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

| | chron | | | restart | | | SGMPCS-$H_1$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | %sol | fails | time | %sol | fails | time | %sol | fails | time |
| mknap1-0 | 100 | **1** | 0.0 | 100 | 2 | 0.0 | 100 | 3 | 0.0 |
| mknap1-2 | 100 | **26** | 0.0 | 100 | 42 | 0.0 | 100 | 41 | 0.0 |
| mknap1-3 | 100 | **523** | 0.0 | 100 | 1062 | 0.0 | 100 | 924 | 0.0 |
| mknap1-4 | 100 | **15123** | **0.4** | 100 | 54635 | 1.5 | 100 | 44260 | 1.2 |
| mknap1-5 | 100 | **3271555** | **67.2** | 54.5 | 6885489 | 167.2 | 70.8 | 5573573 | 137.0 |
| mknap1-6 | 0.2 | 9990291 | **279.9** | 0.0 | 10000000 | 337.2 | 0.8 | **9958245** | 340.9 |
| mknap2-PB1 | 100 | **15223** | 0.3 | 100 | 42651 | 0.8 | 100 | 28770 | 0.6 |
| mknap2-PB2 | 100 | **3088092** | **54.1** | 80.3 | 4970049 | 102.0 | 88.1 | 3741187 | 77.8 |
| mknap2-PB4 | 100 | **10167** | 0.1 | 100 | 38474 | 0.5 | 100 | 28406 | 0.4 |
| mknap2-PB5 | 100 | **7011** | 0.1 | 100 | 16178 | 0.4 | 100 | 15077 | 0.3 |
| mknap2-PB6 | 100 | **16050** | **1.9** | 100 | 28964 | 3.8 | 100 | 25954 | 3.4 |
| mknap2-PB7 | 100 | **1472499** | **138.7** | 76.0 | 5374900 | 551.4 | 85.9 | 4113704 | 423.6 |

Table 5.2: Comparison of multi-dimensional knapsack results for chronological backtracking (*chron*), randomized restart (*restart*) and SGMPCS using the $H_1$ heuristic evaluation function.

## 5.3 Building a Descriptive Model

In this section, we develop a descriptive model of SGMPCS performance based on the fitness-distance correlation. We first define the measure of distance used and then present a deeper analysis of the SGMPCS results in the above table. We then build on the methodology of Beck & Watson [10] to create an artificial heuristic evaluation function that allows us to completely control the fitness-distance correlation of the problem instances. Experiments with this artificial heuristic demonstrate a strong interaction between FDC and search performance. Finally, we develop two new heuristic evaluation functions and examine their performance.

### 5.3.1 A Measure of Distance

A complete solution to a multi-dimensional knapsack problem can be represented by a binary vector $(x_1, ..., x_n)$ of the decision variables. The representation lends itself to using the Hamming distance as a measure of the distance between two (complete) assignments. This is the standard definition in fitness-distance analysis of local search algorithms.[2]

Our elite solutions are dead-ends and so may not be complete assignments (see Section 2.2.2). Therefore, we must adapt the Hamming distance to account for unassigned variables. A given dead-end with $m$ assigned variables, $m < n$, represents a set of $2^{n-m}$ points in the search space with varying distances from the nearest satisfying solution. If we assume a single satisfying solution to a problem instance (see below), then the distribution of distances for the sub-vector of unassigned variables follows a binomial distribution with a minimum sub-distance of 0 and maximum sub-distance of $n - m$. The mean of this distribution is $\frac{n-m}{2}$. We therefore calculate the distance from a dead-end to the satisfying solution as the mean distance

[2]SGMPCS does not move in the search space with the freedom of local search as it is constrained by a search tree. It may be the case, therefore, that a different definition of distance that takes into account the search tree may be more appropriate. We leave the investigation of such a distance function for future work.

of the points represented by the dead-end: the Hamming distance for the assigned variables plus one-half the number of unassigned variables. More formally, for a given elite solution candidate $S = (x_1, ..., x_m)$ and a satisfying solution $S^* = (x_1^*, ..., x_n^*), m \leq n$, the distance is calculated as follows:

$$D(S, S^*) = \sum_{1 \leq i \leq m} |x_i - x_i^*| + \frac{n-m}{2} \tag{5.1}$$

The normalized distance is $ND(S, S^*) = \frac{D(S,S^*)}{n}$.

### 5.3.2 Analysis of the Initial Experiments

As in Chapter 3, traces of SGMPCS-$H_1$ runs show that early in the search all the elite solutions have a heuristic evaluation of 0: all of the variables are assigned but the solution does not satisfy all constraints. The propagation tends not to result in domain wipe-outs but rather full assignments that break one or more constraints. The uniformity of the heuristic evaluation suggests that our simple heuristic evaluation is too coarse to provide useful guidance.

To quantify this observation, we calculate the heuristic evaluation and distance of each elite solution encountered during the search. In order to do this, we must first find all satisfying solutions to each instance. We did this using a small modification to the chronological back-tracking algorithm. To our surprise, each instance has a single satisfying solution, justifying our definition of $D$ above.

Figure 5.1 presents plots of the distance vs. the fitness for two of the problem instances. The plots for the other problem instances are almost identical. It is clear, that the heuristic evaluation provides almost no real heuristic information. These data were gathered by instrumenting the SGMPCS solver to record the fitness and distance from the known satisfying solution of each new entry to the elite set.

### 5.3.3 Manipulating the Fitness-Distance Correlation

Figure 5.1 is consistent with our conjecture that fitness-distance correlation may have a role in a descriptive model of SGMPCS performance. It provides, however, rather weak support: the absence of an FDC accompanies poor performance. A stronger test of the conjecture is to directly manipulate the FDC and observe the performance of SGMPCS. To do this, we adopt the technique introduced in [10] to artificially set the heuristic evaluation based on knowledge of the distance to the satisfying solution.

Let $D(S, S^*)$ be defined as in Equation (5.1). We define the heuristic evaluation of the satisfying solution, $S^*$, to be $h(S^*) = 0$. We set the heuristic evaluation, $h_{FDC+}(S)$, of an elite solution $S$ under perfect FDC equal to $D(S, S^*)$. Similarly, we set the heuristic evaluation $h_{FDC-}(S)$ of an elite solution with perfect negative FDC to be $(n - D(S, S^*))$. To generate instances with intermediate FDC, we interpolate between these two extremes as follows:

$$h(S) = \begin{cases} 0 & \text{if } S = S^* \\ \lceil \alpha \times h_{FDC+}(S) + (1-\alpha) \times RAND(S) \rceil & \text{if } S \neq S^* \wedge \beta = 0 \\ \lceil \alpha \times h_{FDC-}(S) + (1-\alpha) \times RAND(S) \rceil & \text{if } S \neq S^* \wedge \beta = 1 \end{cases} \tag{5.2}$$

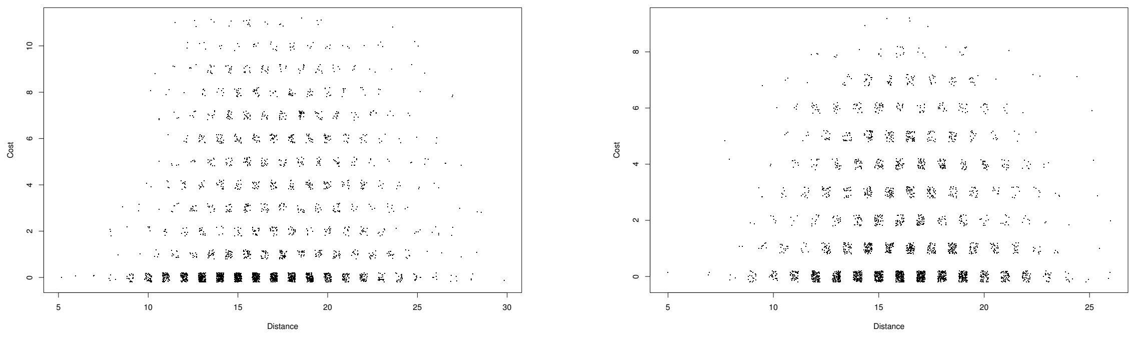

Figure 5.1: Plots of the heuristic evaluation (fitness) of each elite solution vs. its normalized distance from the unique satisfying solution for two of the multi-dimensional knapsack problem instances: Left: mknap1-5; Right: mknap2-PB7. A small noise component is added to the fitness and distance for purposes of visibility in the plot–this noise is not present in the data.

where $\alpha \in [0, 1]$, $\beta \in \{0, 1\}$, and $RAND(S) \in [0, n]$; the latter value is uniformly generated from the interval, using the bit vector $S$ as the random seed. The random component is added to achieve more realism in our model, while still manipulating the FDC. Clearly, when $\alpha = 0$, the heuristic evaluation is purely random. While $\alpha$ determines the strength of the FDC, $\beta$ is a two-valued parameter governing its direction: $\beta = 0$ and $\beta = 1$ induce positive and negative FDC, respectively.

The only difference with our initial experiments is that the heuristic evaluation is changed to Equation (5.2). For a single instance and each pair of values for $\alpha$ and $\beta$, we solve the instance 1000 times with different random seeds. Following Watson [67] we compare FDC against the log of search cost, in our case, the log of the number of fails to find a satisfying solution. Since our problems are of various sizes, the log of the mean number of fails of instance $p$ with $\alpha = a, \beta = b$, $\bar{F}_{p,a,b}$, is normalized with the log of the search cost of *chron* on the same problem ($C_p$) as follows:

$$N_{p,a,b} = \frac{log(\bar{F}_{p,a,b}) - log(C_p)}{log(C_p)}$$

For each problem and setting of $\alpha$ and $\beta$, FDC values are measured by collecting every unique elite solution over the 1000 iterations and taking the correlation between the evaluation function for each entry and its distance to the one known satisfying solution as defined in Equation 5.1.

Figure 5.2 shows the plot of FDC against normalized log of search cost for each problem instance and setting of $\alpha$ and $\beta$. The graph does not contain results for mknap1-0 and mknap1-2. As shown in Table 5.2, these are easily solved during the initialization phase of SGMPCS and so display no correlation with FDC. There is considerable noise for high negative values of FDC due to the fact that SGMPCS could not find a solution on a number of problem instances with high negative FDC, within the fail limit.

As seen in Figure 5.2, all problems show a downward linear trend in normalized log of search cost as the FDC increases. Where the data points cross the zero-line, signifies the FDC needed for SGMPCS to perform better than (chron) (in term of number of choice points). These results show, at least in this artificial setting, that fitness-distance correlation has a significant

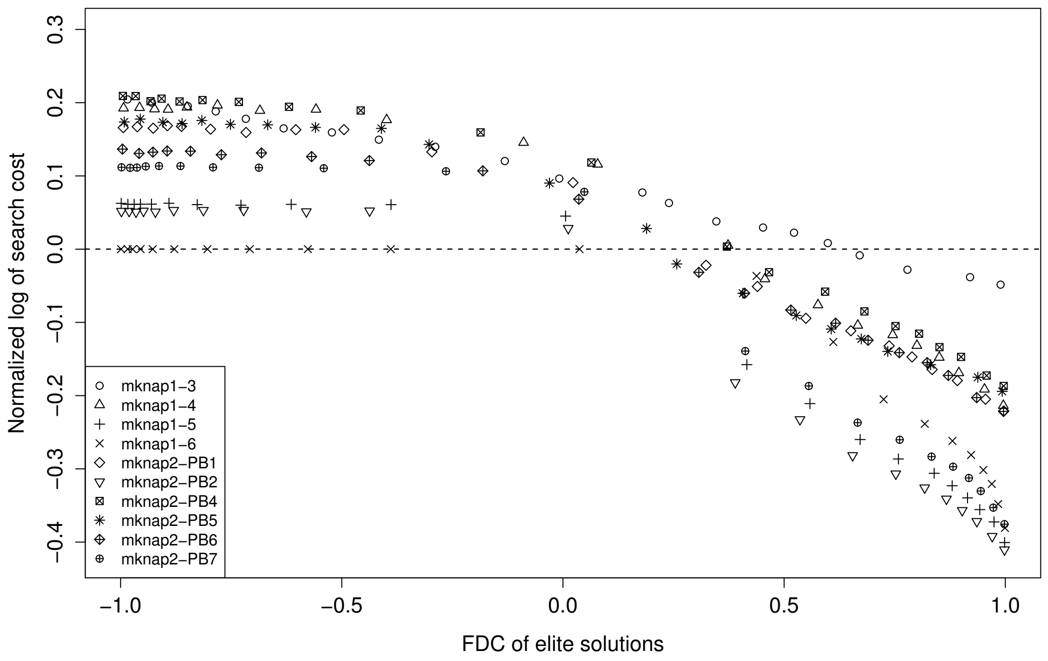

Figure 5.2: A scatter-plot of the measured fitness-distance correlation versus the normalized log of search cost for the artificial heuristic evaluation in Equation (5.2). The low positive values of search cost for high negative FDC stem from problem instances (and settings of $\alpha$ and $\beta$) for which SGMPCS could not find a solution. The graph does not contain the results for problem instances mknap1-0 and mknap1-2 as they are trivially solved.

impact on SGMPCS performance.

### 5.3.4 Toward Better Heuristic Evaluations

Figures 5.1 and 5.2 show that one possible explanation for the poor performance of SGMPCS-$H_1$ is the low fitness-distance correlation. The results of the experiment that manipulated the FDC demonstrated that the performance of SGMPCS is sensitive to the FDC, at least in an artificial setting. In this section, we develop two new heuristic evaluation functions. Our goal is to demonstrate that, in a non-artificial setting, the FDC induced by the heuristic evaluation function has an impact on the search performance of SGMPCS.

The intuition behind both of the new heuristic evaluation functions is to include additional knowledge about the quality of the solution. In particular, we wish to create a finer heuristic evaluation that is able to better distinguish among the elite solutions (i.e., we would like fewer of the elite solutions to have a heuristic evaluation of zero than with the $H_1$ function). Our main goal in proposing these heuristics is to evaluate the relationship between FDC and search performance. We expect that these heuristics will result in a different FDC and wish to test if this leads to a difference in performance.[3]

- $H_2$- Recall (Section 3.2.1) that our CSP model of the multi-dimensional knapsack assumed that the value of the most profitable knapsack, $P^*$, is known. This knowledge is used in the constraint, $P = P^*$, but not otherwise exploited above. Here, we define $H_2 = |P^* - P|$.

- $H_3$- Some preliminary experiments showed that even with $H_2$, the elite pool often stagnated on a set of elite solutions with a zero heuristic evaluation that break one or more constraints. Therefore, in order to further refine the heuristic evaluation, we choose to use the number of broken constraints as a tie-breaker: $H_3 = H_2 + |V|$ where $|V|$ is the number of constraints violated by the (partial) assignment.

It should be noted that the only difference among the $H_1$, $H_2$, and $H_3$ models is the heuristic evaluation function. In particular, the constraint model is identical in all three models. We now solve each of the problem instances 1000 times (with different random seeds) with each heuristic evaluation function. The other experimental details are the same as in Section 5.3.3.

#### 5.3.4.1 Results

Figure 5.3 presents distance vs. fitness plots of the same two problem instances as Figure 5.1 using the two new heuristic evaluation functions. From $H_2$ to $H_3$ the evaluation functions provide slightly better predictions of the partial solution's distance to the satisfying solution. This is evident from the correlations measured (mknap1-5: $r_{H2} = 0.265$, $r_{H3} = 0.312$; mknap2-PB7: $r_{H2} = -0.026, r_{H3} = 0.292$) and can be seen on the graphs in the slightly less uniform distribution of points.

Figure 5.4 displays the scatter plot of the normalized search performance versus FDC for all of our heuristic evaluation functions together with the minimum mean squared error line. As above, we do not include mknap1-0 and mknap1-2. Although limited by our small number of instances, the plot clearly shows a trend of better search cost with higher FDC values. The $r^2$ value is $0.568$ ($r = -0.754$). If we further omit mknap1-3 and mknap1-6, $r^2 = 0.631$. Both these instances are outliers, the former because it is easy and the latter because it is only solved in 14 of 4000 runs (over all SGMPCS algorithms and *chron*) and so exhibits errors due to hitting the overall fail-limit.

Table 5.3 displays the performance of each SGMPCS variation. For completeness we repeat the results for chronological backtracking and SGMPCS-$H_1$ from Table 5.2. While not clearly superior, SGMPCS-$H_3$ is competitive with *chron* overall. For the harder instances (i.e., mknap1-5, mknap2-PB2, mknap-PB6, and mknap2-PB7 where *chron* has a high number of fails) SGMPCS-$H_3$ is 1.5 to 8 times better than *chron* in terms of the number of fails.

[3]It does not seem likely that either of these heuristics will be useful, in general, for solving multi-dimensional knapsack problems because both make use of knowledge of the value of the most profitable knapsack.

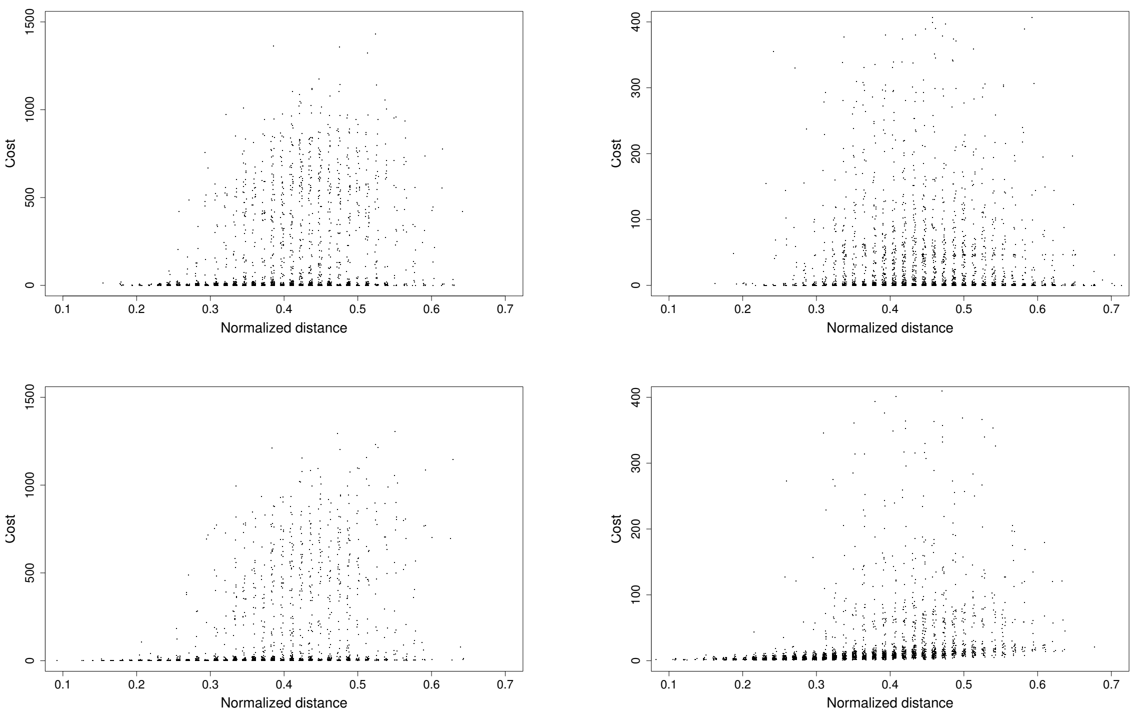

Figure 5.3: Plots of the heuristic evaluations $H_2$ (top) and $H_3$ (bottom) of each elite solution vs. its normalized distance from the unique satisfying solution for two of the multi-dimensional knapsack problem instances: Left: mknap1-5; Right: mknap2-PB7. A small noise component is added to the fitness and distance for purposes of visibility in the plot–this noise is not present in the data.

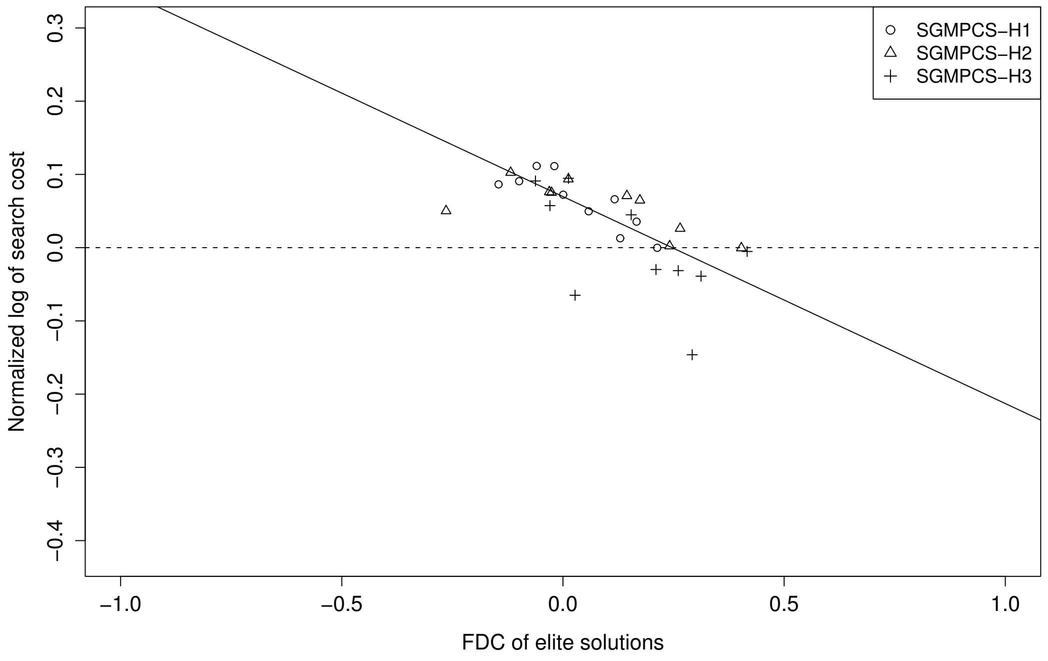

Figure 5.4: A scatter-plot of the measured fitness-distance correlation versus the normalized log of search cost for three heuristic evaluation functions: $H_1$, $H_2$, $H_3$. The graph does not contain the results for problem instances mknap1-0 and mknap1-2 as they are trivially solved.

| | chron | | | SGMPCS-$H_1$ | | | SGMPCS-$H_2$ | | | SGMPCS-$H_3$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %sol | fails | time | %sol | fails | time | %sol | fails | time | %sol | fails | time |
| mknap1-4 | 100 | 15123 | 0.4 | 100 | 44260 | 1.2 | 100 | 29895 | 0.9 | 100 | **11349** | 0.5 |
| mknap1-5 | 100 | 3271555 | **67.2** | 71 | 5573573 | 137.0 | 77 | 4839688 | 126.2 | 98 | **1824457** | 71.6 |
| mknap2-PB1 | 100 | **15223** | 0.3 | 100 | 28770 | 0.6 | 100 | 28405 | 0.6 | 100 | 23445 | 0.7 |
| mknap2-PB2 | 100 | 3088092 | **54.1** | 88 | 3741187 | 77.8 | 92 | 3191853 | 71.2 | 98 | **1933160** | 60.9 |
| mknap2-PB4 | 100 | **10167** | 0.1 | 100 | 28406 | 0.4 | 100 | 24112 | 0.4 | 100 | 24370 | 0.5 |
| mknap2-PB5 | 100 | **7011** | 0.1 | 100 | 15077 | 0.3 | 100 | 13747 | 0.3 | 100 | 11650 | 0.4 |
| mknap2-PB6 | 100 | 16050 | 1.9 | 100 | 25954 | 3.4 | 100 | 26082 | 3.4 | 100 | **8554** | 1.8 |
| mknap2-PB7 | 100 | 1472499 | 138.7 | 86 | 4113704 | 423.6 | 86 | 4287571 | 447.7 | 100 | **184443** | **32.8** |

Table 5.3: Comparison of multi-dimensional knapsack results for chronological backtracking (*chron*), and SGMPCS using the three heuristic evaluation functions $H_1, H_2, H_3$.

## 5.4 Discussion

In this chapter, we addressed the question of developing an understanding of why SGMPCS performs as it does on constraint satisfaction problems. We demonstrated that the correlation between the heuristic evaluation of an elite solution and its distance to the satisfying solution, is well-correlated with the search performance of SGMPCS. Standard constructive search approaches such as chronological backtracking, randomized restart, and limited discrepancy search make no use of such heuristic information.

### 5.4.1 Limitations

There are a number of limitations to the study in this chapter. First, it is a case study of 12 problem instances of one type of problem. While we believe these results are likely to be observed for other problems instances and types, a larger study is needed. Accordingly, the next chapter does provide a larger study on various search space features, including FDC, for job-shop scheduling. Second, the poor performance of randomized restart on the multi-dimensional knapsack problems suggests that they do not exhibit heavy-tails. In Chapter 4, we showed evidence that SGMPCS is able to exploit heavy-tails in the same way as randomized restart. Therefore, a full descriptive model of SGMPCS must address the impact of heavy-tailed distributions. In fact, one of the reasons that the multi-dimensional knapsack problem was chosen for this case study was precisely because we did not have to address the impact of heavy-tailed distributions. Third, it should be acknowledged that multi-dimensional knapsack problems are strange CSPs since the underlying problem is an optimization problem and we exploit this in formulating the new heuristic evaluation functions in Section 5.3.4. Our original motivation for choosing to apply SGMPCS to a CSP version of multi-dimensional knapsack was simply because [54] did so and showed poor performance for randomized restart. Given the relationship between randomized restart and SGMPCS, this appeared to be a fertile choice. There remains some uncertainty regarding the application of the FDC-based descriptive model of SGMPCS performance on more "natural" CSPs. Nonetheless, our model makes clear, testable hypotheses that can be evaluated in future work. Finally, as a descriptive model, the work in this chapter does not, on its own, produce a clear benefit for constraint solvers. That is, we have not demonstrated any improvement on the state-of-the-art for any problem classes. That was not our aim here. What we have done is provided a deeper understanding of the performance of SGMPCS and a potential new source of search guidance for CP search.

## 5.5 Conclusion

In this chapter, steps were taken in understanding the search behaviour of Solution Guided Multi-Point Constructive Search (SGMPCS). Using fitness-distance analysis, a technique common in the metaheuristic literature [36], a descriptive model of SGMPCS search behaviour on multi-dimensional knapsack satisfaction was developed. Empirical results, both in an artificial context and using three different heuristic evaluation functions, demonstrated that the correlation between the heuristic evaluation of a state and its proximity to the satisfying solution has a strong impact on search performance of SGMPCS. This (partial) descriptive model is important for three main reasons:

1. It makes strong, testable predictions about the behaviour of SGMPCS on other constraint satisfaction and optimization problems.

2. It provides a clear direction for improving SGMPCS search performance: the creation of, perhaps domain-dependent, heuristic evaluation functions for partial search states that are well-correlated with the distance to the nearest solution.

3. It re-introduces a heuristic search guidance concept to the constraint programming literature. Though guidance by heuristic evaluation of search states is common in metaheuristics, general AI search (e.g., $A^*$ and game playing), and best-first search approaches, it does not appear to have been exploited in constructive, CP search. We believe this is an important direction for further investigation.

In the next chapter, we explore descriptive models of SGMPCS further by evaluating how well various measures of the search space, including FDC, can predict SGMPCS search cost on the job-shop scheduling problem.

# Chapter 6

# Static Cost Models of Solution Guided Multi-Point Constructive Search

## 6.1 Introduction

The last chapter showed a link between a feature of the search space, fitness-distance correlation, and SGMPCS performance. The purpose of this current chapter is to investigate how SGMPCS performance may be related to other search space features. This is accomplished through a set of experiments analogous to those of Watson et al. [70] on job-shop scheduling problems. Instead of investigating the correlation between search space features and tabu-search performance, we look at the correlation of the exact same measured features and the performance of SGMPCS.

In this chapter, we first review past work of descriptive models of algorithm behaviour in local search and describe the experiments and results of Watson et. al in detail. Then, in Sections 6.3 and 6.4, we show how the various search space features measured by Watson et al. correlate with SGMPCS performance on the same set of problem instances. In Section 6.5, we discuss the significance of our results and limitations of our experiments.

## 6.2 Background

An open problem in search algorithm research is to explain the large variation in search costs observed between problem instances [47, 21, 52, 60]. The focus has been predominantly in local-search algorithms where there is still a lack of understanding of why local search procedures work as well as they sometimes do. A way of reaching this better understanding has been to investigate how different search space features induced by an instance affect a search algorithm's performance.

Watson et al. [70] took four features used in past studies [14, 52, 45, 60] and investigated their accuracy in predicting the search cost for of tabu search on the job-shop scheduling optimization problem. The four features were the number of optimal solutions, backbone size, distance between local optima, and distance between local optima and the nearest optimal solution. Watson et al. refer to the use of these various search space features to account for the variability of search cost between instances as *static cost models* of problem difficulty. By static they are referring to the fact that the features are largely independent of the individual algorithm dynamics, relying on unchanging features of the search space. In contrast, dynamic cost models which do rely on dynamic features of specific algorithms—such as the set of solutions encountered by an algorithm—are investigated for tabu search and job-shop scheduling in [67]. The following static cost models are investigated here:

**Number of Optimal Solutions $|optsols|$:** One of the earliest static cost models of local search investigated was the number of optimal solutions by Clarke et al. [14]. Intuitively, the more often solutions occur in the search space, the quicker it should be for a local search procedure to encounter one. Yet, on satisfiable problems, local search difficulty decreases past the phase transition peak even as the number of solutions continues to drop [60, 74].

**Backbone Size $|backbone|$:** Used initially by Parkes [52] to explain the search cost peak of local-search SAT solvers, a backbone of a SAT instance is the set of variables which always have the same value in any satisfying solution. As the backbone size becomes larger, solutions will necessarily cluster in the search space and hence be more difficult for local search procedures to locate. Parkes observed that many large backbone instances start to appear at the critically constrained region. Parkes also observed that when backbone size is fixed, difficulty always decreases as constrainedness increases.

**Distance Between Local Optima $\overline{loptdist}$:** It has been observed that many local search algorithms consist mainly of movement from one local optima to another [45, 70]. Seen as such, local search performance should be related to how large this area of local optima is. Mattfeld [45] first used this measure to explain the differences in difficulty between general JSPs, where the ordering of operations are chosen at random, and workflow JSPs, where a structure is imposed on the routing orders. In workflow problems, the $m$ resources are split into $q$ ordered subsets of $m/q$ resources, and each job needs to visit the resources in an earlier subset (in a randomly generated order for each job), before visiting the next. Watson et al. and Mattfeld used $q = 2$.

**Distance Between Quasi-Solutions and the Nearest Optimal Solution $\bar{d}_{lopt-opt}$:** Singer et al. [60] observed in local search for SAT, that search consisted mainly of movement from near-solutions found early in the search, to the satisfying solution. They measured the average distance between these *quasi-solutions* (assignments where only 5 of the 100 clauses were unsatisfied) and the nearest satisfying solution, and found that it correlated well with local-search performance.

**Fitness-Distance Correlation FDC:** While not included by Watson et al., as mentioned in Chapter 5, the correlation between the evaluated cost or fitness of a solution and its true distance to an optimal solution has been used in past studies [64] to predict problem difficulty. Watson explains [67, p. 63] that FDC was not explored because he had no reason believe FDC could affect problem difficulty for non-adaptive local search algorithms. Since SGMPCS is an adaptive search algorithm, and because of the positive results of Chapter 5 we also include FDC here.

### 6.2.1 Problem Difficulty for Tabu Search in Job-Shop Scheduling

As this chapter follows directly from the paper by Watson et al. [70], we recount the details and results of their relevant experiments. The purpose of these experiments was to investigate various search cost models of problem difficulty of tabu search on a set of 'typical' instances of the general JSP.

#### 6.2.1.1 Problem Instances

Watson et al. used 6x4 and 6x6 general JSP instances: 6 jobs and 4/6 machines. Such small problem sizes were used because some of the static models require all optimal solutions to be enumerated, and the number of solutions can quickly grow into the billions at larger sizes. The 1000 instances were generated for each size with an earlier generator [69], in which the routings through the machines were randomly generated and the operation durations were independently and randomly drawn from [1, 99]. See Section 2.1.1 for further information on this problem.

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

#### 6.2.1.2 Static Cost Models

The static models of search cost investigated by Watson et al. attempted to predict search cost from a search space feature of an instance. The accuracy of these models was quantified by the $r^2$ value of the linear regression model between the search space feature of an instance, and the log of the median search cost to find an optimal solution $log(cost_{med})$. We now recount how each feature was measured, and the resulting accuracy of their models on the 6x6 JSPs.

**Number of Optimal Solutions $|optsols|$:** Watson et. al. enumerated all solutions to each instance by using a constructive search algorithm [9]. They found a weak correlation ($r^2 = 0.2223$) between the log of the number of solutions and $log(cost_{med})$ for 6x6 JSPs.

**Backbone Size $|backbone|$:** The backbone of an instance is the set of solution components that have the same value in all solutions. This requires all solutions to be enumerated, as well a definition of a solution component. Backbones of JSP instances were measured using a solution's *disjunctive graph* representation [11]. A solution to a 6x6 JSP instance in this representation consists of sets of $\binom{6}{2}$ Boolean variables for each machine. The variables represent the precedence relations for all pairs of operations that run on the same machine. The backbone size of an instance is the proportion of variables that have the same value in all optimal solutions: the proportion of paired orderings which are the same in each optimal solution. Watson et al. also found a weak correlation between log of median search cost and the square of backbone size ($r^2 = 0.2331$). This was a very similar correlation as found for $|optsols|$, and was explained through the very high negative correlation between these two features ($r = -0.9103$).

**Average Distance Between Local Optima $\overline{loptdist}$:** The final three search space features required a definition of distance between solutions and a random sampling of local optima.

Watson et al. defined the distance between two solutions, $s_1$ and $s_2$, as the value of Equation 6.1 [45], where $precedes_{ijk}(s)$ is the predicate of whether job $i$ is processed before job $j$ on machine $k$ on solution $s$, and $\oplus$ is the Boolean XOR operator. The actual distance used for the search space features was normalized: $\bar{D}(s_1, s_2) = 2D(s_1, s_2)/mn(n-1)$.

$$D(s_1, s_2) = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} precedes_{ijk}(s_1) \oplus precedes_{ijk}(s_2) \tag{6.1}$$

5000 random local optima were generated for each instance using a hill climbing procedure with a random starting point and the N1 JSP move operator [40]. Watson et al. found a relatively low $r^2$ value (0.2223) for the static cost model using the average distance between the 5000 randomly generated local optima.

**Distance Between Local Optima and Nearest Optimal Solution $\bar{d}_{lopt-opt}$:** Watson et al. adapted Singer et al.'s quasi-solution measure for JSP optimization by measuring the average distance between random local optima and the nearest optimal solution $\bar{d}_{lopt-opt}$. This measure used the 5000 local optima, and all optimal solutions generated for each instance as described earlier. The correlation found between the log of median tabu search cost and the square root of $\bar{d}_{lopt-opt}$ was the best of all static models investigated ($r^2 = 0.6541$).

## 6.3 Experiment 1: Evaluation of Static Cost Models

In the following experiments, we correlate SGMPCS search cost against the search space featured measured by Watson et al. [70].[1]

### 6.3.1 Problem Instances and Search Space Features

The same 1000 6x6 JSP instances from Watson et al. are used in these experiments. The search space measurements for each instance, as described earlier, were taken directly from Watson et al.'s experiments. Although FDC did not appear in [70], fitness-distance correlations were still measured using the 5000 local optima for each instance, and provided to us.

### 6.3.2 Algorithms

For means of comparison, the cost of finding an optimal solution is measured for three algorithms: SGMPCS, randomized restart (Restart), and standard chronological search (Chron). The last two algorithms are implemented through special parameter settings of SGMPCS. As in previous chapters, randomized restart is replicated by never guiding by a solution, $p = 1$. Chronological search is implemented with SGMPCS using $p = 1$ along with a fail sequence that never restarts search: $\{\infty\}$. These two settings result in one backtracking search which terminates when the optimal solution is found. SGMPCS always guides by a solution, $p = 0$, and uses the Luby [43] fail sequence: $\{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 4, 8, \cdots\}$.

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

| | fail sequence | $\lvert e\rvert$ | $p$ | Init. Fail Bound | Backtrack Method |
|---|---|---|---|---|---|
| SGMPCS | Luby | 1 | 0 | 1 | chron |
| Restart | Luby | 1 | 1 | 1 | chron |
| Chron | $\infty$ | 1 | 1 | 1 | chron |

Table 6.1: Parameter values for the experiments where Restart and Chron are implemented through special parameter settings of SGMPCS

Displayed in Table 6.1 are the parameter settings used by each algorithm. Again from past good performance and as a means to simplify SGMPCS, only one elite solution is maintained. Due to the small size of problem used, a fail limit of 1 is used in all cases to generate the initial elite solution. In the Restart and Chron algorithms, this initial elite solution is only used to set the initial upper bound on the makespan.

The same 10% randomized texture-based heuristic [9] and constraint propagation techniques for scheduling [51, 41, 42] used in Chapter 4 are used again here for all algorithms. All algorithms were implemented in ILOG Scheduler 6.3 and run on a 2GHz Dual Core AMD Opteron 270 with 2GB RAM running Red Hat Enterprise Linux 4.

Since all algorithms contain a degree of randomness, search cost for each instance and algorithm is measured over 1000 independent runs with different random seeds. The measure of search cost for each instance ($cost_{med}$) is the median number of choice points—over the 1000 independent runs—needed to *find*, but not prove, the optimal solution.

### 6.3.3 Initial Results

Various statistics of $cost_{med}$ over the 1000 instances are shown at the top of Table 6.2. Scatter plots of the first experiments are displayed in the left sides of Figures 6.1–6.5. In all plots, the y-axis is on a log scale, and the least-squares fit line is included. Pearson $r$ and $r^2$ values are shown in the top half of Table 6.3.

Log-log scatter plots of $|optsols|$ versus $cost_{med}$ for the three algorithms appear on the left side of Figure 6.1. The low $r$ values of -0.1829, -0.1814, and 0.0975 for SGMPCS, Restart and Chron respectively suggest the log of the number of solutions is a poor static cost model of JSP difficulty for any of the three constructive search algorithms.

Similar, inverted, results can be seen for $|backbone|^2$ in Figure 6.2 with $r$ values of 0.2094,

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

Table 6.2: Mean, median and standard deviation of $cost_{med}$ for the original and weakened versions of the three algorithms over the 1000 instances.

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

Table 6.3: Pearson's correlation coefficients $r(r^2)$ between $log(cost_{med})$ and the various search space features for the original three algorithms, the three algorithms with weakened propagation, and the original results for tabu search by Watson et al. [70].

0.2055, and -0.0123 for the same three algorithms. Since we are using the exact same instances as Watson et. al, this similarity can be attributed to the high correlation between $log(|optsols|)$ and $|backbone|^2$ ($r = -0.9103$) first noticed by Watson et al. [70].

More positive results can be seen for the average distance between local optima $\overline{loptdist}$. Scatter plots for each algorithm (Figure 6.3 Left) show a slight upward trend. $r$ values of 0.6445, 0.6536 and 0.5856 for SGMPCS, Restart and Chron respectively are all larger than that found by Watson et al. for tabu search (0.5238). Surprisingly, the $r$ values for all algorithms are quite similar.

Scatter plots of the square root of the distance between local optima and the nearest optimal solution $\sqrt{\bar{d}_{lopt-opt}}$ versus $cost_{med}$ (log scale) appear in the left of Figure 6.4. $r$-values of 0.6026, 0.6150, 0.4230 for the three algorithms are slightly less than those found for $\overline{loptdist}$.

Scatter plots of FDC versus log-scale $cost_{med}$ for the three algorithms are shown on the left side of Figure 6.5. $r$ values of -0.2025, -0.2290, and -0.1577 for the three algorithms suggest that FDC is a poor predictor of search cost for these instances.

## 6.4 Experiment 2: SGMPCS with Weaker Propagation

From the results seen so far, there was a concern that the algorithms were behaving much too similar to each other because of the small problem size. The thought was that in all cases, the algorithms were mainly finding the optimal solutions through the powerful propagation techniques of constraint programming, and less through the individual backtracking techniques of the varying algorithms. Gomes et al. [27, p. 81] noted a similar experience on smaller quasi-group with holes problems. When the all-different global constraints were used, the heavy-tailed distribution in run-times disappeared, which they attributed to a lack of balance between the propagation and backtracking parts of search.

In this second experiment, all algorithms are modified in an attempt to bring a similar balance between the effects of search and propagation on these smaller instances by changing the variable ordering and level of constraint propagation. The variable ordering is changed from a randomized texture heuristic to a completely random variable ordering. The enforcement level on the precedence and capacity constraints are set back to the default level used by the ILOG Scheduler [58], down from the medium-high level used previously for SGMPCS. This lower level does not maintain the global constraint on precedence, and removes the edge-finder algorithm [51] from the unary resource constraint enforcement. The weakened Restart$_{weak}$ and Chron$_{weak}$ algorithms are implemented in the exact same way using the weakened version of SGMPCS with the parameters in Table 6.1. All other experimental details are the same.

### 6.4.1 Results

As shown in Table 6.2, for the weakened algorithms, SGMPCS now outperforms Restart, yet Chronological still performs best in both cases. As seen in Chapter 4, our problem size appears to be too small for SGMPCS to outperform standard backtracking.

Scatter plots, and least-squares fit lines, of the search space features versus the median search cost (log scale) of the weakened algorithms appear on the right sides of Figures 6.1–6.5. The plots appear similar to the plots for the original algorithms, although the range of the log-scale y-axis is now ten times larger. Pearson $r$ and $r^2$ values are displayed under the original results in Table 6.3.

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

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

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

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

## 6.5 Discussion

Through the investigation of correlations of SGMPCS performance with the search space features measured by Watson et al. [70] some interesting relations are shown. On *all* constructive algorithms investigated, results were surprisingly similar to those found by Watson et al. with tabu search. That these relations can also be seen in parameter settings of SGMPCS that never guide from solutions (Restart) and never restart (Chron) is quite unexpected and makes any conclusions from this study unclear.

Although they are still quite similar in many regards, the accuracy of the various static cost models did vary from those found by Watson et al. for tabu search. Both $|optsols|$, and $|backbone|^2$ models are much less accurate for SGMPCS search cost, even though Watson et al. conclude they were also weak models for tabu search. In contrast, $\overline{loptdist}$ is a better predictor of search cost for all constructive algorithms used here than it was for tabu search. The best predictor of search cost for tabu search, $\sqrt{\bar{d}_{lopt-opt}}$, while giving the second to best models of search cost here, was less accurate for SGMPCS than it was for tabu search. Although FDC was not included by Watson et al., the FDC models were slightly more accurate for weakened versions of SGMPCS and Chron than they were at predicting tabu search cost.

By weakening the propagation we do see an increase in the correlation with all the search space features. This suggests that we may have partially succeeded in balancing propagation with the various backtracking techniques. Yet, chronological search still outperformed by Restart and SGMPCS. This suggests that the problem size, 6x6, may still be too small for SGMPCS to perform well, and that a larger size is needed to investigate SGMPCS behaviour. A slightly greater correlation for the feature which best predicted tabu local search difficulty, $\sqrt{\bar{d}_{lopt-opt}}$, may be seen as evidence, although not very strong evidence, that SGMPCS is behaving more like local search.

If we argue that the correlations seen with local search related search space features suggest that SGMPCS is performing local search, we then must also argue that Restart and Chron, due to the similar (and even higher) correlations seen for these algorithms, also perform local search. And perhaps, in some sense, they do. All optimization algorithms used in this chapter find better makespans in a step-wise fashion. In SGMPCS, the best solution found is used at each restart to guide the search. With each iteration in Restart, search is not guided by a solution, but the bounds on the makespan are decreased to the best one found so far. Even though Chron never restarts, the bounds on the optimization criterion are still lowered whenever a better solution is encountered. Whenever this criterion is lowered, one changes the set of solutions one is searching through. In a crude sense, all algorithms are performing movement between *very* large neighborhoods defined by the current bounds on the makespan. This idea is just speculation and no concrete explanation of our results exist at this time. Further studies that may aid in understanding these results are discussed in the next chapter.

One rather puzzling result of the second experiment results is that the correlation with FDC is strongest with Chron. One would expect the chronological algorithm would be least affected by how sub-optimal solutions are evaluated. This may again be due to the step-wise decreasing of the optimization function. Or, perhaps there is another factor, not measured in these experiments, which is associated with both FDC and search cost that can better explain our results.

Given the results of Chapter 5, it is interesting that FDC does not account for much of the variability of SGMPCS, although differences in how this correlation was measured may help explain the results. In Chapter 5, we were looking at FDC values between algorithms with different evaluation functions instead of the variability within one algorithm on many random instances. As well, in Chapter 5, we computed FDC using solutions that had ever entered the elite set, not N1-operator local optima. Using solutions encountered during search is quite similar to a technique later used by Watson [67] to create an even more accurate *quasi-dynamic* model of tabu search problem difficulty. We speculate that more accurate models would result if the relevant search space features were defined and measured using solutions encountered by SGMPCS. That we found any relationship at all using search space features based on hill climbing local optima is itself a bit surprising.

## 6.6 Conclusion

By evaluating the static cost models from [67] on three constructive search algorithms, we were able to find some correlations between SGMPCS behaviour and various search space features. Models of tabu search behaviour were similarly—and some times better—able to predict constructive search cost. We believe this is the first time such an analysis has been applied to constructive search procedures. Unexpectedly, similar correlations were found for both randomized restart and standard backtracking algorithms. While also interesting, these similar results remain mostly unexplained, and limit us from an argument that SGMPCS is behaving more like local search.

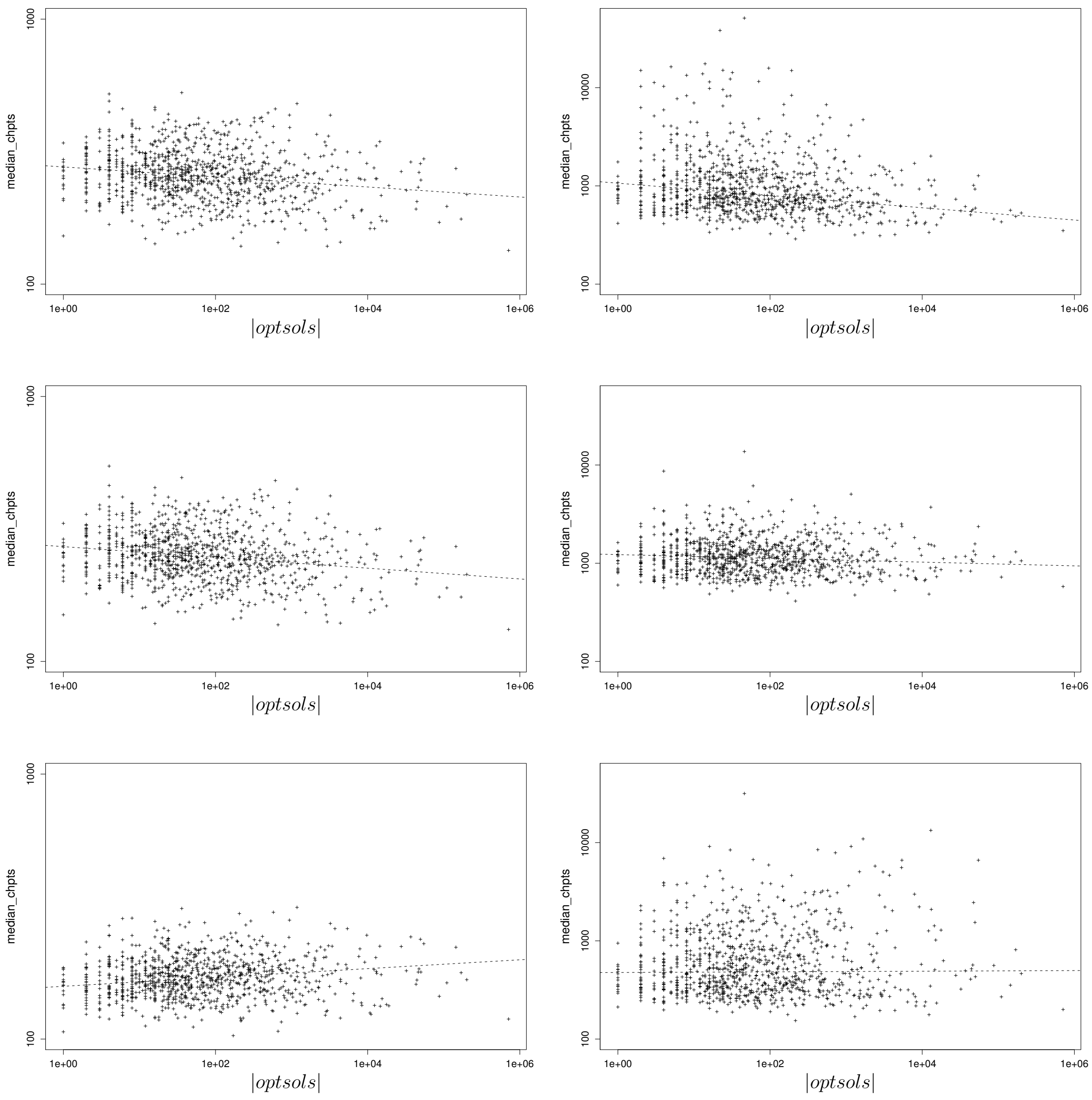

Figure 6.1: Log-log scatter plots and least-squares best-fit lines of $|optsols|$ versus $cost_{med}$ for the original **Left** and weakened **Right** versions of SGMPCS **Top**, Restart **Middle**, and Chron **Bottom**.

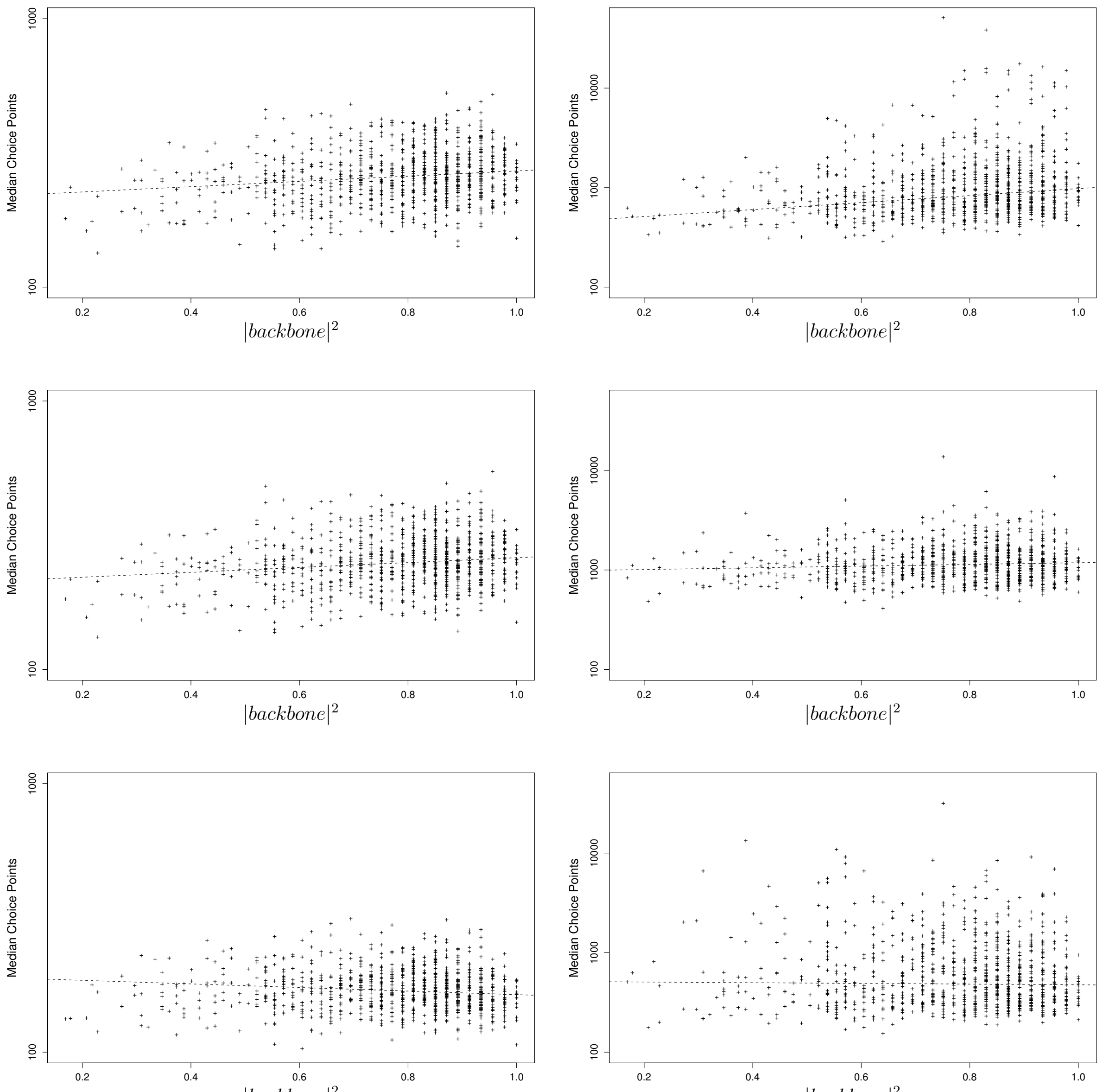

Figure 6.2: Scatter plots and least-squares best-fit lines of $|backbone|^2$ versus $cost_{med}$ for the original **Left** and weakened **Right** versions of SGMPCS **Top**, Restart **Middle**, and Chron **Bottom** (log-scale y-axis).

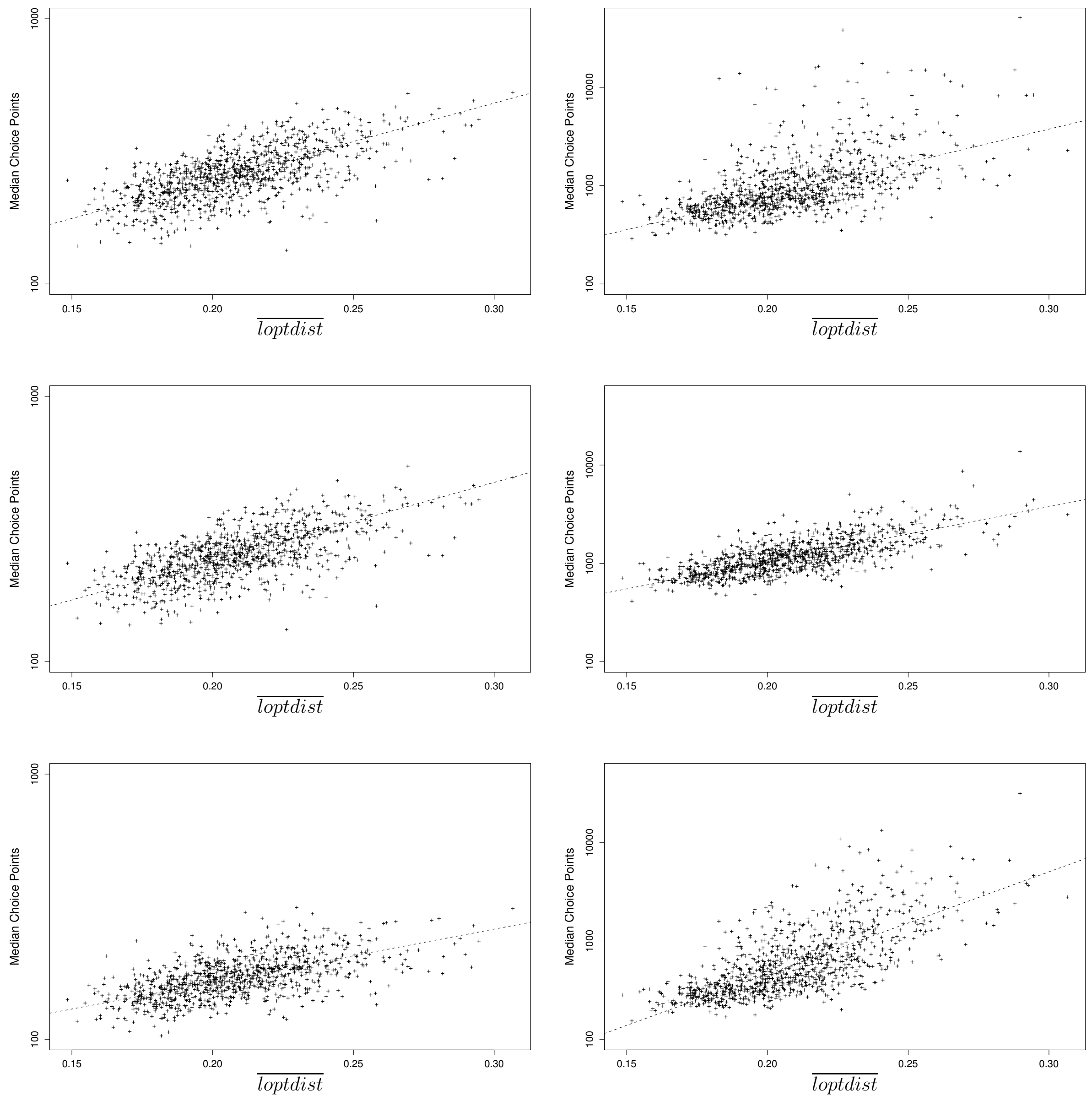

Figure 6.3: Scatter plots and least-squares best-fit lines of $\overline{loptdist}$ versus $cost_{med}$ for the original **Left** and weakened **Right** versions of SGMPCS **Top**, Restart **Middle**, and Chron **Bottom** (log-scale y-axis).

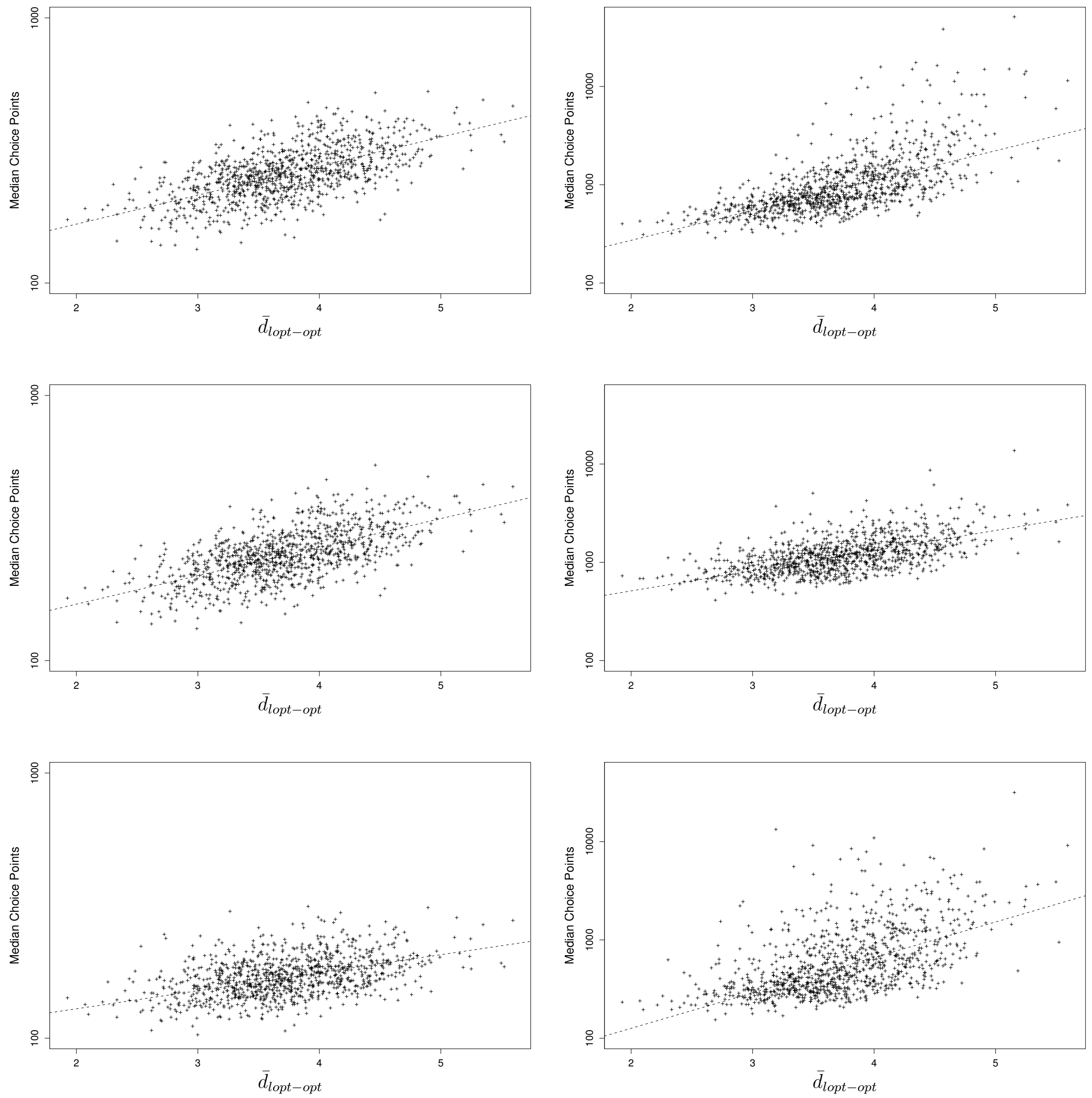

Figure 6.4: Scatter plots and least-squares best-fit lines of $\sqrt{\bar{d}_{lopt-opt}}$ versus $cost_{med}$ for the original **Left** and weakened **Right** versions of SGMPCS **Top**, Restart **Middle**, and Chron **Bottom** (log-scale y-axis).

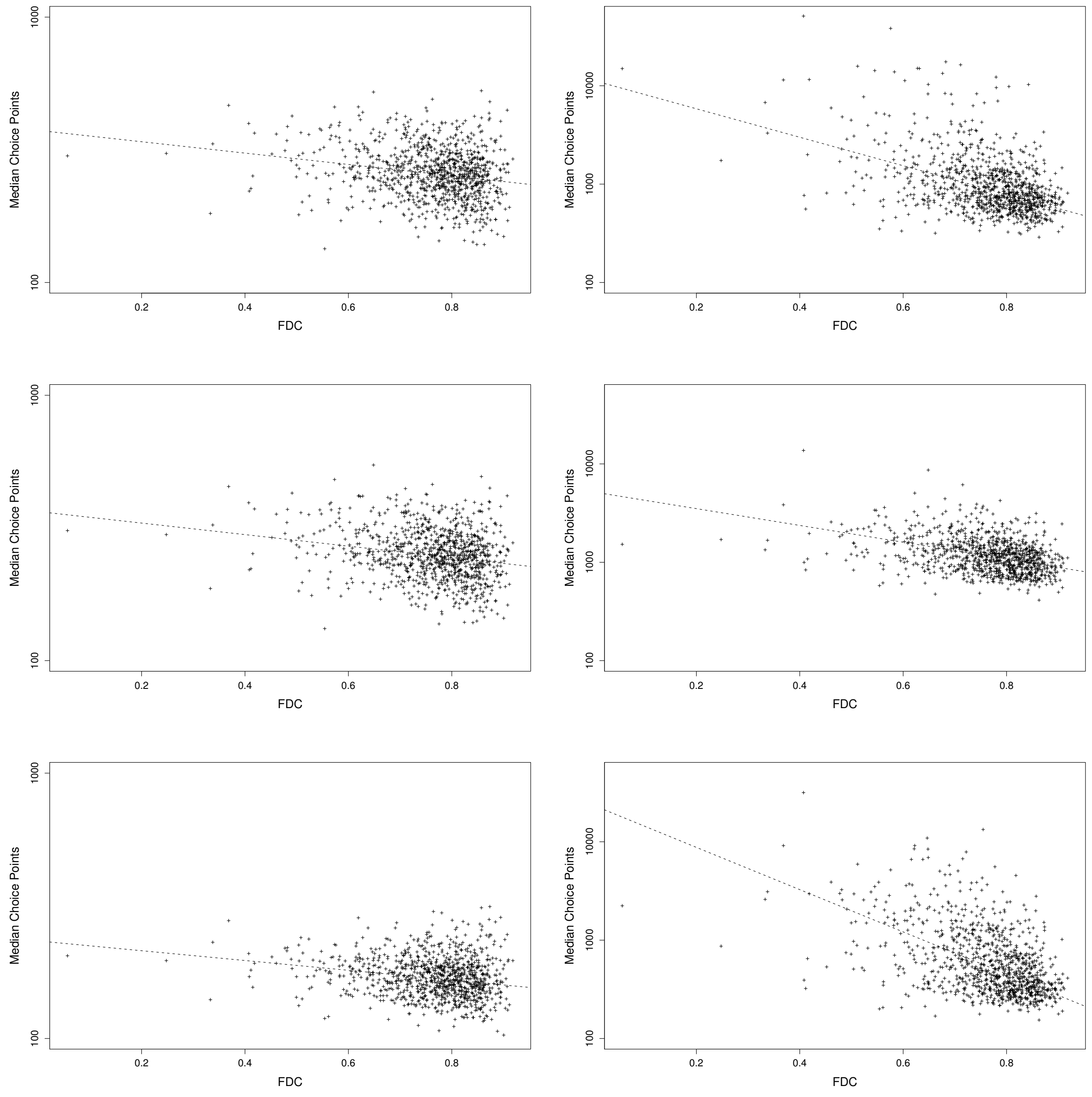

Figure 6.5: Scatter plots and least-squares best-fit lines of $FDC$ versus $cost_{med}$ for the original **Left** and weakened **Right** versions of SGMPCS **Top**, Restart **Middle**, and Chron **Bottom** (log-scale y-axis).

# Chapter 7

# Conclusions and Future Work

In this chapter, we recount the major contributions from this dissertation and suggest future work.

## 7.1 Contributions

The contributions made by this dissertation were as the result of the various empirical investigations of Solution Guided Multi-Point Constructive Search (SGMPCS). The contributions are the application of SGMPCS to constraint satisfaction problems, and the investigation of heavy-tailed distributions, evaluations functions and static cost models of SGMPCS.

### 7.1.1 SGMPCS on CSPs

This dissertation provided the first systematic application of Solution Guided Multi-Point Constructive Search to constraint satisfaction problems. Empirical results demonstrated that SGMPCS 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 led to such strong results. In general, our results are in agreement with previous studies on optimization problems in the scheduling domain, however the results reinforce the intuition that exploiting multiple viewpoints can be of substantial benefit in heuristic search.

An interesting pattern was revealed in the three constraint satisfaction problems experimented with. SGMPCS significantly out-performed chronological backtracking on the quasigroup-with-holes (QWH) and magic square problems but significantly under-performed on the multi-dimensional knapsack problems. Similarly, SGMPCS out-performed randomized restart on the quasigroup problems, performed slightly worse on the magic square problems, and performed about the same on the multi-dimensional knapsack problems. Evaluating the elite solutions by their cost instead of the number of assigned variables as an optimization problem resulted in orders of magnitude speed-up in finding the optimal solution on the multi-dimensional knapsack problems.

### 7.1.2 Heavy Tails and SGMPCS

Our thesis is that one factor for the good performance of SGMPCS is the exploitation of heavy tails. We show that SGMPCS should theoretically also benefit in the same way as other rapid randomized restart techniques when heavy-tailed distributions are present. From the experiments of Chapter 4, we confirm that the distribution in run-times of the backtracking search on JSP instances can be heavy-tailed. It was shown that SGMPCS and randomized restart techniques perform better at the same instance sizes in which heavy tails start to appear. Guided runs used by SGMPCS were also shown to exhibit heavy tails. To our knowledge, this is also the first work to measure heavy-tailed distributions in job-shop scheduling optimization.

### 7.1.3 Evaluation Functions in SGMPCS

The second claim of our thesis is that SGMPCS performance is partly due to the benefit of searching in the area of good solutions. In Chapter 5, we investigated the related conjecture that SGMPCS performance, unlike that of randomized restart and chronological backtracking, is partially affected by the quality of the heuristic that is used to select the guiding partial solutions. Empirical results, both in an artificial context and using three different heuristic evaluation functions, demonstrated that the correlation between the heuristic evaluation of a state and its proximity to the satisfying solution has a strong impact on search performance of SGMPCS. This (partial) descriptive model is important for three main reasons:

1. It makes strong, testable predictions about the behaviour of SGMPCS on other constraint satisfaction and optimization problems.

2. It provides a clear direction for improving SGMPCS search performance: the creation of, perhaps domain-dependent, heuristic evaluation functions for partial search states that are well-correlated with the distance to the nearest solution.

3. It re-introduces a heuristic search guidance concept to the constraint programming literature. Though guidance by heuristic evaluation of search states is common in metaheuristics, general AI search (e.g., $A^*$ and game playing), and best-first search approaches, it does not appear to have been exploited in constructive, CP search. We believe this is an important direction for further investigation.

### 7.1.4 Search Cost Models of SGMPCS

This dissertation has provided some interesting initial results investigating search cost models of SGMPCS. As noted in the last section, Chapter 5 investigated a descriptive model of SGMPCS based on the fitness-distance correlation induced by the heuristic evaluation. Approximately 44% of the variation in search performance can be accounted for by the quality of the heuristic.

By evaluating the static cost models from [70] on three constructive search algorithms in Chapter 6, we were able to find correlations between SGMPCS behaviour and various search space features. Surprisingly, models of similar, or greater, strength were found for algorithms that never guided search (randomized restart), or restarted (standard chronological search).

While the accuracy of the various static cost models vary from those found by Watson et al. [70] for tabu search, they remain quite similar. Stronger correlations were found for models based on average distance between local optima $\overline{loptdist}$, where the strongest models of any of our experiments were found for randomized restart (Random) and chronological search (Chron). Slightly stronger correlations were also found between FDC and all constructive algorithms with weakened propagation. The best predictor of search cost for tabu search, based on the distance between local optima and the closest optimal solution $\sqrt{\bar{d}_{lopt-opt}}$, did give the best model for SGMPCS, although it was weaker than that of tabu search, and $\overline{loptdist}$ with Chron and Restart. Overall, there is no clear explanation for the results we saw. From the correlations measured, local-search based search space features do seem to be related to the performance of all three constructive search algorithms investigated. We believe this is the first time such an analysis has been applied to such constructive search algorithms.

## 7.2 Future Work

### 7.2.1 Applying SGMPCS to Other Problems

One direction for future work is to apply SGMPCS to a larger variety of satisfaction and optimization problems. SGMPCS behaved differently on each of the three satisfaction problems investigated in Chapter 3. The other chapters focused on a single problem: multidimensional knapsack satisfaction or job-shop scheduling optimization.

As the results of Chapter 3 show, for constraint satisfaction problems, SGMPCS only outperformed the other constructive search algorithms on the quasigroup-with-holes (QWH) problem. An important goal of future research is to find out whether SGMPCS can perform similarly on other satisfaction problems, or if QWH is a special case.

Heavy-tailed distributions were only measured for instances of job shop optimization. Similar experiments can be applied to measure the distribution of run-times for other optimization and satisfaction problems. We would again expect SGMPCS to perform better with problem instances that exhibit heavy-tailed distributions.

The fitness-distance correlation (FDC) analysis of SGMPCS on multiknapsack satisfaction problems can be naturally extended to a wider range of satisfaction problems. From other problems, we can test our hypothesis for SGMPCS that performance is affected by how well the evaluation of partial/suboptimal solutions predicts distance to the closest satisfying/optimal solution.

### 7.2.2 Applying Analysis to Larger Problems

One common constraint in many of the experiments in this dissertation was the computational cost of using larger instances. As shown in Chapter 4, SGMPCS only outperforms other techniques at larger problem sizes. To understand how SGMPCS works, we should be investigating it in conditions when it does actually work.[1]

[1]The rational for using these smaller instances was that even though SGMPCS was not outperforming other techniques, there was still high variability between instances. We could still investigate relative performance: why

The 6x6 job-shop instances in Chapter 6 were taken from Watson et al.'s 2003 paper [70]. Speed of computers have increased since then, so moderately larger JSP instances may be considered. We may also rely on statistical techniques to estimate search space feature measurements. Achlioptas et al. use such a technique to estimate backbone sizes of the quasigroup-with-holes problem [1]. Perhaps the distance in fitness-distance correlation measures could also be estimated by redefining distance as the average, or minimum, distance to $k$ random, uniformly distributed, solutions.

### 7.2.3 Extending Cost Models of SGMPCS

Chapter 6 showed some interesting initial results investigating cost models of SGMPCS, from which further work can be done in order to understand the results and improve on the models. Still left mostly unexplained are the similar correlations found for the algorithms other than SGMPCS. One way of investigating whether our results were due to the similar updating of the bounds on the optimization criterion is to look at problems where these bounds are less important to search performance, such as when the objective of minimization is the weighted tardiness of the individual jobs [6]. On these problems, we would expect the correlations to be weaker for randomized restart and chronological search relative to those for SGMPCS.

The search space features used in Chapter 6 were taken directly from Watson et al.'s [70] study on tabu search. A number of these features are based on the local minima defined by the move operator used by tabu search. A first step in improving the cost models for SGMPCS would be to redefine local minima, and possibly distance as well, as they pertain to SGMPCS. As suggested in Chapter 6, dynamic models of SGMPCS based on solutions that enter the elite set may also be more accurate predictors of search cost.

## 7.3 Conclusion

The central thesis of this dissertation is that, as proposed by Beck [7], SGMPCS performance is partially influenced by at least two non-mutually exclusive factors: the exploitation of heavy tails and guiding search in the area of past good solutions. In regards to the first factor, we have shown theoretically and empirically that SGMPCS should benefit when heavy tails are present. In terms of support for the second factor, guiding in the area of good solutions, our results are more mixed. In Chapter 5, we were able to show strong empirical results, both in an artificial context and using three different heuristic evaluation functions, demonstrating that the FDC induced by the evaluation function has a strong impact on SGMPCS performance. The results of Chapter 6, which show that static cost models for local search perform similarly in predicting all constructive search techniques, while interesting, remain mostly unexplained and provide little support to our thesis. Overall, through our empirical investigations, we have gained a considerable amount of information on how and why SGMPCS works, as well as created opportunities for further research on SGMPCS and heuristic search in general.

SGMPCS performed better on one instance than another.

# Bibliography

[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] R. Backofen and S. Will. A constraint-based approach to structure prediction for simplified protein models that outperforms other existing methods. In *Proceedings of the 19th International Conference on Logic Programming (ICLP 2003)*, pages 49–71, 2003.

[3] Luis Baptista and Joao P. Marques Silva. Using randomization and learning to solve hard real-world instances of satisfiability. In *Principles and Practice of Constraint Programming*, pages 489–494, 2000.

[4] J. C. Beck. Multi-point constructive search. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP05)*, pages 737–741, 2005.

[5] 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.

[6] 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 (ICAPS06)*, pages 274–283, 2006.

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

[8] J. C. Beck, A. J. Davenport, E. D. Davis, and M. S. Fox. The ODO project: Toward a unified basis for constraint-directed scheduling. *Journal of Scheduling*, 1(2):89–125, 1998.

[9] 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.

[10] J. C. Beck and J.-P. Watson. Adaptive search algorithms and fitness-distance correlation. In *Proceedings of the Fifth Metaheuristics International Conference*, 2003.

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

[12] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, volume 1, pages 331–337, 1991.

[13] Hubie Chen, Carla P. Gomes, and Bart Selman. Formal models of heavy-tailed behavior in combinatorial search. In *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 408–421, London, UK, 2001. Springer-Verlag.

[14] D. A. Clark, J. Frank, I. P. Gent, E. MacIntyre, N. Tomov, and T. Walsh. Local search and the number of solutions. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (CP96)*, pages 119–133. Springer, 1996.

[15] Philippe Collard, Alessio Gaspar, Manuel Clergue, and Cathy Escazut. Fitness distance correlation, as statistical measure of genetic algorithm difficulty, revisited. In *European Conference on Artificial Intelligence*, pages 650–654, 1998.

[16] M. Davis and H.. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[17] B. DeBacker, P. Furnon, V. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuritics. *Journal of Heuristics*, 6:501–523, 2000.

[18] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[19] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[20] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[21] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, volume 1, pages 246–252, 1996.

[22] I. P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.

[23] M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[24] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, Mass., 1989.

[25] C. P. Gomes and B. Selman. Search strategies for hybrid search spaces. In *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, page 359, Los Alamitos, CA, USA, 1999. IEEE Computer Society.

[26] 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.

[27] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.

[28] 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.

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

[30] C.P. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.

[31] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–314, 1980.

[32] W. D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Department of Computer Science, Stanford University, 1995.

[33] T. Hogg and C. P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69:359–377, 1994.

[34] J. Holland. *Adaptation in natural systems*. University of Michigan Press, Ann Arbor, Mich., 1975.

[35] J. N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42:201–212, 1994.

[36] H.H. Hoos and T. Stüzle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

[37] J. Huang. The effect of restarts on the efficiency of clause learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI07)*, pages 2318–2323, 2007.

[38] T. Hulubei and B. O'Sullivan. The impact of search heuristics on heavy-tailed behaviour. *Constraints*, 11(2–3):159–178, 2006.

[39] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, pages 32–44, 1992.

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

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

[42] 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.

[43] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180, 1993.

[44] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[45] D.C. Mattfeld, C. Bierwirth, and H. Kopfer. A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, 86:441–453, 199.

[46] P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–345, 2001.

[47] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, 1992.

[48] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[49] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[50] E. Nowicki and C. Smutnicki. New algorithm for the job shop problem. Technical report, Institute of Engineering Cybernetics, Wroclaw University of Technology, Poland, 2003.

[51] 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.

[52] A. J. Parkes. Clustering at the Phase Transition. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 340–345, Providence, RI, 1997.

[53] 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.

[54] 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.

[55] 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.

[56] W. Ruml. Incomplete tree search using adaptive probing. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[57] W. Ruml. *Adaptive tree search*. PhD thesis, Dept. of Computer Science, Harvard University, 2002.

[58] Scheduler. *ILOG Scheduler 6.2 User's Manual and Reference Manual*. ILOG, S.A., 2006.

[59] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP98)*, pages 417–431. Springer-Verlag, 1998.

[60] J. Singer, I.P. Gent, and A. Smaill. Backbone fragility and local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

[61] B. M. Smith and M. E. Dyer. Locating the phase transition in constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.

[62] B. M. Smith and S. A. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1, pages 646–651, 1995.

[63] R. Stallman and G. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196, 1977.

[64] Jones T. and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, 1995. Morgan Kaufmann.

[65] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[66] T. Walsh. The constrainedness knife-edge. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[67] J.-P. Watson. *Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem*. PhD thesis, Dept. of Computer Science, Colorado State University, 2003.

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

[69] 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.

[70] 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.

[71] J.-P. Watson, A. E. Howe, and L. D. Whitley. Deconstructing Nowicki and Smutnicki's *i*-TSAB tabu search algorithm for the job-shop scheduling problem. *Computers and Operations Research*, 33(9):2623–2644, 2006.

[72] C. P. Williams and T. Hogg. Using deep structure to locate hard problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 472–477, 1992.

[73] R. Williams, C. Gomes, and B. Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Proceedings of Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT-03)*, 2003.

[74] M. Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Principles and Practice of Constraint Programming*, pages 356–370, 1997.