Real Time Solving of Discrete Optimization Problems

Yair Nof

Real Time Solving of Discrete Optimization Problems

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Information
Management Engineering

Yair Nof

Submitted to the Senate
of the Technion — Israel Institute of Technology
Adar 5778 Haifa March 2018

This research was carried out under the supervision of Prof. Ofer Strichman, in the Faculty of Industrial Engineering & Management.

Acknowledgements

I would like to thank my adviser, for showing me the right way. Thanks to my parents, who encouraged me to learn. Thanks to my kids, wife and her parents who supported the thesis completion, with their patience.

Contents

List of Figures

List of Tables

Abstract

Many hard real-time systems have a desired requirement which is impossible to fulfill: to solve a computationally hard optimization problem within a short and fixed amount of time. For such a task, the exact, exponential algorithms are out of scope. Polynomial-Time Approximation Schemes guarantee a $1 - \epsilon$ approximation with a polynomial run-time, but this run-time can easily exceed the short and fixed requirement. In this thesis we define the 'fixed-time variant' of a hard optimization problem, based on giving weights to the hard constraints. In practice only *any-time* algorithms are relevant for such tasks.

We define a concrete optimization problem that we call RAFP and prove that its decision variant is NP-complete. We then study the performance of several probabilistic algorithms (most of them are local-search) that we fit to RAFP's fixed-time variant, with very short time bounds. We study the practical impact of automatically tuning the parameters of those algorithms. In addition, we consider the problem of optimizing a *parallel portfolio* of algorithms. Specifically, we study the problem of choosing k algorithms out of n, for a machine with k computing cores, and the related problem of detecting the minimum number of required cores to achieve an optimal portfolio, with respect to a given training set of benchmarks. The thesis includes the results of numerous experiments that compare the various methods.

Abbreviations and Notations

PTAS	:	Polynomial Time Approximation Scheme
CSP	:	Constraint Satisfaction Problem
WCSP	:	Weighted CSP
VCSP	:	Valued CSP
COP	:	Constraint Optimization Problem
SAT	:	Satisfiability
FT	:	Fixed Time
RAFP	:	Resource Allocation with Forbidden Pairs
k-MCSP	:	Maximum k-Colorable Subgraph Problem
RS	:	Random Search
RW	:	Random Walk
SHC	:	Stochastic Hill Climbing
TS	:	Tabu Search
SA	:	Simulated Annealing
CE	:	Cross Entropy
SMT	:	Satisfiability Modulo Theories
QF_LRA	:	Quantifier Free Linear Real Arithmetic

Chapter 1

Introduction

Hard real-time systems frequently have a seemingly impossible requirement: To solve within a short, fixed amount of time T (e.g., $T = 0.5$ second), a computationally hard optimization problem[1]. For some hard optimization problems there are known polynomial approximations, which may seem to meet this challenge. Such algorithms, known by the name Polynomial-Time Approximation Schemes (PTAS) [WS11], can guarantee a $1 - \epsilon$ proximity to the optimal solution[2]. For example, they can guarantee a solution with an objective value which is not lower than 0.5 the value given by the (exponential) exact algorithm. But PTAS are not necessarily relevant for hard real time systems, for two main reasons:

- They only guarantee a polynomial bound on the run-time, whereas the hard real time system requires a fixed bound. Moreover, this bound increases when the precision parameter ϵ becomes smaller.
- The guaranteed upper-bound on the distance from the optimal value, as implied by the precision parameter ϵ, is frequently less important than the average quality of the solution.

[1] In many real time systems there are other issues that we do not address in this thesis: The input might be a stream of problems with inaccuracies or uncertainties, with a cost for a late delivery of a solution

[2] Throughout this work we refer to maximization problems. The definitions for minimization problems are similar: in this case it will be a $1 + \epsilon$ proximity.

We focus on optimization problems that their decision variant is complete in NP (henceforth, NP-optimization problems); By definition, those can be reduced to the NP-complete Constraint Satisfaction Problem (CSP)[3], which gives us a unified starting point in the discussion that follows. CSP has an optimization variant called the Constraint Optimization Problem (COP), in which the goal is to satisfy the constraints while maximizing (or minimizing) some objective function. It is common to distinguish between *hard* and *soft* constraints in COP, where each of the latter is associated with a weight that reflects the 'reward' for satisfying it. Every solution has to satisfy the hard constraints, and an optimal solution has to additionally maximize the reward by satisfying soft constraints. The problem may also have an objective other than the soft constraints, but soft constraints and the objective are reducible to one another, if we assume that the objective function is a linear function. Hence for convenience we can talk about a constraints system with hard and soft constraints only, without an explicit objective function.

We use COP to define a *fixed-time variant* of an NP-optimization problem. Given the fixed time bound T our goal is to find algorithms that their solution is as good as possible at time T. This, by definition, implies that we cannot guarantee that our solution satisfies all the hard constraints of the original optimization problem, and we therefore need to prioritize them by giving them weights. In other words, we need to turn the hard constraints into soft constraints (in doing so, it is reasonable to assign those constraints larger weight than those associated with the original soft constraints). This gives rise to the following definition:

Definition 1.0.1 (The fixed-time variant of an NP-optimization problem)**.** Given a

- NP-hard optimization problem P cast as a COP, and
- weights to the hard constraints, reflecting their importance relative to each other and the original soft constraints (if there are any),

let $soft(P)$ denote a problem identical to P except that all the hard constraints are turned into soft constraints with the given weights. Given a time limit T, the *fixed-time* variant of P, denoted $FT(P)$, is the problem of finding a solution within time T to $soft(P)$.

Note that the time limit is given in absolute terms, which means that the best solution may depend also on the hardware.

[3] CSP generalizes the propositional satisfiability problem (SAT), but is still in NP. It allows variables with any finite discrete domain (rather than SAT's restriction to the Boolean domain), and a rich modeling language.

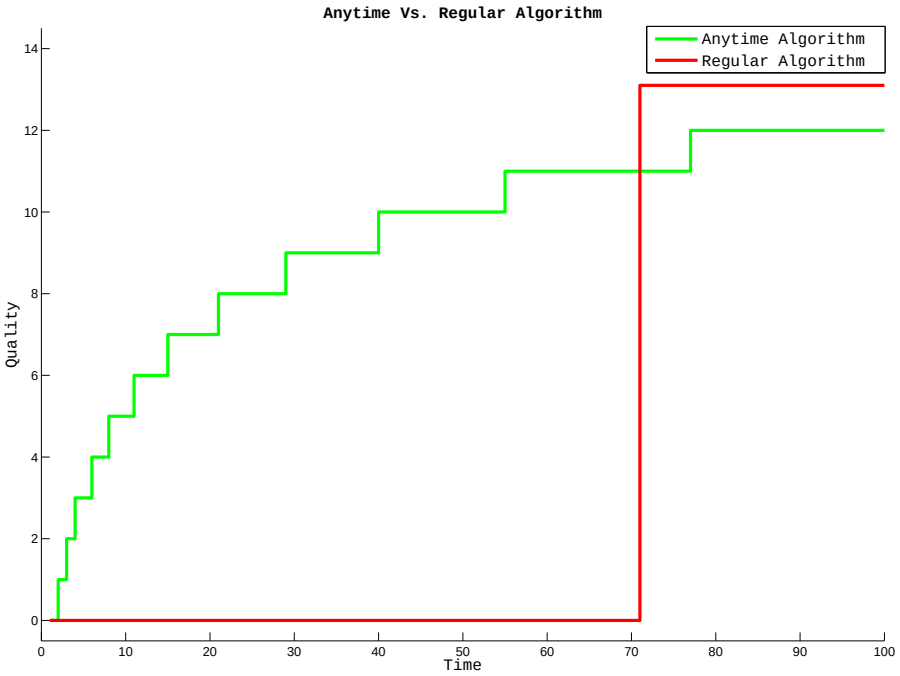


Figure 1.1: Quality vs. time of an anytime and a regular algorithm

Anytime algorithms

To satisfy the requirement of a fixed-time bound, in practice the algorithm has to be an *anytime* algorithm [DB88]. Anytime algorithms maintain intermediate sub-optimal results, and hence if interrupted, they can emit *some* solution. This solution is not likely to be optimal, but is better than nothing. Figure 1.1 illustrates the difference between the quality over time of an anytime algorithm and an algorithm which is not any-time. The diagram demonstrates a case in which the anytime algorithm is *worse* in the long run; this reflects the fact that sacrifices may be necessary for achieving the anytime property. Another family of algorithms are *heuristic* methods. Generally, these methods do not guarantee anything: Their run-time might exceed the fixed bound, and they have no upper-bound on their distance from the optimal value. Some of the heuristic methods, in practice, reach high quality solutions (although not optimal) relatively fast, and hence are good candidates as anytime algorithms.

For our purpose, the value of the algorithm is measured by the quality of the solution that it is able to produce at time T. Most of the work in anytime algorithms that we found, e.g., [BBNP04], [Lou03], [CZ06], [OD12], [SBLI12], [Cla99], [WC00] did not try to measure empirically their success after a fixed and short amount of time. One of the goals of this thesis is to do just that for a particular problem that we will define in the next chapter.

Figure 1.2 summarizes the quality/time trade-off according to the strategies discussed so far.

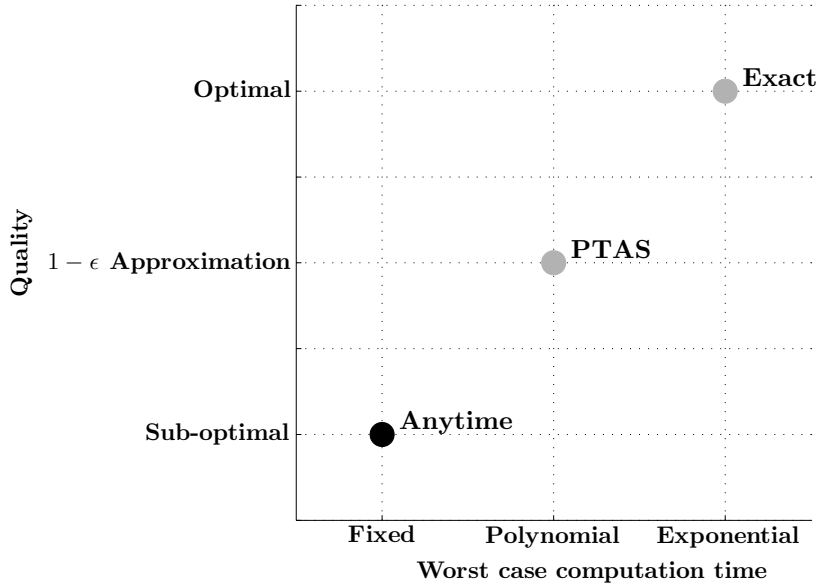


Figure 1.2: Quality vs. Worst computation time of various algorithmic strategies

In this thesis we focus on one particular hard optimization problem which we call RAFP, and give two examples of using it. We define its fixed time variant, and then study the performance of over 7 different algorithms in solving its fixed-time variant with low values of the time bound T. Most of these algorithms are adaptation of known stochastic search algorithms (such as *Tabu Search*, *Stochastic Hill Climbing*, *The Cross-Entropy Method* and *Simulated Annealing*) and their combinations. We then address the problem of optimizing a parallel portfolio of algorithms, for a given number k of available computing cores, and also study the related problem of *convergence*: what is the lowest value of k for which we can get an optimal portfolio?

The structure of the thesis

The thesis continues in the next chapter by defining a hard optimization problem called RAFP, demonstrate its use, and define its fixed-time variant. In chapter 4 we prove that the decision variant of this problem is NP-complete. In chapter 5 we describe a set of anytime algorithms that cope with the fixed time-variant of RAFP. In chapter 6 we describe empirical results based on 500 random inputs for RAFP, some general and some with a focus on hard instances. In chapter 7 we suggest methods to construct optimal parallel portfolios of the algorithms, which will enable us to benefit the most from a computing environment with multiple cores. In chapter 8 we describe empirical results of building a parallel portfolio for our concrete problem. Chapter 9 summarizes our work.

Chapter 2

Preliminaries

In Chapter 5 we will formally define several search algorithms (including pseudo-code) that we use in this work. These meta-heuristics have many variants, and here we only describe the essence, informally, of the common variant. We explain the principles of these algorithms by using the maximization problem in figure 2.1. The figure describes a problem in one dimension, where the x-axis represents the solution space, and the y-axis represents the quality. Although the graph looks continuous, we refer to a discrete set of solutions. For the purpose of these preliminaries (and opposed to Chapter 5), the solution space is assumed to be ordered, which allows us to depict the x-axis. For this reason a 'neighbor' is simply the next or previous value in that order. In the more general case, which is relevant to this thesis, the solutions are not ordered but there are multiple dimensions. When there are multiple dimensions, a 'neighbor' means that only a few dimensions (i.e., variables) change. In the following figures, we will describe a solution as a point on the graph, where the goal is to detect the global maximum. The numbers on a point corresponds to its visit time, starting in point '1'.

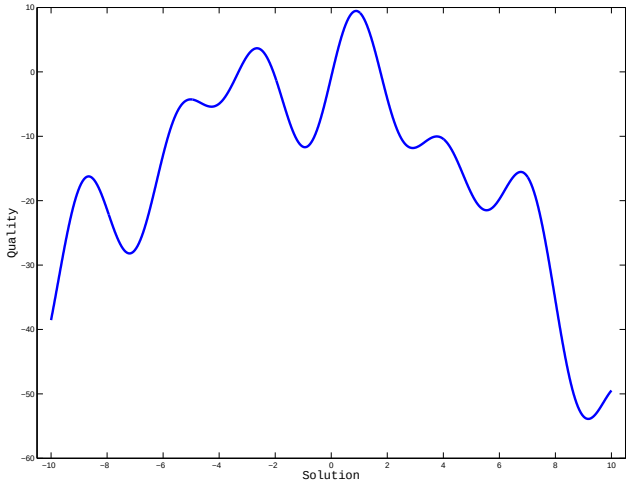


Figure 2.1: Search Illustration - A maximization problem in one dimenstion

First, we show an example of a very simple search algorithm.

Steepest Ascent Hill Climbing

The steepest ascent hill climbing algorithm [Bro11] simply examine all neighboring solutions, and moves to the best neighbor. In our setting, if we define a step of 1 unit, each solution has two neighbors – the right and the left. We move to the solution that has a higher quality. Figure 2.2 describes this process where point '1' is our initial solution, and we are going up-hill until we stop in the top of the hill. At this stage, both our neighbors are worse than the current solution, and the steepest ascent hill climbing algorithm cannot continue. The local maximum that we found is our solution.

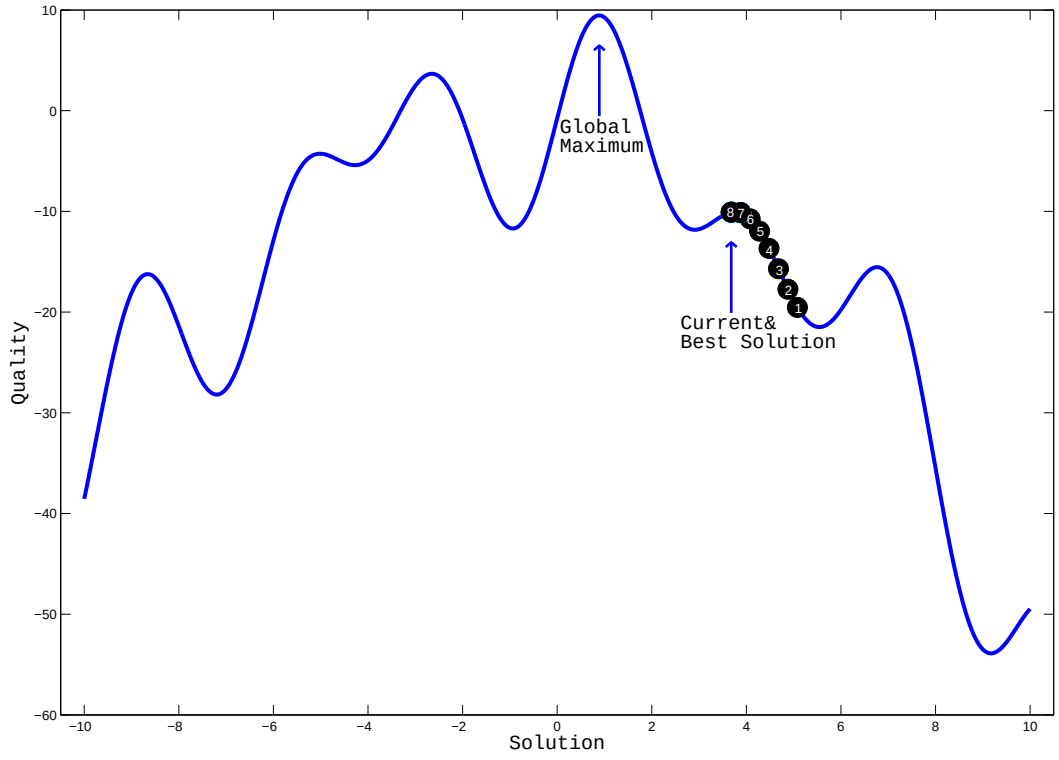


Figure 2.2: Search Illustration - Steepest Ascent Hill Climbing

2.1 Stochastic Search Algorithms

Stochastic algorithms use randomness in the search process, which might help to escape local maxima or accelerate the progress. We show several examples of stochastic search algorithms.

2.1.1 Random Search

Random search [Bro58] is a very simple algorithm. It chooses uniform random solutions iteratively, evaluates their quality and maintains the best solution. When it reaches its timeout, the best solution is returned. Figure 2.3 describes this process, where we can see a random sampling of solutions, the last is marked as *Current solution*, and the *Best solution* is the output.

2.1.2 Random Walk

Random walk [Yan10] iteratively and randomly chooses one of the current solution's neighbors. In our setting, with a step of 1 unit, random walk uniformly chooses between the current solution's right and left neighbors, and makes the chosen neighbor its current solution. Figure 2.4 describes this process, where we can see a more local sampling of solutions than we saw in random search, the current solution is the last solution, and the best solution is the output.

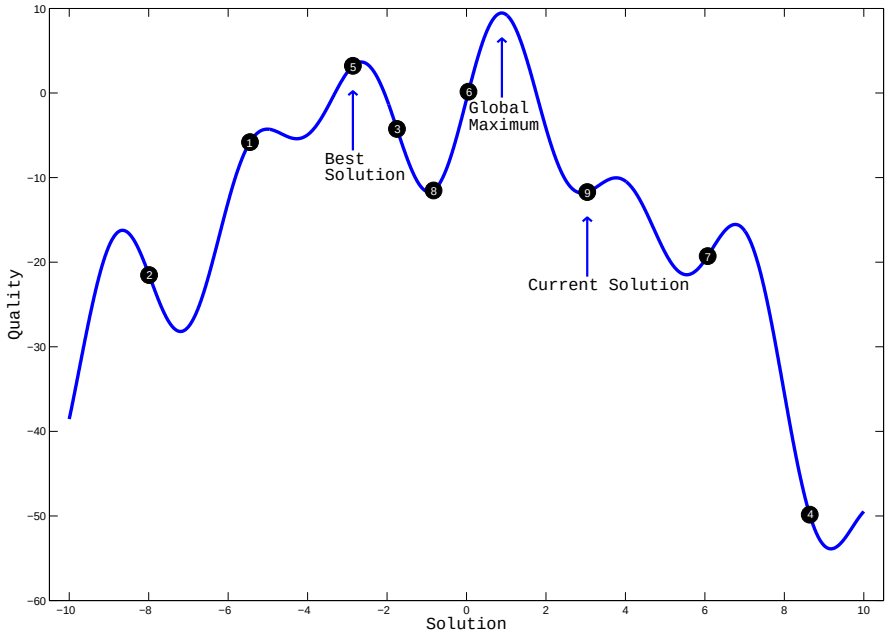


Figure 2.3: Search Illustration - Random Search. We can see a wide and uniform random sampling

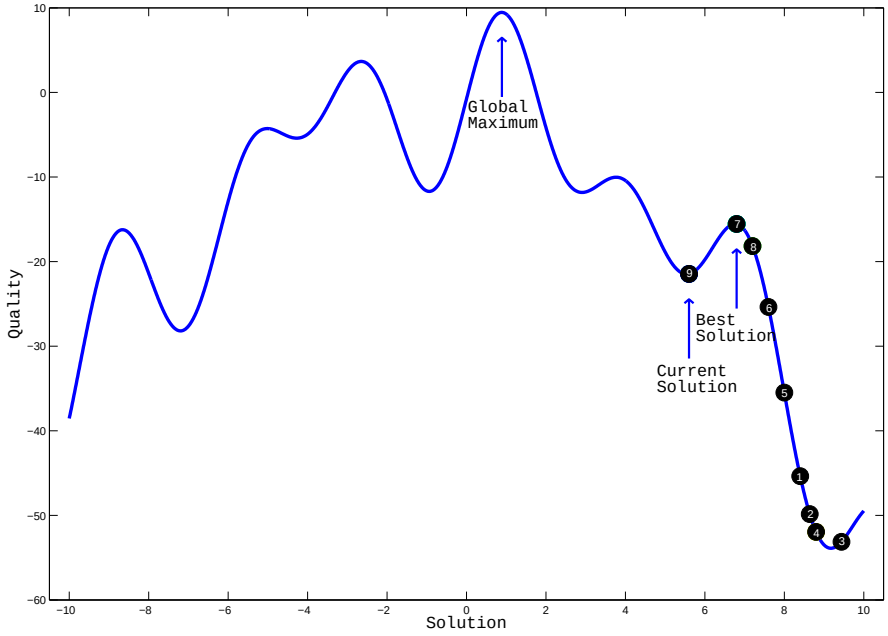


Figure 2.4: Search Illustration - Random Walk. We can see a local and uniform random sampling

2.1.3 Stochastic Hill Climbing

Stochastic hill climbing [FM93] iteratively and randomly samples one of the current solution's neighbors, and accepts it if its quality is higher than the current solution's quality. As opposed to random search and random walk, we do not accept any solution. Figure 2.5 describes this process, where we can see hollow circles in sampled solutions which we did not accept, because their quality was lower than the current solution's quality. We can see the local sampling of solutions. The filled circles show the hill-climbing behavior of this method.

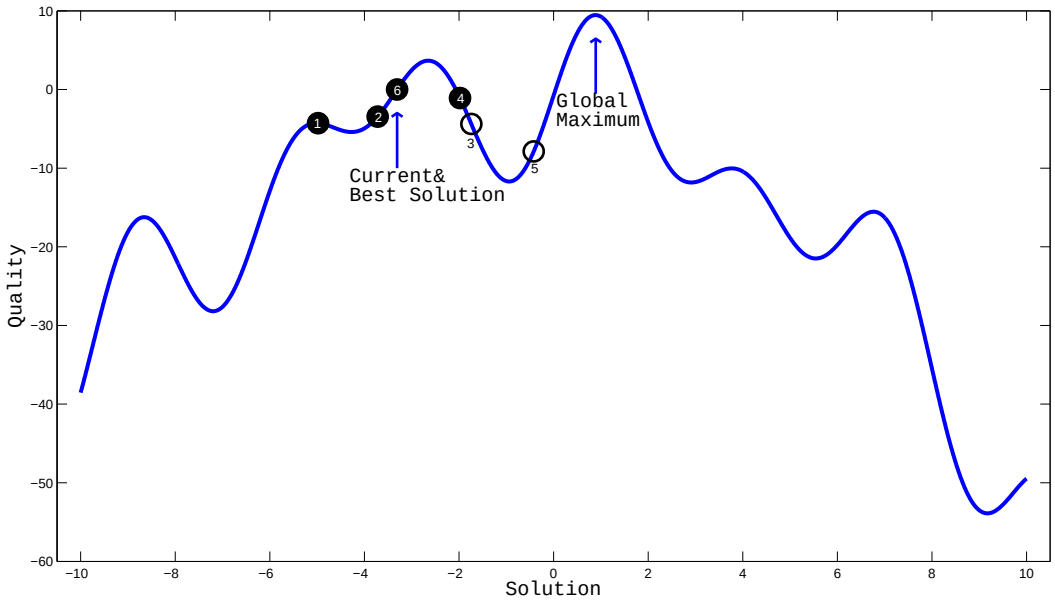


Figure 2.5: Search Illustration - Stochastic Hill Climbing. We can see two solutions (3,5) that were not acceptable since they were worse than the current solution, and acceptable solutions which improve with time

2.1.4 Tabu Search

Tabu search [Glo86] iteratively chooses (deterministically or stochastically) the best neighboring solution, which is not forbidden. A forbidden solution might be a recently visited solution. The forbidden solutions are handled using a constant size *tabu list*. When the tabu list reaches its maximum size, the oldest items are removed. Accepting worse solutions and avoiding last visited solutions help to escape local maxima. Figure 2.6 describes this process, where we can see a tabu list with the last three items, which are temporarily forbidden.

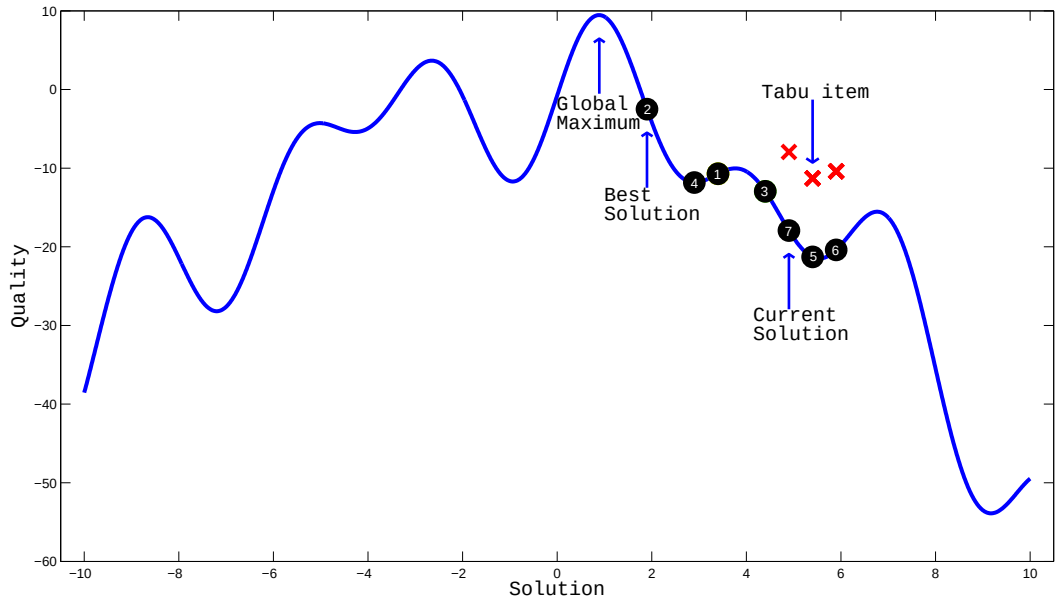


Figure 2.6: Search Illustration - Tabu Search. We can see that the last three solutions (5,6,7) are temporarily forbidden (there is an 'x' above them)

2.1.5 Simulated Annealing

Simulated annealing [JV83] iteratively and randomly samples one of the current solution's neighbors. There is a probability of accepting a neighbor solution. If it is better than the current solution, it is chosen with probability 1. If it is worse than the current solution, the probability of choosing it is lower as the differences in qualities between the current and the candidate solution. A *temperature* parameter also influences the probability of choosing a neighbor solution. At the beginning of the search the temperature is high, and the probability of accepting worse solutions is high too. When time passes, a temperature reduction leads to a lower acceptance probability of worse solutions. This method enables a wide exploration of solutions at the beginning of the search, and a more local exploration of solutions later. Figure 2.7 describes this process, where we can see late candidate solutions that were not chosen because their value is too low to accept in the last period of the search.

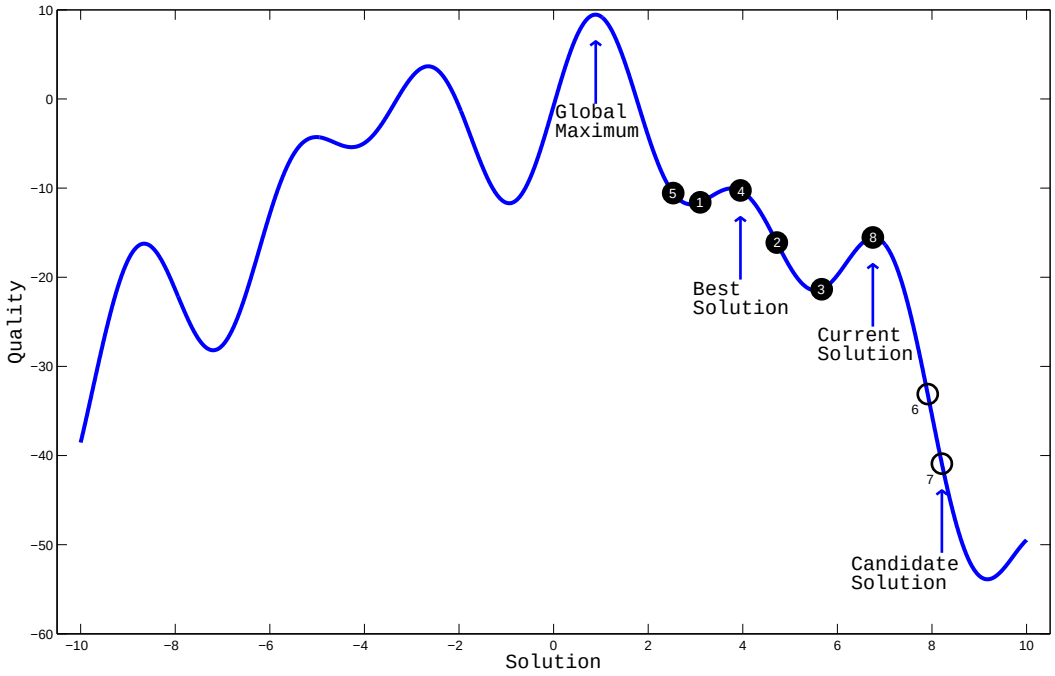


Figure 2.7: Search Illustration - Simulated Annealing. We can see that there are two solutions (6,7) which we do not accept, since their quality is too low at their visit time

In what follows, we do not use the above maximization problem for illustration any more.

2.1.6 The Cross-Entropy Method

The cross-entropy method (CE) [RK04], in its basic form, is parameterized by $N \in \mathbb{N}$, $0 < \rho < 1$, $0 < \alpha < 1$ and $0 < \epsilon < 1$ (typical values could be $\rho = 0.2, \alpha = 0.1, \epsilon = 0.01$). The role of these parameters will be clear momentarily.

CE maintains a probability distribution, initially uniform, for the values in the domain of each variable. It samples N solutions using this distribution, and then evaluates them. The best $\lceil \rho * N \rceil$ samples is called the *elite* set. Based on this set, CE recomputes the distribution, so it becomes more biased towards the elite set. For example, suppose that $\alpha = 0.2$, the probability of x to be assigned the value 12 is currently 0.1, and that $x = 12$ appears in 30% of the solutions in the elite set. Then the new probability of x being assigned 12 will be $0.2 * 0.3 + (1 - 0.2) * 0.1 = 0.14$. After recomputing the distribution in this way, the process is reiterated. Each time a sample is made, if its evaluation is the highest so far from all samples, it is saved. In the end of the process, the best observed sample is returned.

This process can *converge*, which means that for each variable, there is one value with probability larger than $1 - \epsilon$. With a small value of ϵ, this indicates that there is no point reiterating since the results will stay the same with high probability. If convergence occurs, CE can either be terminated or restarted with a different seed, provided that there is enough time.

Figure 2.8 describes this process, where the x-axis represent the values of one variable, the y-axis represent the iteration of the algorithm, and the z-axis represent the probability for each value. We can see a convergence of the algorithm to one of its values.

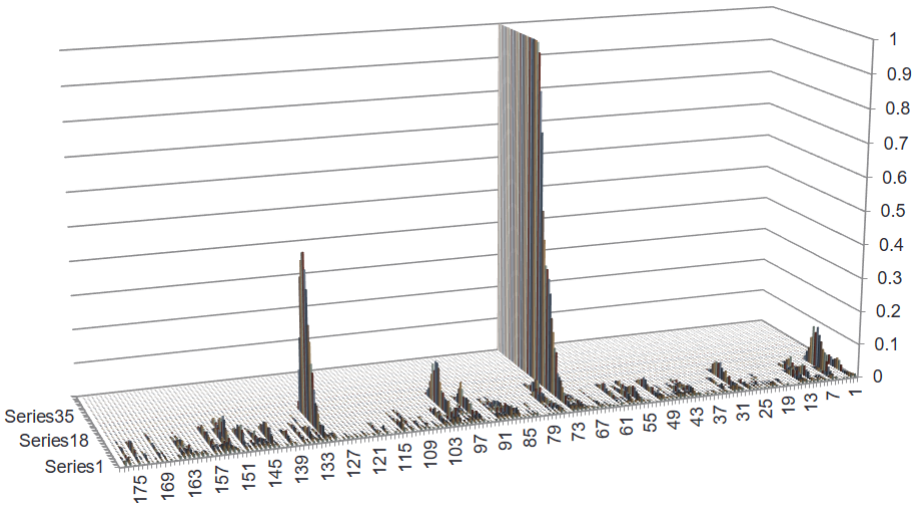


Figure 2.8: Search Illustration - The Cross-Entropy Method

Chapter 3

Problem Formulation

We begin by formulating a hard optimization problem. The rest of the thesis will focus on solving its fixed-time variant.

3.1 The Resource Allocation with Forbidden Pairs (RAFP) Problem

RAFP is defined by

- $A = \{A_1, ..., A_n\}$ is a set of variables.
- D is a set of tactics indices (henceforth 'tactics').
- Res is a set of available resources $\{r_1, r_2, ..., r_m\}$.
- $D_i \subseteq D$ is the domain of tactics available for A_i, for $i \in [1..n]$.
- $R : D \mapsto Res$ maps a tactic d to a resource in Res.
- $U : D \mapsto [0..1]$ is a unary utility function.
- $U : D \times D \mapsto [0..1]$ is a binary utility function.
- $F : D \times D \mapsto \{0, 1\}$ is a matrix of Boolean values, indicating the forbidden pairs of tactics.
- $B_1, B_2, ..., B_m \in \mathbb{N}$ are bounds on the resources of Res.

A solution to RAFP is an assignment $a_1, a_2, \ldots, a_n$ to the variables in A, where $a_i \in D_i$ for $i \in [1..n]$. The hard constraints on the solution are implied by the forbidden-pairs matrix F, and the bound on the resources $B_1, ..., B_m$:

- For each pair of variables A_i, A_j, their selected tactics $a_i \in D_i$ and $a_j \in D_j$ satisfy $F(a_i, a_j) = 0$.

- For each $r_k \in Res$, $|\{R(a_i)\}_{R(a_i)=r_k}| \leq B_k$.

The objective is to maximize the utility:

$$\max \left(\sum_{i=1}^{n} U(a_i) + \sum_{i,j=1,i\neq j}^{n} U(a_i, a_j) \right) . \tag{3.1}$$

We note that RAFP can be defined using an existing framework for hard and soft constraints called *Weighted Constraint Satisfaction Problem*. Appendix A shows this formulation.

3.2 Examples

The RAFP is a rather general discrete optimization problem, which includes binary constraints, some form of cardinality constraints and a utility function with unary and binary factors. Several other general, but more specific known problems, can be derived from RAFP, including the Binary Constraint Satisfaction Problems. Here we present some real world problems which are natural to represent using RAFP.

3.2.1 Computer Aided Dispatch for Medical Services

An emergency control center might use a recommendation system that should react very fast. When medical resources are limited, deciding to send a vehicle for a specific task might cause a delay in other task and risk lives or cause more pain to someone. The following details suggest a RAFP representation for this kind of a system:

- $A = \{A_1, ..., A_n\}$ are emergency medical tasks received at the control center; e.g., An injury of a worker in a construction site, a car accident in a specific junction, etc.
- D are the set of emergency medical solutions; e.g., medication, first-aid, evacuation, CPR, surgery, etc.
- $D_i \subseteq D$ is the set of solutions that fits each emergency medical task; e.g., For a specific car accident first-aid or medication is enough. For some injuries an evacuation over ambulance van or a CPR using an ambulance motorcycle are the only options.
- $U(d)$ for $d \in D$ is the unary utility of using a single solution to an emergency medical task: e.g., the utility of using a well-equipped ambulance van for CPR might be higher than using an ambulance motorcycle for the same task.
- $U(d_i, d_j)$ for $d_i, d_j \in D$ is the binary utility of using two solutions simultaneously: Performing a surgery at the stationary intensive care unit for task d_i delays another task d_j which we should carry out at the same unit, thus reducing $U(d_i, d_j)$.

- $F(d_i, d_j)$ for $d_i, d_j \in D$ is the binary matrix which indicates the pairs of tasks that are impossible to accomplish simultaneously; e.g. we cannot send away a helicopter with medical experts while handling a mass-casualty incident. In these cases we have $F(d_i, d_j) = 1$, which prevents d_i, d_j from being performed together.
- Res is the set of the mobile and stationary resources types; e.g. Ambulance van, ambulance motorcycle, helicopter, intensive care unit, etc.
- $R(d)$ for $d \in D$ maps a single medical solution to its medical resource.
- $B_1, B_2, ..., B_m \in \mathbb{N}$ are bounds on the medical resources.

3.2.2 Automated Trading System

An automated trading system performs buying and selling orders in the stock market. The short system response time and quality of solution is critical for positive and high revenue. The more profitable orders might be carried out only by few machines, special times, have high correlation to other orders, or are forbidden to perform together. The following details suggest a RAFP representation for this kind of a system:

- $A = \{A_1, ..., A_n\}$ is a batch of buying/selling orders to decide in the next trading period.
- $D_i \subseteq D$ is a financial product (stock, bond, option etc.), a buy / sell order and a discrete amount.
- $U(d)$ for $d \in D$ is the expected revenue for order d. Some orders are more profitable than others.
- $U(d_i, d_j)$ for $d_i, d_j \in D$ is the inverse of correlation between orders d_i, d_j. Low correlations are preferred.
- $F(d_i, d_j)$ for $d_i, d_j \in D$ is a binary matrix which indicates which pairs of orders are forbidden together in the same trading period. If d_i, d_j is a forbidden pair we have $F(d_i, d_j) = 1$.
- Res is a set of types of trading machines: Different sizes of servers and clusters that are available to perform our orders.
- $R(d)$ for $d \in D$ maps a single buying/selling order to a trading machine type.
- $B_1, B_2, ..., B_m \in \mathbb{N}$ are bounds on the number of available trading machines of each type, for the next trading period.

3.3 The Fixed Time Variant of RAFP

$soft(RAFP)$ is defined as RAFP with a weight of 0 on the hard constraints. Given a time limit T, the *fixed-time* variant of $RAFP$, denoted $FT(RAFP)$, is the problem of finding a solution within time T to $soft(RAFP)$.

Chapter 4

RAFP's Complexity

We prove the NP-completeness of RAFP by a reduction from the following problem.

4.1 The Maximum k-Colorable Subgraph Problem

The Maximum k-Colorable Subgraph Problem (k-MCSP) is defined as follows.

- $G = (V, E)$ is a graph
- k is a positive integer, which defines a set of colors $C = \{c_1, \ldots, c_k\}$

A solution to k-MCSP is the largest vertex set $V' \subseteq V$ which induces a k-colorable graph, i.e. there is an assignment of colors to the vertices such that no two adjacent vertices are assigned the same color.
For k=1, k-MCSP becomes the Maximum Independent Set problem, which is known to be NP-hard [GJ79]. Therefore k-MCSP is NP-hard.

4.2 The Decision Variants of RAFP and k-MCSP

First we define the decision variants of RAFP and k-MCSP.
We add to k-MCSP an integer $h \in [1..n]$, where $|V| = n$ and ask whether there is a solution to k-MCSP with h or more vertices.
We add to RAFP a value $u \in \mathbb{R}$ and ask whether there is a solution to RAFP with $\sum_{i=1}^{n} U(a_i) + \sum_{i,j=1,i\neq j}^{n} U(a_i, a_j) \geq u$.

4.3 The Decision Variant of RAFP is NP-Complete

Theorem 4.1. *The decision variant of RAFP is NP-Hard*

Proof. We will show that k-MCSP $\leq_P$ *RAFP*.
Given an instance of k-MCSP, we will construct an instance of RAFP.
Given $G = (V, E), k, h$, we define RAFP's elements:

- $A = V$, i.e., the set of variables is the set of vertices.
- $D = \{d_{vc}\}_{v \in V, c \in C'}$, where $C' = C \cup \{c_0\}$. Tactics are all possible colorings of vertices, with the possibility of a vertex without a color, marked by c_0.
- $D_i = \{d_{ic}\}_{c \in C'}$, variable i's tactics are all possible colorings of the i-th vertex, with the possibility that it has no color. Note that $D = \bigcup_i D_i$.
- For each $v \in V, c \in C'$, the unary utility is defined as:

$$U(d_{ic_i}) = \begin{cases} 0 & \text{if } c_i = c_0 \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

- For each $i, j \in V$, and $p, q \in C'$ we have $d_{ip}, d_{jq} \in D$ and the binary utility is defined as:

$$U(d_{ip}, d_{jq}) = \begin{cases} -\infty & \text{if } (i,j) \in E, p = q, p \neq c_0, q \neq c_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

- $F = \{1\}^{|D| \times |D|}$ – There are no forbidden pairs of tactics
- $Res = \{r_1\}, R(d) = r_1$ for $d \in D$, and $B_1 = n$. There is only one resource type with n items.
- We define $u = h$, the utility value of RAFP should be equal to the size of the subgraph in k-MCSP.

Since $|D| = |V| \cdot |C'| = n(k+1), |D_i| = k+1$, the unary domain size is $|D|$, the binary domain size for the binary utility function and the forbidden pairs matrix is $|D|^2$ and $|Res| = 1$, the construction is done in a polynomial time in the size of the k-MCSP instance.

Now we show that the answer for the original instance of k-MCSP is yes if and only if the answer for the RAFP instance we created is yes.

($\Leftarrow$) Suppose k-MCSP has a k-colorable subgraph $G' = (V', E')$ with $|V'| \geq h$, for $h \in \mathbb{N}$, $h \leq |V|$. Let $c_i \in C'$ denote the color of $i \in V$ on G. For each $i \in V'$ we have $U(d_{ic_i}) = 1$ and for each $i \notin V'$ we have $c_i = c_0$ and $U(d_{ic_0}) = 0$. Since we have a coloring with at least h vertices we have:

$$\sum_{i \in V} U(d_{ic_i}) = \sum_{i \in V'} U(d_{ic_i}) + \sum_{i \notin V'} U(d_{ic_i}) \geq h + 0 \geq h. \tag{4.3}$$

Regarding the binary utility we have:

$$\sum_{i,j=1,i\neq j}^{n} U(d_{ic_i}, d_{jc_j}) = \sum_{\substack{(i,j)\in E, c_i=c_j \\ c_i,c_j\neq c_0}} U(d_{ic_i}, d_{jc_j}) + \sum_{\substack{(i,j)\notin E \vee c_i\neq c_j \vee \\ c_i=c_0 \vee c_j=c_0}} U(d_{ic_i}, d_{jc_j}) \tag{4.4}$$

Since we have a coloring, the following sum is over an empty set, thus it is equal to zero:

$$\sum_{\substack{(i,j)\in E, c_i=c_j \\ c_i,c_j\neq c_0}} U(d_{ic_i}, d_{jc_j}) = 0 \tag{4.5}$$

According to (4.2) we have:

$$\sum_{\substack{(i,j)\notin E \vee c_i\neq c_j \vee \\ c_i=c_0 \vee c_j=c_0}} U(d_{ic_i}, d_{jc_j}) = 0 \tag{4.6}$$

Thus, (4.4) becomes

$$\sum_{i,j=1,i\neq j}^{n} U(d_{ic_i}, d_{jc_j}) = 0 \tag{4.7}$$

From (4.3) and (4.7) we have

$$\sum_{i\in V} U(d_{ic_i}) + \sum_{i,j=1,i\neq j}^{n} U(d_{ic_i}, d_{jc_j}) \geq h + 0 \geq h \tag{4.8}$$

as needed for the utility.

Regarding the resource constraints we have for each $i \in V$ and $c_i \in C$: $|\{R(d_{ic_i})\}_{R(d_{ic_i})=r_1}| = n \leq B_1$. The binary hard constraints are trivially met because $F = \{1\}^{|D|\times|D|}$. This concludes this side of the proof.

($\Rightarrow$) Now suppose that the assignment $\{d_{ic_i}\}_{i\in[1..n]}$ is a solution to the constructed RAFP instance, with

$$\sum_{i=1}^{n} U(d_{ic_i}) + \sum_{i,j=1,i\neq j}^{n} U(d_{ic_i}, d_{jc_j}) \geq h \tag{4.9}$$

for $h \in [1..n]$. From (4.9) we infer $U(d_{ic_i}, d_{jc_j}) \neq -\infty$, which implies $U(d_{ic_i}, d_{jc_j}) = 0$, according to (4.2), for each $i, j \in V$ and $c_i, c_j \in C'$. Thus,

$$\sum_{i,j=1,i\neq j}^{n} U(d_{ic_i}, d_{jc_j}) = 0 \tag{4.10}$$

From (4.9) and (4.10) we have

$$\sum_{i=1}^{n} U(d_{ic_i}) \geq h \tag{4.11}$$

According to (4.1), this is equivalent to a k-coloring in V with at least h vertices, different than c_0. The coloring is legal since $U(d_{ic_i}, d_{jc_j}) \neq -\infty$ and according to (4.2).□

Theorem 4.2. *The decision version of RAFP is in NP*

Proof. Let $a_1, a_2, \ldots, a_n$ be a solution to RAFP, with a utility larger or equal to u. To validate the binary constraints, $U(a_i, a_j) \neq -\infty, i \neq j$, we have $O(n^2)$ operations. To validate the resource constraints, $|\{R(a_i)\}_{R(a_i)=r_k}| \leq B_k$, for each $r_k \in Res$, we have n operations. To validate $\sum_{i=1}^{n} U(a_i) + \sum_{i,j=1,i\neq j}^{n} U(a_i, a_j) \geq u$ we have $O(n^2)$ operations. Totally, we have $O(n^2)$ operations to validate that $a_1, a_2, \ldots, a_n$ is a solution to RAFP, a polynomial number of operations, which implies $RAFP \in NP$. □

Theorem 4.3. *The decision version of RAFP is NP-Complete*

Proof. From theorems 4.1 and 4.2 we get that the decision version of RAFP is in NPC.□

Chapter 5

Algorithms for RAFP

5.1 Local Search

Local search is a heuristic framework for solving hard decision and optimization problems. Local search is relatively simple to implement, output a stream of solutions without a setup time, and contains a rich toolbox of parametrized algorithms. Local search meta-heuristics might be easily adapted to many problems and solutions' structures [HS04]. Local search moves from one solution to a neighbor solution in the space of candidate solutions using an evaluation function, and stops if an optimal solution is found or the time bound is reached. The *Random Walk, Stochastic Hill-Climbing, Tabu Search and Simulated Annealing* that we present later in this chapter are local search algorithms.

5.2 Beyond Local Search

When the candidate solutions are sampled using a guidance which is different from a neighboring relation, the search is not local. The *Random Search*, *Greedy* and the *Cross Entropy Method* that we present later in this chapter are search algorithms which are not considered as local search.

5.3 A Unifying Approach For Algorithms

Algorithm 5.1 describes the meta-heuristic that we used, which we call *Fixed-Time Search*. Each derived algorithm implements parts of *Fixed-Time Search*, and adapted to RAFP, when necessary. By using a single framework for all variations of the algorithm, we achieve a relatively fair comparison between them. In all the following references to RAFP, we refer to RT(RAFP).

5.3.1 Fixed-Time Search

Algorithm 5.1 *FixedTimeSearch*(Initial Solution *Init*, Timeout T)

```
Current ← Init
Best ← Current
while not (StopCriterionReached() or timeout T reached) do
  Candidate ← ChooseCandidate(Current)
  if AcceptanceCriterionReached(Candidate) then
    Current ← Candidate
    if EvaluateSolution(Best) ≤ EvaluateSolution(Current) then
      Best ← Current
    end if
  end if
  if RestartCriterionReached() then
    Current ← Restart()
  end if
end while
return Best
```

Next, we describe common definitions and procedures of all algorithms, in order to fit *Fixed-Time Search* to RAFP.

Definition 5.3.1 (Distance between solutions)**.** Let $S = (a_1, a_2, ..., a_n)$ and $S' = (b_1, b_2, ..., b_n)$ be two solutions to a RAFP instance. The *distance between the solutions* is defined as the number of different elements between them : $D(S, S') = |\{i \mid a_i \neq b_i\}|$.

Definition 5.3.2 (K-Neighborhood of a solution)**.** The *K-Neighborhood* of S, is the set of all solutions that their distance from S is no more than K: $N_K(S) = \{S' | D(S, S') \leq K\}$.

5.3.2 The Procedure *GenerateRandomNeighbor*

The procedure *GenerateRandomNeighbor* is used by most of the stochastic algorithms that follow, in order to generate a single neighbor of the current solution. The neighbor is in the *K-Neighborhood* of the current solution. Procedure 5.2 describes this process.

Procedure 5.2 *GenerateRandomNeighbor*(Solution *Current*, Neighborhood Size K)

$Neighbor \leftarrow Current$
for K times **do**
  $i \leftarrow$ a variable's index chosen uniformly at random
  $v \leftarrow$ a value chosen uniformly at random from the domain of $Neighbor.Var_i$
  $Neighbor.Var_i \leftarrow v$
end for
return $Neighbor$

5.3.3 The Procedure *EvaluateSolution*

EvaluateSolution is a procedure tailored for the RAFP problem. It computes the objective value described in Chapter 3, given a full assignment.

5.4 Instances of Fixed-Time Search

In what follows, we describe how to derive specific search algorithms from *Fixed-Time Search*, using the procedures *GenerateRandomNeighbor* and *EvaluateSolution*.

5.4.1 Random Search

Random Search or Pure Random Search [Bro58] is the simplest algorithm of Fixed-Time Search: Generate a series of random solutions of N variables and choose the best one according to the objective function. Procedures 5.3-5.7 describe the random search algorithm for RAFP. Random search is not likely to be competitive, and we only include it for reference.

Procedure 5.3 $StopCriterionReached_{RS}()$

return false

Procedure 5.4 $ChooseCandidate_{RS}$(Solution *Current*)

$GenerateRandomNeighbor(Current, N)$

Procedure 5.5 $AcceptanceCriterionReached_{RS}$(Solution *Candidate*)

return true

Procedure 5.6 $RestartCriterionReached_{RS}()$

return false

Procedure 5.7 $Restart_{RS}()$

return empty solution

5.4.2 Random Walk

Random Walk [Yan10] is a simple local search algorithm which moves between random neighbor solutions, according to some neighboring relation. Procedures 5.8-5.12 describe the random walk algorithm for RAFP.

Procedure 5.8 $StopCriterionReached_{RW}()$

return false

Procedure 5.9 $ChooseCandidate_{RW}$(Solution $Current$, Neighborhood Size K)

$GenerateRandomNeighbor(Current, K)$

Procedure 5.10 $AcceptanceCriterionReached_{RW}$(Solution $Candidate$)

return true

Procedure 5.11 $RestartCriterionReached_{RW}()$

return false

Procedure 5.12 $Restart_{RW}()$

return empty solution

5.4.3 Stochastic Hill Climbing

Stochastic Hill Climbing [FM93] is a simple local search algorithm which only adds a simple acceptance criterion to random walk: Move to the neighbor if and only if it is better than the current solution. Procedures 5.13-5.17 describe the stochastic hill climbing algorithm for RAFP.

Procedure 5.13 $StopCriterionReached_{SHC}()$

return false

Procedure 5.14 $ChooseCandidate_{SHC}$(Solution $Current$, Neighborhood Size K)

$GenerateRandomNeighbor(Current, K))$

Procedure 5.15 $AcceptanceCriterionReached_{SHC}$(Solution $Current$, Solution $Candidate$)

return $EvaluateSolution(Current) \leq EvaluateSolution(Candidate)$

Procedure 5.16 $RestartCriterionReached_{SHC}()$

return false

Procedure 5.17 $Restart_{SHC}()$

return empty solution

5.4.4 Tabu Search

Tabu Search [Glo86] is a local search algorithm which uses constant memory in order to avoid visiting the same solutions, and escapes from local maxima. In the tabu search for RAFP, we maintain a tabu list of variables. The list contains variables which changed their value recently. A variable in the tabu list will not change its value unless it yields a better solution than the best solution[1]. Procedures 5.18-5.24 describe the tabu search algorithm for RAFP.

Procedure 5.18 $StopCriterionReached_{TS}()$

return false

Procedure 5.19 $ChooseCandidate_{TS}$(Solution $Current$, Neighborhood Size K)

$GenerateRandomNeighbor(Current, K)$

Procedure 5.20 $TabuForbidSolution$(Solution $Candidate$, Solution $Current$)

```
for each variable index i do
  if Candidate.Var_i ≠ Current.Var_i and TabuList.Contain(i) then
    return true
  end if
end for
return false
```

Procedure 5.21 $TabuUpdate$(Solution $Candidate$, Solution $Current$)

```
for each variable index i do
  if Candidate.Var_i ≠ Current.Var_i then
    TabuList.InsertFront(i)
  end if
  if TabuList.Size() > Maximum allowed tabu list size then
    TabuList.RemoveBack()
  end if
end for
```

[1] Here we deviate from the common implementation of tabu search since a-priori it seemed to be better. In our variant, instead of choosing the **best** neighbor that is not forbidden, we choose **some** neighbor which is not forbidden.

Procedure 5.22 *AcceptanceCriterionReached*$_{TS}$(Solutions *Best*, *Current*, *Candidate*)

if $EvaluateSolution(Best) \leq EvaluateSolution(Candidate)$ **or not** $TabuForbidSolution(Candidate, Current)$ **then**
  $TabuUpdate(Candidate, Current)$
  return true
end if
return false

Procedure 5.23 *RestartCriterionReached*$_{TS}$()

return false

Procedure 5.24 *Restart*$_{TS}$()

return empty solution

5.4.5 Simulated Annealing

Simulated Annealing [JV83] is a local search algorithm which uses the concept of temperature in order to change its behavior during the search. The acceptance criterion of a neighbor solution might approve a solution which has a lower quality than the current solution. The probability of accepting worse solutions is a function of its relative quality and the temperature. There is an initial temperature and a cooling rate, where low temperatures yield low acceptance probabilities of worse solutions. Procedures 5.25-5.29 describe the simulated annealing algorithm for RAFP.

Procedure 5.25 *StopCriterionReached*$_{SA}$()

return false

Procedure 5.26 *ChooseCandidate*$_{SA}$(Solution *Current*, Neighborhood Size K)

$GenerateRandomNeighbor(Current, K)$

Procedure 5.27 *AcceptanceCriterionReached*$_{SA}$(Solution *Current*, Solution *Candidate*)

$T \leftarrow \frac{T}{Rate}$
$\Delta \leftarrow EvaluateSolution(Candidate) - EvaluateSolution(Current)$
if $\Delta \geq 0$ **then**
  $P \leftarrow 1$
else
  $P \leftarrow e^{\Delta/T}$
end if
return true with probability P
return false

Procedure 5.28 $RestartCriterionReached_{SA}()$

return false

Procedure 5.29 $Restart_{SA}()$

return empty solution

5.4.6 The Cross-Entropy Method

The Cross-Entropy Method [RK04] is a stochastic algorithm used for simulation of rare events and for optimization. The Cross-Entropy Method for RAFP maintains a probability distribution function for the domain values of each variable. Initially, the probability distributions are uniform. In each iteration, we sample n solutions from the current distribution, and evaluate them. We take the best $\lceil \rho \times n \rceil$ solutions, for some $0 < \rho < 1$, to shape the distribution for the next iteration, using a smoothing factor and the current distribution. Procedures 5.30-5.34 describe the cross entropy method.

Procedure 5.30 $StopCriterionReached_{CE}()$[2]

return false

Procedure 5.31 $ChooseCandidate_{CE}$(Sample size n, Smoothing factor α, Elite ratio ρ)

$Samples \leftarrow$ an empty list
for n times **do**
  for each variable index i **do**
    $Current.Var_i \leftarrow$ Random value according to pdf_i
  end for
  $Current.Val \leftarrow EvaluateSolution(Current)$
  $Samples.Insert(Current)$
end for
$Elite \leftarrow \lceil \rho \times n \rceil$ best samples in $Samples$
for each variable index i **do**
  $pdf'_i \leftarrow$ distribution according to $Elite$
  $pdf_i \leftarrow \alpha \times pdf'_i + (1 - \alpha) \times pdf_i$
end for
return best sample in $Samples$

Procedure 5.32 $AcceptanceCriterionReached_{CE}$(Solution $Candidate$)

return true

[2] In our current implementation of the cross-entropy method, the *StopCriterionReached*() contains what is under *RestartCriterionReached*() and the *Restart*() procedure returns an empty solution

Procedure 5.33 $RestartCriterionReached_{CE}$(Convergence constant ε)

return $\forall i \max pdf_i \geq 1 - \varepsilon$

Procedure 5.34 $Restart_{CE}()$

Initialize $pdf_i's$ as uniform distributions biased by the initial solution's values

Set a new random seed

5.4.7 A Greedy Algorithm

The greedy method described here is a natural one for RAFP: For some variable ordering, pick the value of each variable as the one which optimizes the objective function of RAFP, while satisfying all constraints derived from previous variables assignments. Procedures 5.35-5.39 describes the greedy algorithm for RAFP.

Procedure 5.35 $StopCriterionReached_{Greedy}()$

return $CurrentVar \geq n$

Procedure 5.36 $ChooseCandidate_{Greedy}$(Solution $Current$, Variable $CurrentVar$)

$Candidate \leftarrow Current$
$BestCandidate \leftarrow$ Worst solution
for Value in $CurrentVar.Values$ **do**
  $Candidate.CurrentVar \leftarrow Value$
  if $BestCandidate.Value < EvaluateSolution(Candidate)$ **then**
    $BestCandidate \leftarrow Candidate$
  end if
end for
$CurrentVar \leftarrow CurrentVar + 1$
return $BestCandidate$

Procedure 5.37 $AcceptanceCriterionReached_{Greedy}$(Solution $Current$, Solution $Candidate$)

return true

Procedure 5.38 $RestartCriterionReached_{Greedy}()$

return false

Procedure 5.39 $Restart_{Greedy}()$

return empty solution

5.5 Leveraging the Greedy Algorithm

All algorithms above, except the greedy algorithm, yield a series of solutions, terminated by the timeout. The greedy algorithm yields one solution and stops. Greedy's run time might be below the timeout, and it might be useful to use the remaining run time for other computations.

5.5.1 Iterated Greedy

First, We can use the greedy algorithm to produce a series of solutions by using several variable orderings, since it may produce a different solution for each variable ordering. Procedures 5.40-5.44 describes the greedy loop algorithm for RAFP.

Procedure 5.40 $StopCriterionReached_{GreedyLoop}()$

return false

Procedure 5.41 $ChooseCandidate_{GreedyLoop}$(Solution $Current$, Variable $CurrentVar$)

$Candidate \leftarrow Current$
$BestCandidate \leftarrow$ Worst solution
for Value in $CurrentVar.Values$ **do**
  $Candidate.CurrentVar \leftarrow Value$
  if $BestCandidate.Value < EvaluateSolution(Candidate)$ **then**
    $BestCandidate \leftarrow Candidate$
  end if
end for
$CurrentVar \leftarrow CurrentVar + 1$
return $BestCandidate$

Procedure 5.42 $AcceptanceCriterionReached_{GreedyLoop}$(Solutions $Current$, $Candidate$)

return $EvaluateSolution(Current) \leq EvaluateSolution(Candidate)$

Procedure 5.43 $RestartCriterionReached_{Greedy}()$

return $CurrentVar \geq n$

Procedure 5.44 $Restart_{Greedy}()$

Produce a random variable ordering
$CurrentVar \leftarrow 1$
return empty solution

5.5.2 Hybrid: Greedy + Search

Another way of using the greedy algorithm is to start with its solution as an initial solution for any of the other algorithms above. For most of the algorithms above it allows a local search in the neighborhood of the greedy solution, hopefully a neighborhood with high-quality solutions.

5.5.3 Hybrid for the Cross-Entropy Method

For *The Cross-Entropy Method*, running the greedy algorithm first returns a solution that we can use in another way: Initially, the probability distributions have the weight $0 \leq w < 1$ for the initial solution's values, and uniform distribution for the rest of the values. This biases the distribution towards the values of the greedy solution, that have high qualities, we believe.

Chapter 6

Empirical Results: Individual Algorithms

6.1 Implementation

We implemented all algorithms from chapter 5 in C++ under Windows 10. We ran the algorithms serially on a Dell Vostro 3360 with Core i7-3517U at 1.9-2.4Ghz, using the same timeout.

6.1.1 Inputs

We generated random inputs, using the following values for RAFP:

- Random number of variables – $n \in [1..200]$
- Constant set of tactics – $|D| = 6000$
- Derived variables domain – $D_i = \left[1 + \frac{|D|}{n} \times (i-1), \ldots, \frac{|D|}{n} \times i\right]$, for $i \in [1..n]$
- Random unary utility – $U : D \mapsto [0..1]$
- Random binary hardness parameter – $L \in [1..10]$
- Random binary utility with $L+1$ values, equally distributed in the range $[0,1]$ – $U : D \times D \mapsto [0, \frac{1}{L}, \frac{2}{L}, \ldots, 1]$
- Random forbidden pairs matrix, each value is 0 with probability $\frac{1}{L+1}$ – $F(d_i, d_j) = \begin{cases} 1 & U(d_i, d_j) > 0 \\ 0 & \text{otherwise} \end{cases}$
- Random number of resource types – $m \in [1..20]$
- Random resource mapping – $R : D \mapsto [1..m]$
- Random bounds over each resource type – $B_i \in [1..30]$, for $i \in [1..m]$

6.1.2 Anytime behavior

Figure 6.1 shows a comparison of the performance profile of the algorithms [Zil96], using a timeout of 1 second, over 50 random inputs. We can see that there is a cluster of similar anytime behavior of hybrid algorithms, i.e., algorithms that used the greedy as an initial solution. The cluster's quality is high, but there is a low improvement trend long before it reaches the timeout. Algorithms which do not use the greedy as an initial solution have diverse qualities. *Simulated Annealing* and *Stochastic Hill Climbing* reach the highest score among the non-hybrid algorithms. Our basic *Random Search* and *Random Walk* are inferior to the other algorithms, as expected.

6.2 Automatic Parameters Tuning

An algorithm's parameters have a significant impact on its performance, and a systematic tuning is less prone to biases. There are several efficient methods and tools for automatic parameters tuning (or algorithm configuration): *GGA* [AST09] uses genetic algorithms to search the configurations space. *irace* [LIDLC$^+$16] uses iterated racing i.e. maintains a distribution from which it samples configurations and update the distribution according to the samples' performance. *SMAC* [HHLB11] iterates between learning a model from algorithm's runs, selecting configurations from the model and comparing them to the best known configuration. *ParamILS* [HHLBS09] uses iterated local search in the space of configurations. *ParamILS* is probably the most widely used and cited state-of-the-art tool for parameter tuning. It has been used mainly to reduce the run time of algorithms. It has dozens of academic applications e.g.,[HBHH07], [HHLB10], [VFG$^+$11] and several industrial applications, on which it yielded significant speed-ups over state-of-the-art solvers.

6.2.1 Automatic Parameters Tuning for RAFP

We used *ParamILS* for automatic tuning of the parameters of the algorithms that we implemented. Since we are solving in real-time, the objective of the tuning is to get the best quality at a given timeout. One benefit of a real-time algorithm is that for a given offline time budget, it can be tuned much more than regular algorithms.
First, we used the same 50 random inputs that we used before, with a tuning process of 1500 seconds, in order to tune the parameters of the algorithms for the best quality at a 1 second timeout. $RandomSearch$, $Greedy$, $Greedy + RandomSearch$ and $Greedy_{LOOP}$ did not participate in the tuning process since they have no parameters to tune. Figure 6.2 shows an anytime behavior which is comparable to the behavior we saw in figure 6.1. We can see that all the non-hybrid algorithms improve their quality more than the hybrid algorithms. *Simulated Annealing* and *Stochastic Hill Climbing* now reach a quality which is higher than the hybrid cluster. The *Cross-Entropy Method* has a quite low quality at the timeout but the biggest improvement trend.

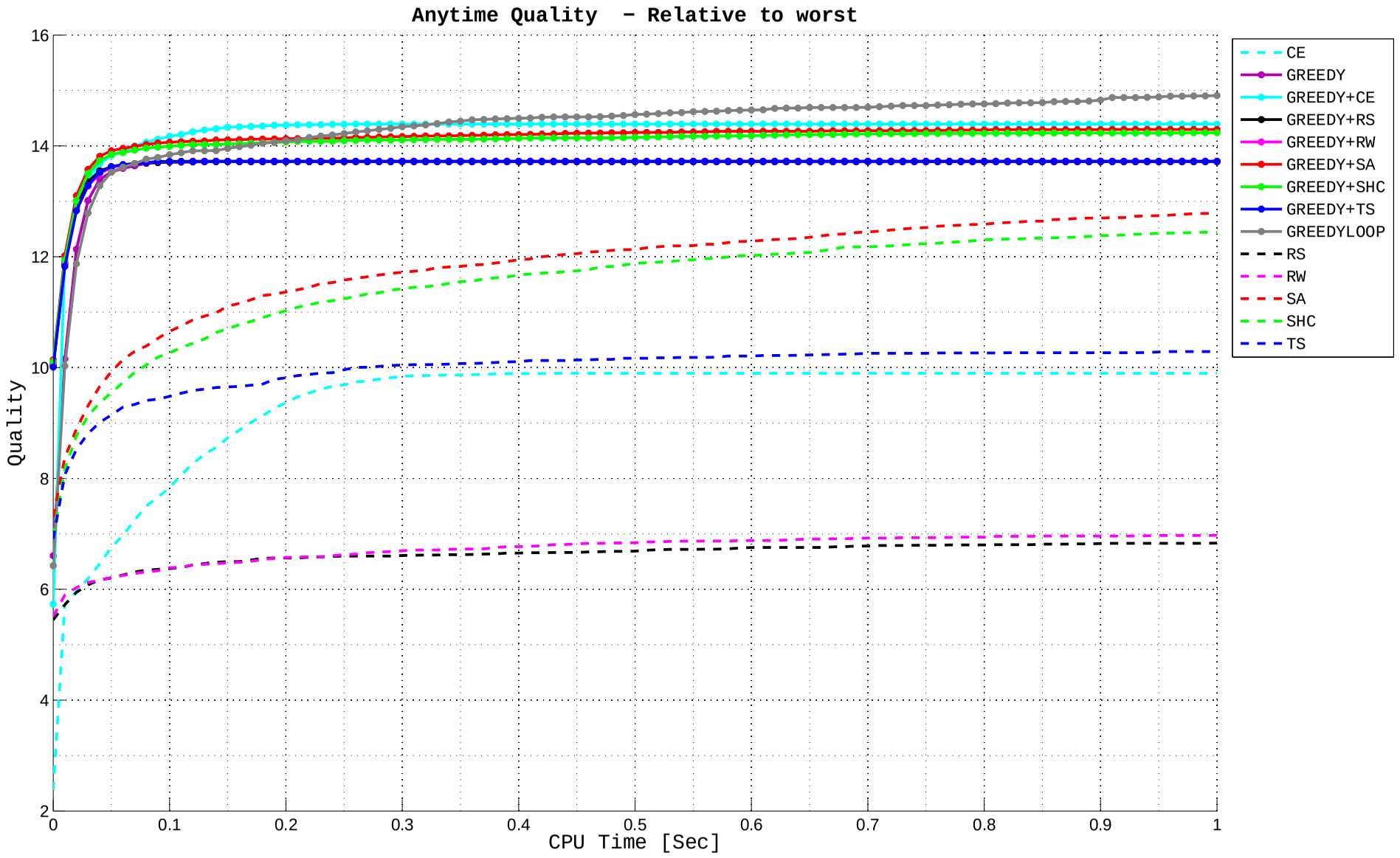


Figure 6.1: Anytime behavior - Mean over 50 random inputs, before tuning

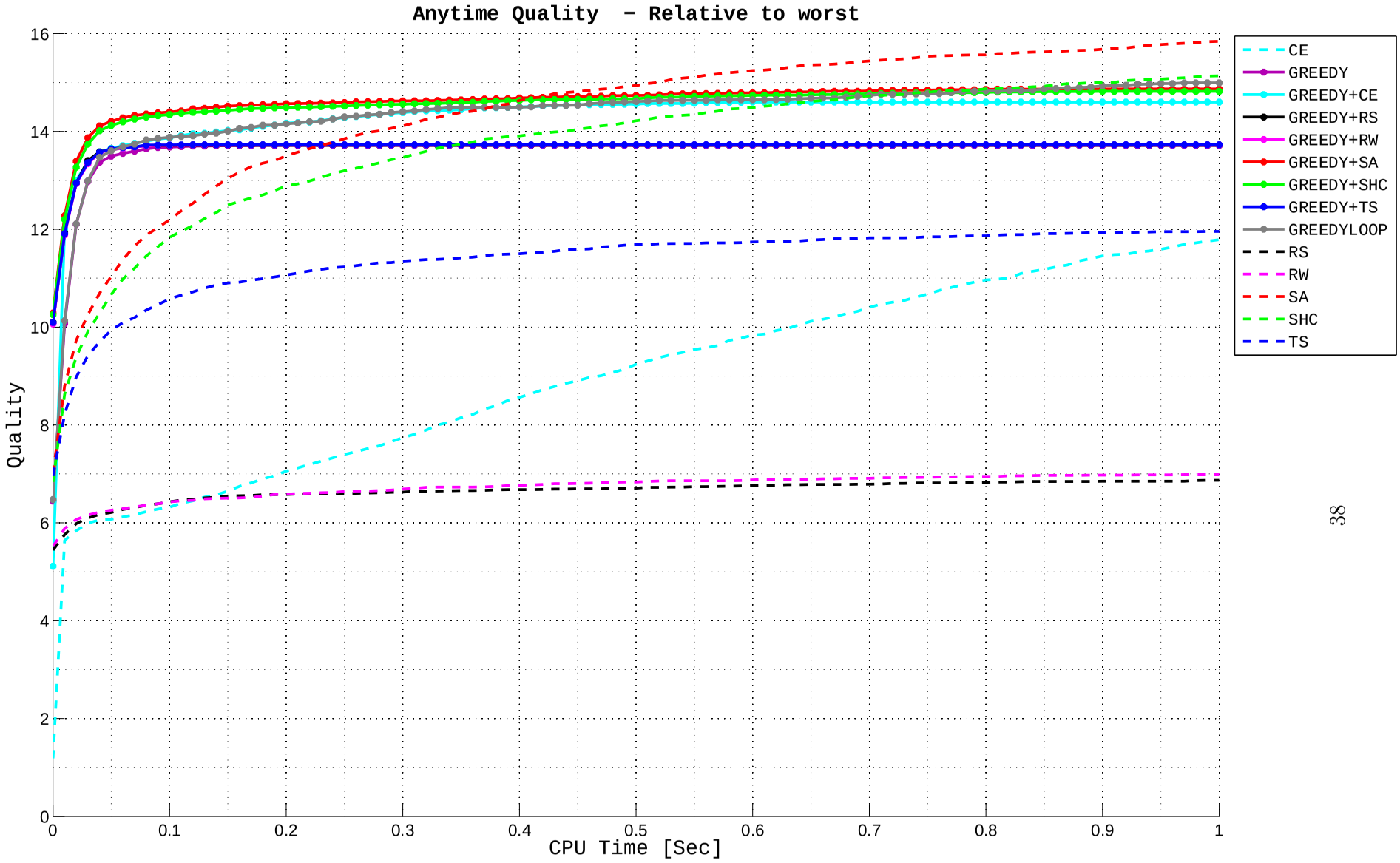


Figure 6.2: Anytime behavior - Mean over 50 random inputs, after tuning

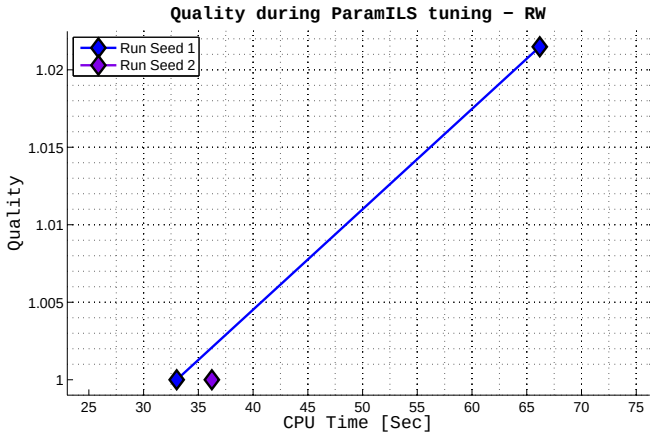
In order to focus on real-time scales, we now shift out focus to the quality of the algorithms at timeouts of 0.1 seconds. In the tuning process, we used 10 random inputs, a tuning time of 1500 seconds, and a timeout of 0.1 seconds for each algorithm.

6.2.2 Tuning of a Single Algorithm

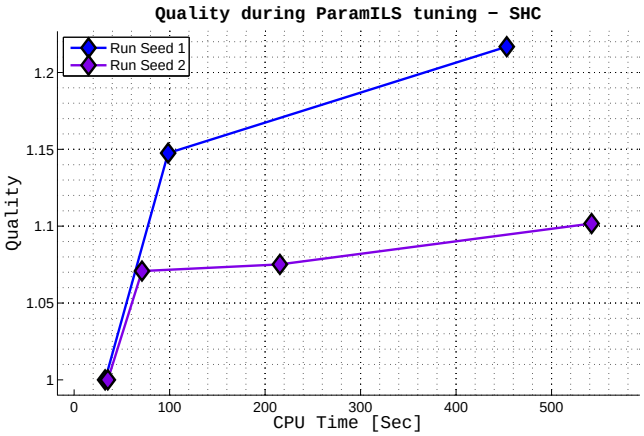
The results of the algorithms' tuning are shown in figures 6.3 (no initial solution) and 6.4 (hybrid). In each figure we can see the quality of one algorithm normalized to its initial quality, over two separate runs. Each run of *ParamILS* finds a different configuration by using a different seed. The two versions of each tunable algorithm will help us create a diverse set of algorithms for the later build of a portfolio. We ran all algorithms for the same tuning time, and the graphs end at the last time of improvement. We can see that the tuning process yields a quality improvement of up to 35%, which is a significant improvement.

6.2.3 Comparison of Algorithms During Tuning

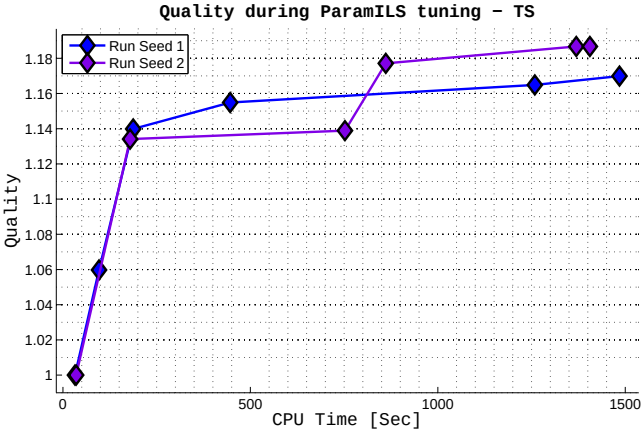
After we saw each algorithm's improvement by itself, we want to see the comparison between algorithms, including the untunable ones. The results of algorithms comparison are shown in figures 6.5 (First version of all algorithms) and 6.6 (Second version of all algorithms). Both figures include the untunable algorithms which have only one version. We can see that the ranking between algorithms is changing during the tuning process, and that it is a similar ranking when comparing all first versions and all second versions of the algorithms. The leading algorithms at the end of the tuning are *Greedy+SA*, *Greedy+SHC* and *Greedy+CE*.



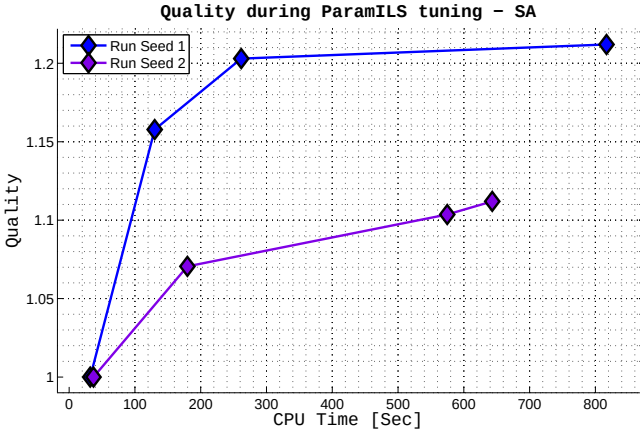
(a) Random Walk



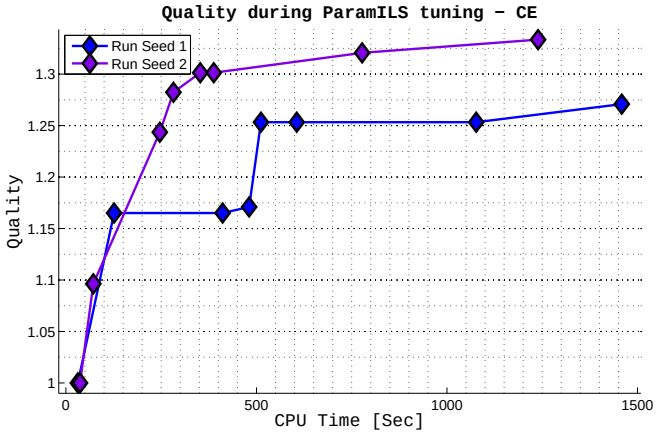
(b) Stochastic Hill Climbing



(c) Tabu Search

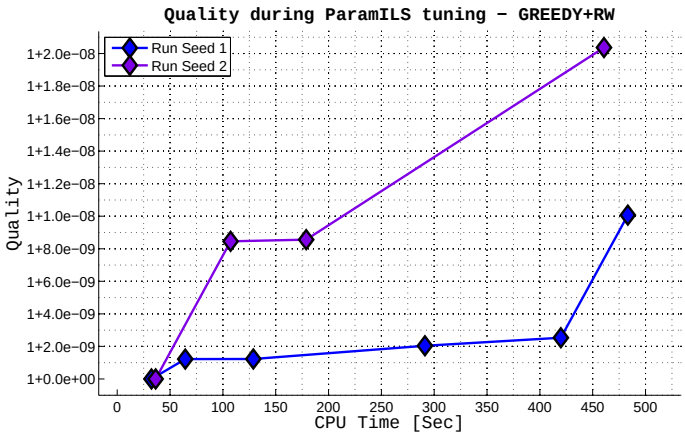


(d) Simulated Annealing

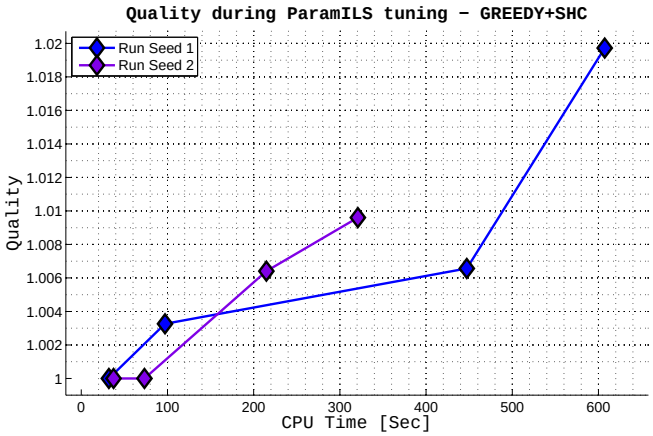


(e) The Cross Entropy Method

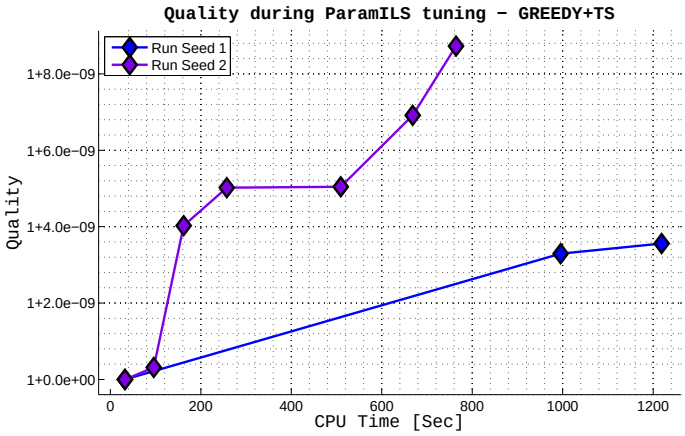
Figure 6.3: Automatic Parameters Tuning: Initial solution = None, Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L \in [1..10]$



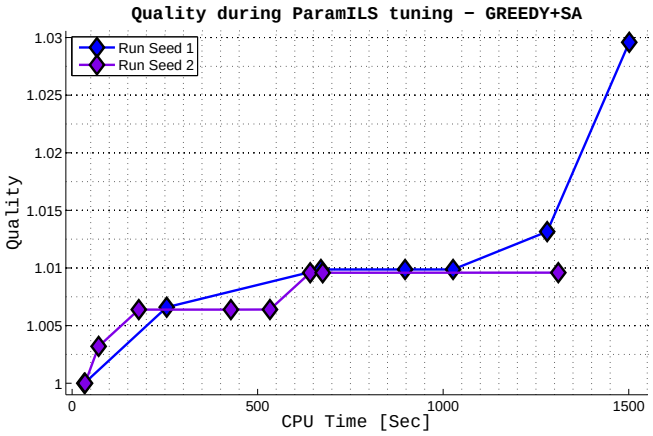
(a) Random Walk



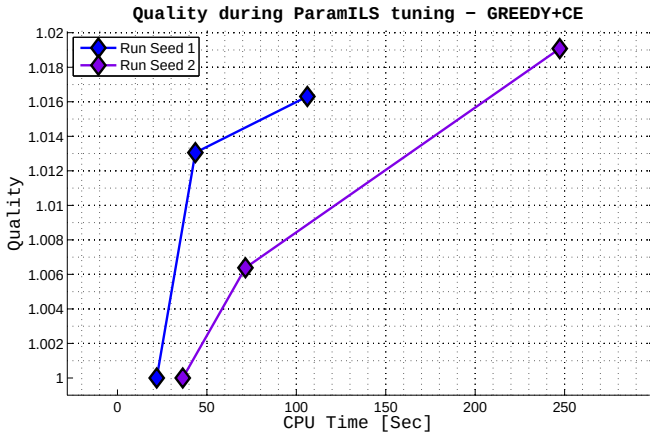
(b) Stochastic Hill Climbing



(c) Tabu Search



(d) Simulated Annealing



(e) The Cross Entropy Method

Figure 6.4: Automatic Parameters Tuning. Initial solution = Greedy, Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L \in [1..10]$

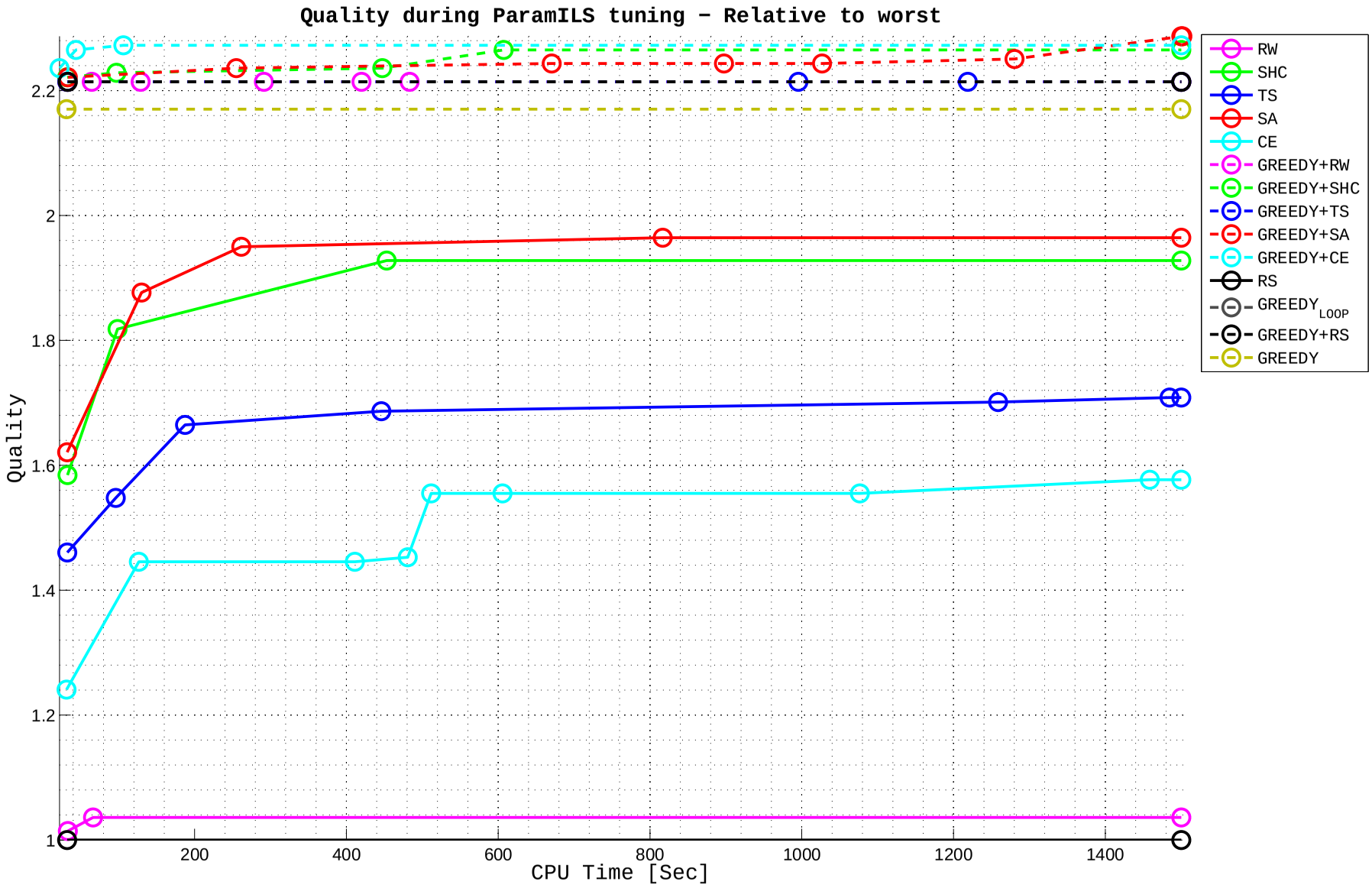


Figure 6.5: Automatic Parameters Tuning - Ranking Version 1: Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L \in [1..10]$

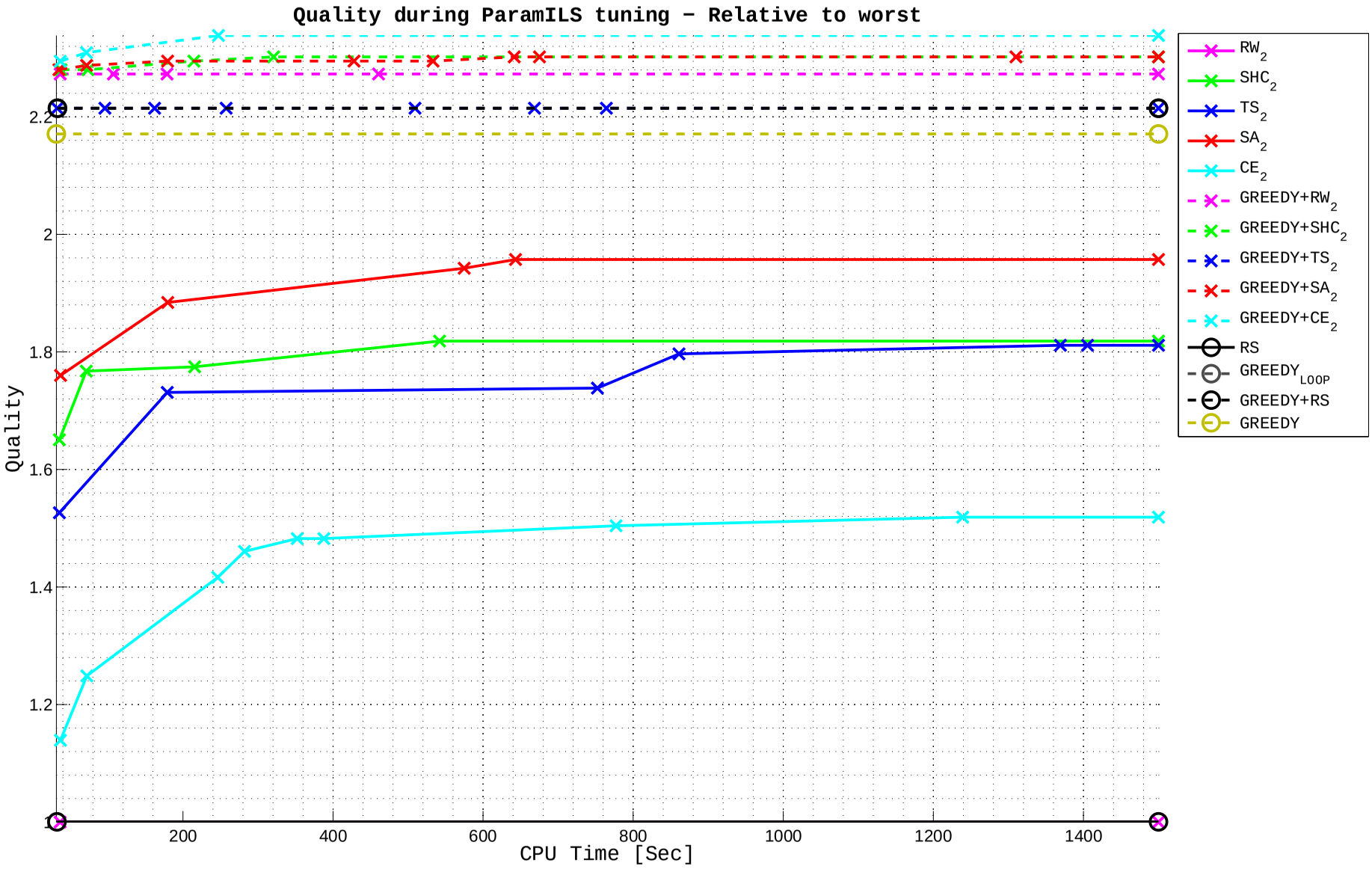


Figure 6.6: Automatic Parameters Tuning - Ranking Version 2: Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L \in [1..10]$

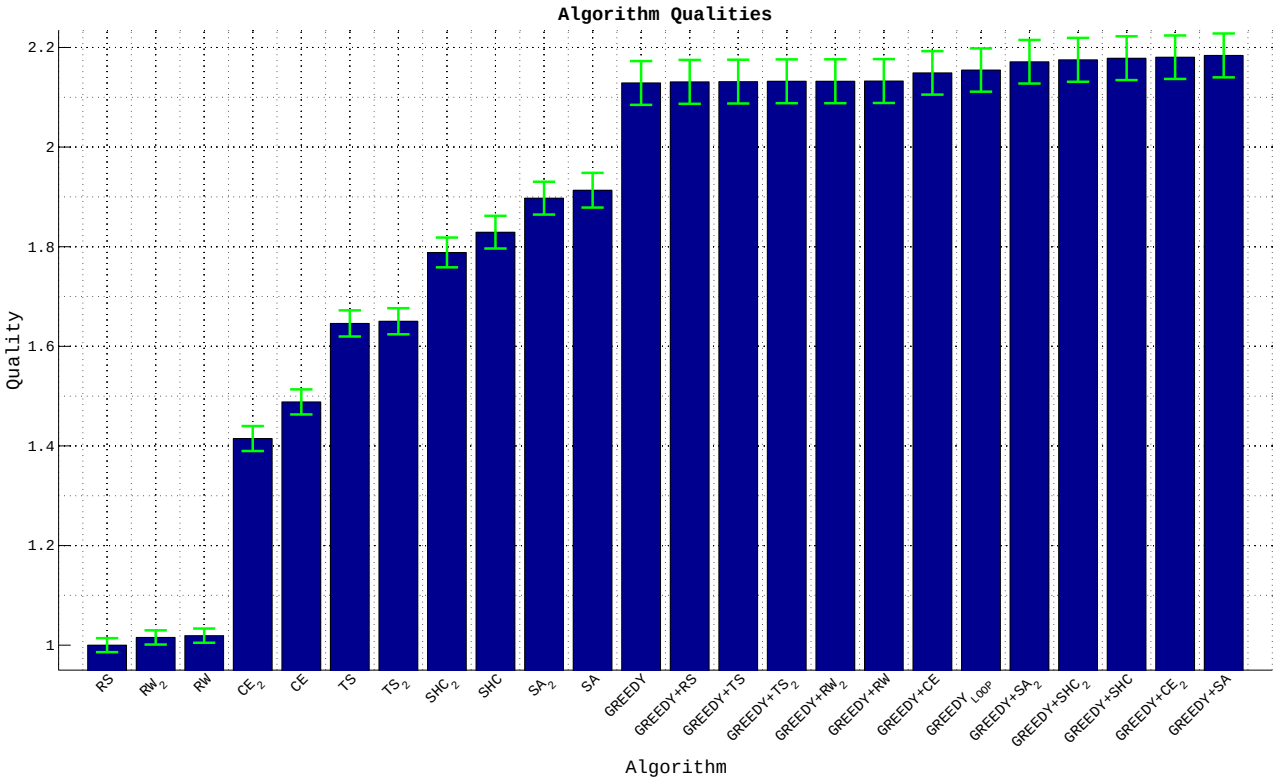


Figure 6.7: Automatic Parameters Tuning - Validation Qualities: Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L \in [1..10]$

6.3 Validation of Final Configurations

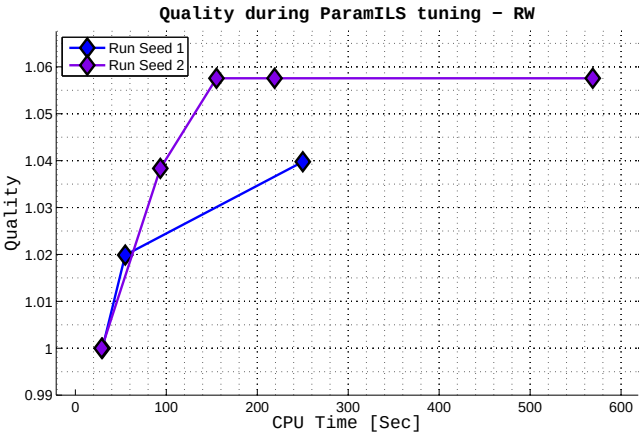
The validation is the process of a deeper evaluation of the final configuration. Here we used 500 random inputs, in the same timeout of 0.1 second. The results of the validation are shown in figure 6.7. We can see that the two versions of the leading algorithms at the end of the tuning process (*Greedy+SA*, *Greedy+SHC* and *Greedy+CE*) are leading in the validation process too. This is a sign that the training problems represented well the test problems. We can also see that the quality of the best algorithms is about 2.2 times than the quality of the worst algorithm, *Random Search*.

6.4 Harder Problems

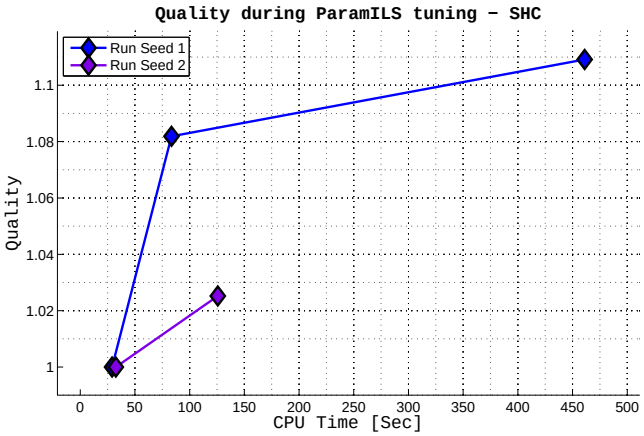
The random inputs described in 6.1.1 are very diverse. Next, we focus on problems which we created using $L = 1$. This makes the forbidden pairs matrix F get uniform random values from $\{0, 1\}$. Thus, we have a probability of 0.5 that any pair of tactics is possible in a solution. This is our heuristic method of creating harder problems. We repeated the tuning and comparison process for these problems, and the results are shown in figures 6.8-6.12. Generally, we can see in figures 6.8 and 6.9 that the maximum tuning profit is about 19%, less than the tuning profit over general problems, but still significant. Figures 6.10 and 6.11 show a ranking of algorithms over hard problems in which several non-hybrid algorithms take the first places during the tuning process. In figure 6.12 we can see the quality of the various algorithms in the validation process. We can see a smaller span of qualities when comparing to general problems: a factor of 1.6 between the best and the worst quality of algorithms. Several non-hybrid algorithms are better than some hybrid algorithms, and the the non-hybrid simulated annealing is not far from the best algorithm.

6.5 Algorithms Configurations

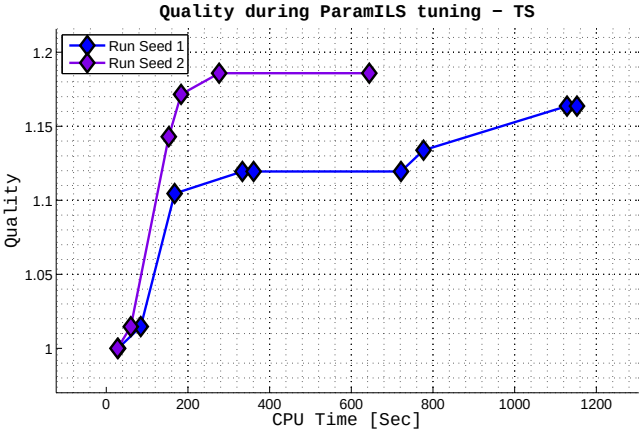
Tables 6.1-6.5 describe for each algorithm its tunable parameters, their range and initial configuration. Then for each version of algorithm and problem set (general or hard) we can see the final tuned configurations.



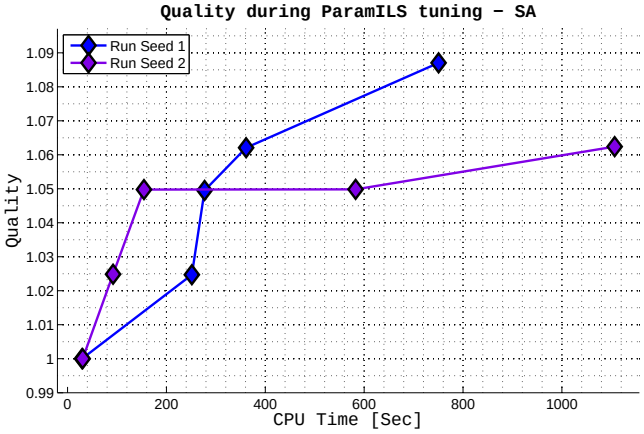
(a) Random Walk



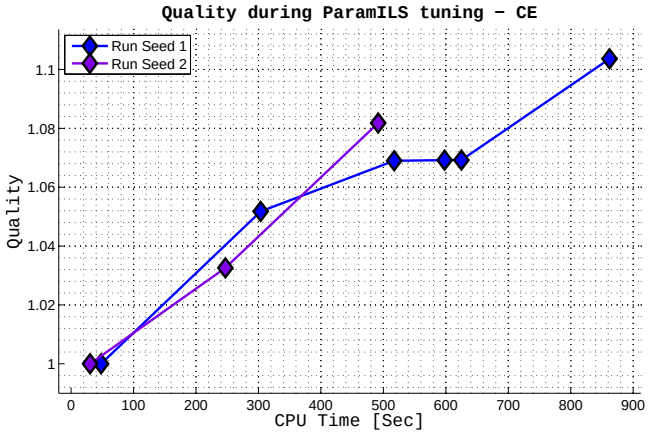
(b) Stochastic Hill Climbing



(c) Tabu Search

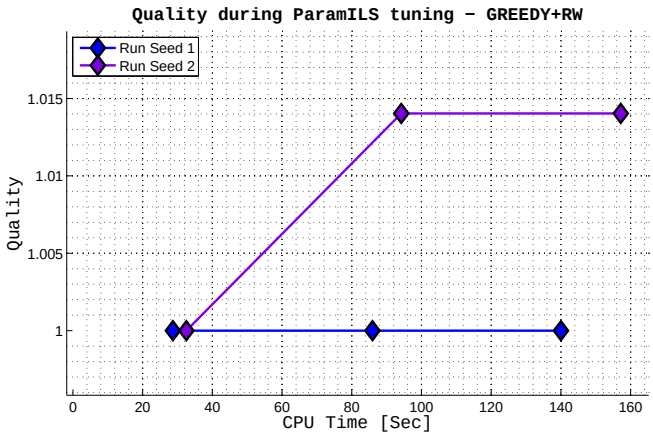


(d) Simulated Annealing

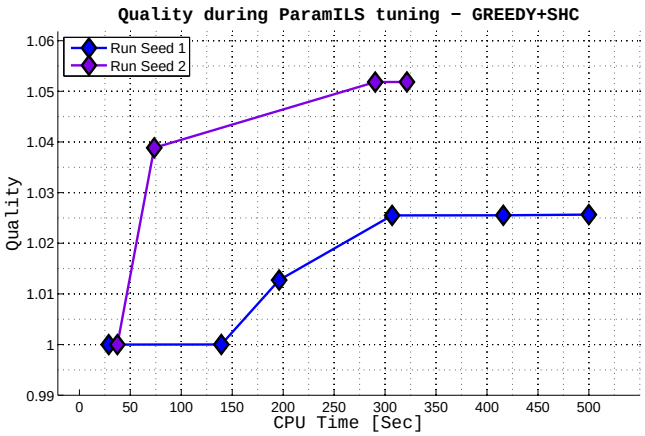


(e) The Cross Entropy Method

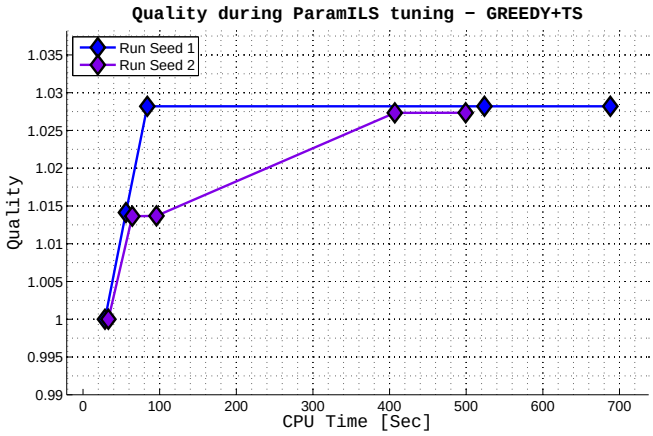
Figure 6.8: Initial solution = None, Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, L=1



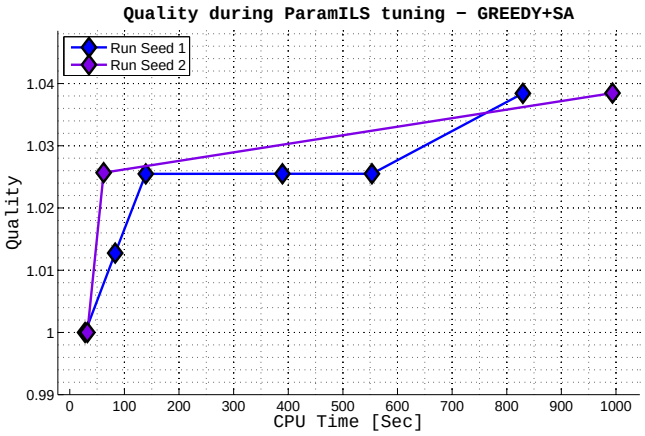
(a) Random Walk



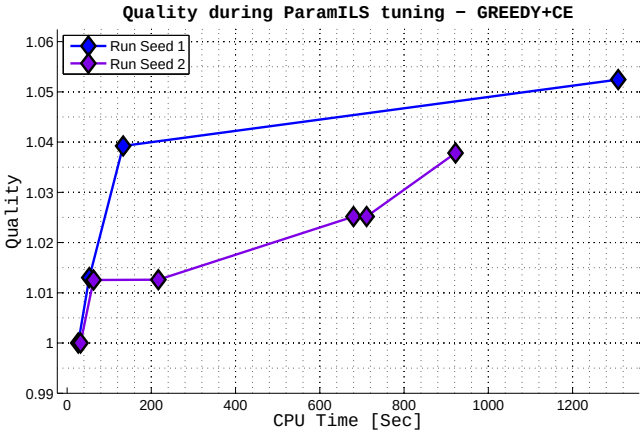
(b) Stochastic Hill Climbing



(c) Tabu Search



(d) Simulated Annealing



(e) The Cross Entropy Method

Figure 6.9: Automatic Parameters Tuning. Initial solution = Greedy, Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, L=1

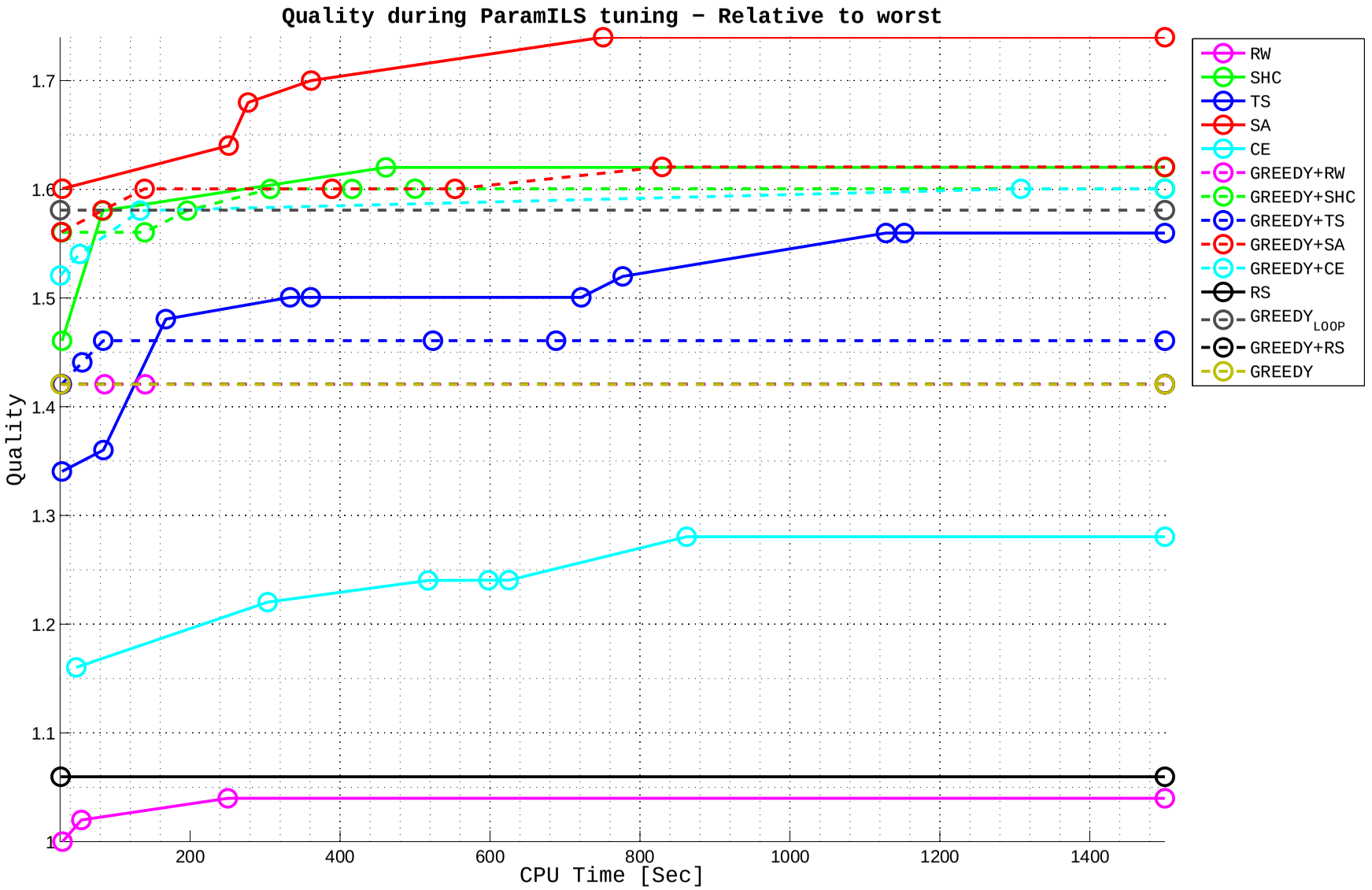


Figure 6.10: Automatic Parameters Tuning - Ranking Version 1: Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L = 1$

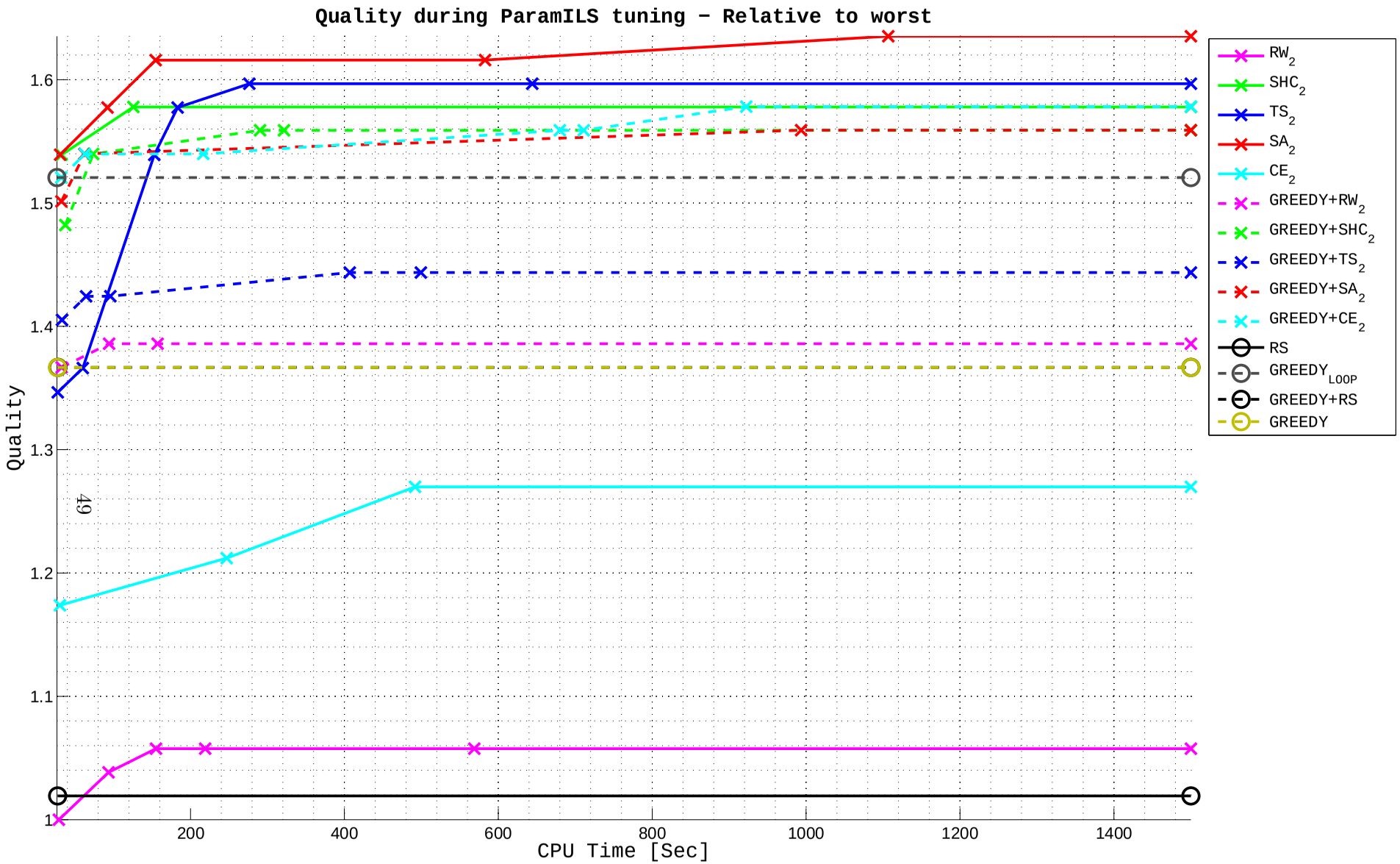


Figure 6.11: Automatic Parameters Tuning - Ranking Version 2: Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L = 1$

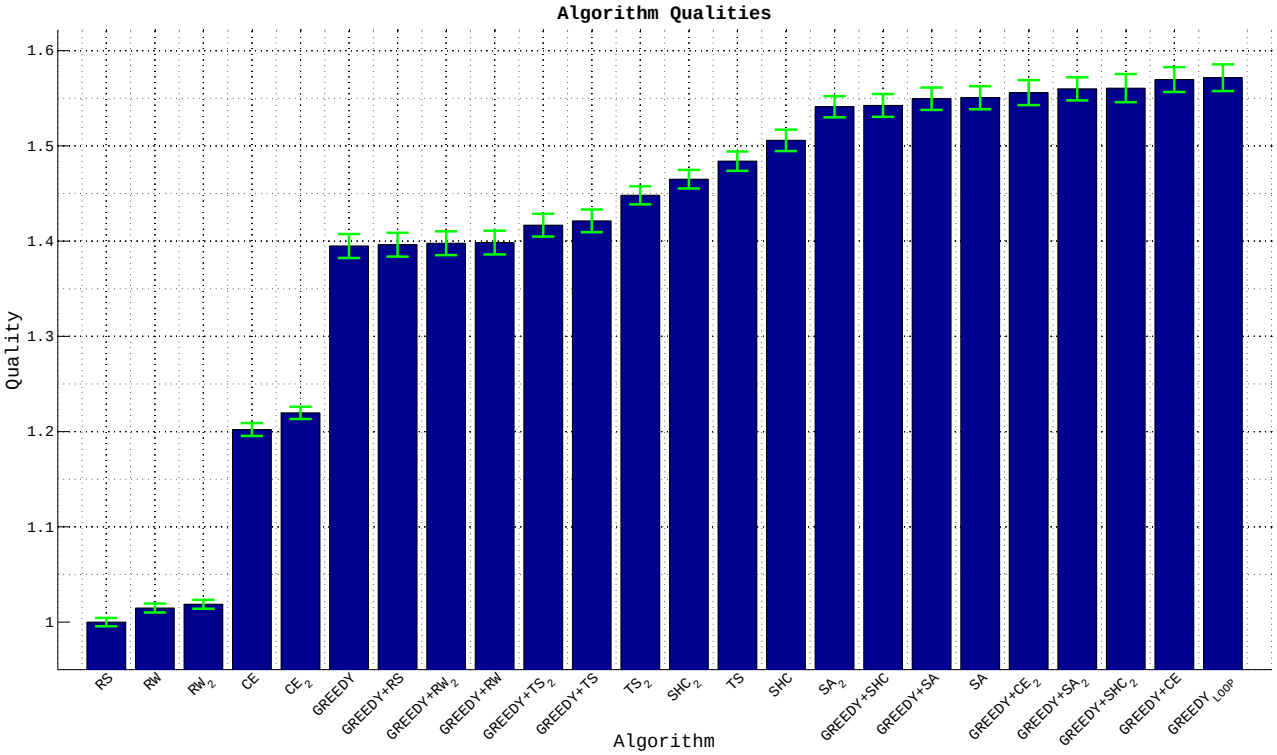


Figure 6.12: Automatic Parameters Tuning - Validation Qualities: Benchmarks = 10, Tuning time = 1500 sec, Timeout = 0.1 sec, $L = 1$

Parameter	Range	Initial config.	Final config. 1 RW	Final config. 2 RW	Final config. 1 Greedy+RW	Final config. 2 Greedy+RW	Final config. 1 RW hard	Final config 2 RW hard	Final config. 1 Greedy+RW hard	Final config. 2 Greedy+RW hard
Neighborhood	$1, 2, \ldots, 20$	10	16	10	2	7	20	18	9	15

Table 6.1: Automatic Parameters Tuning - Random Walk Parameters

Parameter	Range	Initial config.	Final config. 1 SHC	Final config. 2 SHC	Final config. 1 Greedy+SHC	Final config. 2 Greedy+SHC	Final config. 1 SHC hard	Final config 2 SHC hard	Final config. 1 Greedy+SHC hard	Final config. 2 Greedy+SHC hard
Neighborhood	$1, 2, \ldots, 20$	10	2	4	1	1	6	13	19	1

Table 6.2: Automatic Parameters Tuning - Stochastic Hill Climbing Parameters

Parameter	Range	Initial config.	Final config. 1 TS	Final config. 2 TS	Final config. 1 Greedy+TS	Final config. 2 Greedy+TS	Final config. 1 TS hard	Final config 2 TS hard	Final config. 1 Greedy+TS hard	Final config. 2 Greedy+TS hard
Neighborhood	$1, 2, \ldots, 20$	10	9	9	9	8	11	17	16	14
Tabu size	$10, 20, \ldots, 200$	100	180	180	30	110	150	150	100	110

Table 6.3: Automatic Parameters Tuning - Tabu Search Parameters

Parameter	Range	Initial config.	Final config. 1 SA	Final config. 2 SA	Final config. 1 Greedy+SA	Final config. 2 Greedy+SA	Final config. 1 SA hard	Final config 2 SA hard	Final config. 1 Greedy+SA hard	Final config. 2 Greedy+SA hard
Neighborhood	$1, 2, \ldots, 20$	10	2	3	1	3	4	5	18	16
Init. Temperature	$100, 10, \ldots, 0.0001$	0.1	0.01	100	0.0001	1	0.1	0.0001	0.0001	0.001
Temperature Step	$11, 1.1, \ldots, 1.000001$	1.001	11	11	1.1	1.1	1.1	11	11	1.01

Table 6.4: Automatic Parameters Tuning - Simulated Annealing Parameters

Parameter	Range	Initial config.	Final config. 1 CE	Final config. 2 CE	Final config. 1 Greedy+CE	Final config. 2 Greedy+CE	Final config. 1 CE hard	Final config 2 CE hard	Final config. 1 Greedy+CE hard	Final config. 2 Greedy+CE hard
#Samples	$10, 20, \ldots, 100$	50	70	40	90	50	90	90	90	40
Initial Solution's Weight	$0.0, 0.1, 0.2, \ldots, 0.9$	0.5	0.0	0.0	0.5	0.9	0.0	0.0	0.5	0.9
Smoothing Factor (α)	$0.1, 0.2, \ldots, 0.9$	0.5	0.5	0.7	0.7	0.1	0.4	0.4	0.7	0.1
Elite Factor (ρ)	$0.1, 0.2, \ldots, 0.9$	0.5	0.1	0.4	0.5	0.1	0.2	0.1	0.2	0.3
Convergence distance (ϵ)	$0.1, 0.01, \ldots, 0.00001$	0.001	0.00001	0.1	0.00001	0.0001	0.1	0.01	0.1	0.0001

Table 6.5: Automatic Parameters Tuning - The Cross Entropy Method Parameters

Chapter 7

Constructing The Best Portfolio

Using a *parallel portfolio* of algorithms [HLH97] can significantly improve the performance relative to a single algorithm [GS01]. The idea of a *static* portfolio [PZ06] is to run several algorithms in parallel, on separate cores, and after the time bound T has elapsed, take the best result achieved by any of those algorithms. The same portfolio is used for all inputs. A *dynamic* portfolio is a portfolio that can adapt per input [MS12]. Two automated methods for constructing a static portfolio are described in [HLBSS12]. These methods integrate the tuning process with the construction of the portfolio, and was used to reduce the computation time of SAT-solving. The first method was to treat a portfolio of algorithms as one algorithm with a configuration space of all its components and tune this one algorithm to get the best quality. The second and less exhaustive method, was a greedy method of using a highly parameterized single algorithm, start with an empty portfolio and tune one algorithm instance at a time, to find the best parameters we can, in order to improve the quality of the growing portfolio. As any greedy method for a computationally hard problem, it is not optimal since the results depend on the order of tuning.

We will focus on static parallel portfolios, composed of different algorithms, which are already tuned. We also assume that the algorithms do not communicate during their run. This process is modular and easily allows to get the best out of a given set of algorithms. As before, our portfolio measure relates to the quality of the portfolio when reaching the short and fixed timeout, and it cannot be based on an on-line time consuming process like pre-runs, in order to keep the real-time requirements. We are now interested in the following problem: Given a set of n algorithms, $k < n$ cores and a set of benchmarks, how to choose the best parallel portfolio of k algorithms according to some performance measure. This parallel portfolio is called *virtual best solver* [XHHLB12]. We suggest a general method of solving this problem optimally.

7.1 Constructing a Portfolio as an Optimization Problem

Choosing a bounded subset of algorithms in order to achieve optimum for a given performance measure is a special case of the subset selection problem [QYZ15]. We define several variants of this problem.

7.2 K-Algorithms Cover Problems

7.2.1 Definitions

Definition 7.2.1. (K-Algorithms Cover Problem). An instance of the k-algorithms cover problem is a 5-tuple $\langle S, I, M, m, k \rangle$, where:

- S is a set of n algorithms
- I is a set of inputs for S
- $M : S \times I \mapsto \mathbb{R}$ is the quality of solution that each algorithm in S returns with each input in I
- $m_I : \mathcal{P}(S) \mapsto \mathbb{R}$ is a portfolio measure over I, of a parallel portfolio $s \subseteq S$, which satisfies the property $m_{\{i\}}(\{A\}) = M(A, i)$, for $i \in I, A \in S$
- $k < n$ is the number of algorithms to choose from S

A solution to the instance $\langle S, I, M, m, k \rangle$ is a parallel portfolio of k algorithms from S with the best performance, according to m_I:

$$s^* = argmax_{s \subset S : |s| = k}(m_I(s)) \tag{7.1}$$

or

$$s^* = argmin_{s \subset S : |s| = k}(m_I(s)) \tag{7.2}$$

Definition 7.2.1 allows to define the best performance measure m_I as a maximum or minimum, according to the context. In the follwing definition the performance measure has to be maximized.

Definition 7.2.2. (K-Algorithms Max-Sum Problem) An instance of the k-algorithms max-sum problem is an instance of the k-algorithms cover problem with the following portfolio measure:

$$m_I(s) = \sum_{i \in I} \left(\max_{A \in s} M(A, i) \right) \tag{7.3}$$

In words, the objective of a k-algorithms cover problem with this measure is to maximize the portfolio's sum of qualities across benchmarks.

In the follwing definition the performance measure has to be minimized.

Definition 7.2.3. (K-Algorithms Min-Max-Gap Problem) An instance of the k-algorithms min-max-gap problem is an instance of the k-algorithms cover problem with the following portfolio measure:

$$m_I(s) = \max_{i \in I} \left(\min_{A \in s} Gap(A, i, S) \right) \tag{7.4}$$

Where

$$Gap(a, i, S) = \max_{A \in S}(A, i) - M(a, i) \tag{7.5}$$

In words, the objective of a k-algorithms cover problem with this measure is to minimize the portfolio's worst gap to the optimal algorithm in S, across benchmarks.

7.2.2 Examples

Example 7.2.4. Let $\langle S, I, M, m, k \rangle$ be an instance of the K-Algorithms Cover Problem, with its first three components S, I, M defined as follows:

- $S = \{A_1, A_2, A_3\}$, thus $n = 3$
- $I = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7\}$
- M is defined using the following table:

	i_1	i_2	i_3	i_4	i_5	i_6	i_7
A_1	1	4	3	4	3	4	3
A_2	1	3	4	3	4	3	4
A_3	3	3	3	3	3	3	3

Now, for the K-algorithms max-sum problem, $m_I(s)$ is given in (7.3).

if $k = 1$ we have:

$$m_I(\{A_1\}) = \sum_{i \in I} \left(\max_{A \in \{A_1\}} M(A, i) \right) = \sum_{i \in I} M(A_1, i) = 22 \tag{7.6}$$

$$m_I(\{A_2\}) = \sum_{i \in I} \left(\max_{A \in \{A_2\}} M(A, i) \right) = \sum_{i \in I} M(A_2, i) = 22 \tag{7.7}$$

$$m_I(\{A_3\}) = \sum_{i \in I} \left(\max_{A \in \{A_3\}} M(A, i) \right) = \sum_{i \in I} M(A_3, i) = 21 \tag{7.8}$$

Thus $\{A_1\}$ or $\{A_2\}$ is the best 1-portfolio in this case.
If $k = 2$, we have:

$$m_I(\{A_1, A_2\}) = \sum_{i \in I} \left(\max_{A \in \{A_1, A_2\}} M(A, i) \right) = 1 + 4 + 4 + 4 + 4 + 4 + 4 = 25 \tag{7.9}$$

$$m_I(\{A_1, A_3\}) = \sum_{i \in I} \left(\max_{A \in \{A_1, A_3\}} M(A, i) \right) = 3 + 4 + 3 + 4 + 3 + 4 + 3 = 24 \quad (7.10)$$

$$m_I(\{A_2, A_3\}) = \sum_{i \in I} \left(\max_{A \in \{A_2, A_3\}} M(A, i) \right) = 3 + 3 + 4 + 3 + 4 + 3 + 4 = 24 \quad (7.11)$$

Thus $\{A_1, A_2\}$ the best 2-portfolio in this case. For the K-algorithms min-max-gap, $m_I(s)$ is computed according to (7.4), where $Gap(A, i, S)$ is computed from M using (7.5):

	i_1	i_2	i_3	i_4	i_5	i_6	i_7
A_1	2	0	1	0	1	0	1
A_2	2	1	0	1	0	1	0
A_3	0	1	1	1	1	1	1

if $k = 1$ we have:

$$m_I(\{A_1\}) = \max_{i \in I} \left(\min_{A \in \{A_1\}} Gap(A, i, S) \right) = \max\{2, 0, 1, 0, 1, 0, 1\} = 2 \quad (7.12)$$

$$m_I(\{A_2\}) = \max_{i \in I} \left(\min_{A \in \{A_2\}} Gap(A, i, S) \right) = \max\{2, 1, 0, 1, 0, 1, 0\} = 2 \quad (7.13)$$

$$m_I(\{A_3\}) = \max_{i \in I} \left(\min_{A \in \{A_3\}} Gap(A, i, S) \right) = \max\{0, 1, 1, 1, 1, 1, 1\} = 1 \quad (7.14)$$

Thus $\{A_3\}$ is the best 1-portfolio in this case.

if $k = 2$, we have

$$m_I(\{A_1, A_2\}) = \max_{i \in I} \left(\min_{A \in \{A_1, A_2\}} Gap(A, i, S) \right) = \max\{2, 0, 0, 0, 0, 0, 0\} = 2 \quad (7.15)$$

$$m_I(\{A_1, A_3\}) = \max_{i \in I} \left(\min_{A \in \{A_1, A_3\}} Gap(A, i, S) \right) = \max\{0, 0, 1, 0, 1, 0, 1\} = 1 \quad (7.16)$$

$$m_I(\{A_2, A_3\}) = \max_{i \in I} \left(\min_{A \in \{A_2, A_3\}} Gap(A, i, S) \right) = \max\{0, 1, 0, 1, 0, 1, 0\} = 1 \quad (7.17)$$

Thus $\{A_1, A_3\}$ or $\{A_2, A_3\}$ is the best 2-portfolio in this case.

7.3 Minimum Algorithms Cover Problems

For a given k, it is possible that the same performance can be achieved with $k' < k$. A multi-core machine should use k' cores to achieve maximum performance. More cores do not contribute at all and might be used for other purposes. Another conclusion might be that we should develop more diverse algorithms in order to utilize the multi-core machine.
We define now the problem of finding the smallest best portfolio of algorithms. It is equivalent to definition 7.2.1, except that k is a property of the solution and not a parameter, and we allow $k = n$.

7.3.1 Definitions

Definition 7.3.1. (Minimum Algorithms Cover Problem). An instance of the minimum algorithms cover problem is a 4-tuple $\langle S, I, M, m \rangle$, where:

- S is a set of n algorithms
- I is a set of benchmarks for S
- $M : S \times I \to \mathbb{R}$ is the quality of solution that each algorithm in S returns with each input in I
- $m_I : \mathcal{P}(S) \to \mathbb{R}$ is a portfolio measure over I, of a parallel portfolio $s \subseteq S$, which satisfies the property $m_{\{i\}}(\{A\}) = M(A, i)$, for $i \in I, A \in S$

A solution to the instance $\langle S, I, M, m \rangle$ is a parallel portfolio of algorithms from S with the best performance, and $|s^*|$ of minimal size, where

$$s^* = argmax_{s \subseteq S}(m_I(s)) \tag{7.18}$$

or

$$s^* = argmin_{s \subseteq S}(m_I(s)) \tag{7.19}$$

Minimum Algorithms Cover Problem can be defined with portfolio measures according to definitions 7.2.2 and 7.2.3, to get the Minimum Algorithms Max Sum Problem and the Minimum Algorithms Min-Max-Gap Problem, respectively.

7.3.2 Examples

Example 7.3.2. Let $\langle S, I, M, m \rangle$ be an instance of the Minimum Algorithms Cover Problem, with the following properties:

- $S = \{A_1, A_2, A_3\}$, thus $n = 3$
- $I = \{i_1, i_2\}$

- M is defined using the following table:

	i_1	i_2
A_1	1	0
A_2	0	1
A_3	0.8	0.7

- $m_I(s) = \sum_{i \in I} (\max_{A \in s} M(A, i))$

For $k = 1$ we have

$$m_I(\{A_1\}) = 1 + 0 = 1 \tag{7.20}$$

$$m_I(\{A_2\}) = 0 + 1 = 1 \tag{7.21}$$

$$m_I(\{A_3\}) = 0.8 + 0.7 = 1.5 \tag{7.22}$$

Thus $\{A_3\}$ is the optimal solution.
For $k = 2$ we have

$$m_I(\{A_1, A_2\}) = 1 + 1 = 2 \tag{7.23}$$

$$m_I(\{A_1, A_3\}) = 1 + 0.7 = 1.7 \tag{7.24}$$

$$m_I(\{A_2, A_3\}) = 0.8 + 1 = 1.8 \tag{7.25}$$

Thus, $\{A_1, A_2\}$ is the optimal solution. As we can see, since $\{A_3\} \not\subset \{A_1, A_2\}$, the solutions are not monotonic.
For $k = 3$ we have

$$m_I(\{A_1, A_2, A_3\}) = 1 + 1 = 2 \tag{7.26}$$

Thus we can conclude that the minimum k for maximum portfolio measure is 2, and the minimum portfolio is $\{A_1, A_2\}$.
Generally, the fact that two portfolios with consecutive sizes share the same quality (in our example 2-portfolio and 3-portfolio), does not imply that the portfolio has reached its optimum quality, as we will see in chapter 8.

7.4 Modeling the K-Algorithms Cover Problem with SMT

The Satisfiability Modulo Theories problem (SMT) [KBS10] is to decide the satisfiability of a first-order formula over some decidable theories. Among theories in use we find: Arithmetics [DDM06], the theories of bit-vectors [BDL98], arrays [McC62] and equality of uninterpreted functions [Ack]. There is an extensive research in the field of SMT, and SMT solvers that participate in contests manage to solve large problems within reasonable run times. Next, we represent our algorithms cover problems using SMT with the theory of Quantifier-Free Linear Real Arithmetic (QF_LRA). This theory enables representation of Boolean formulas of inequalities between linear polynomials, using real variables. For example, consider the following QF_LRA formula: $F = (x \geq -2) \vee (y \geq -1 \wedge y \leq 5)$. F contains two real variables x and y, that are used in predicates within a Boolean formula. The set of solutions to this formula is a union of two areas in the x-y plane.

7.4.1 Modeling the K-Algorithms Max-Sum Problem with QF_LRA

The following encoding is the SMT representation of the K-Algorithms Max-Sum Problem.

The *real variables* are V_i for $i \in I$,which represent the quality of the portfolio over benchmark i.

The *Boolean decision variables* are A_i for $i \in [1..n]$, which represent whether algorithm A_i is chosen for the optimal k-portfolio.

The *objective* is to maximize the sum of qualities across the benchmarks and is defined as:

$$\max \sum_{i=1}^{|I|} V_i \tag{7.27}$$

Constraints (7.28)-(7.30) below are connected by a *logical and* ($\wedge$).

The *value choice constraints* allow a choice of quality for each benchmark, according to a chosen algorithm, using M:

$$\forall i \in I : (V_i = M(A_1, i)) \vee (V_i = M(A_2, i)) \vee \ldots \vee (V_i = M(A_n, i)) \tag{7.28}$$

The *implied algorithm constraints* connect between the chosen quality of a benchmark and the algorithms that return this quality:

$$\forall i \in I, V \in \{M(A_j, i)\}_{j \in \{1..n\}} : (V_i = V) \rightarrow \bigvee_{A \in S : M(A,i) = V} A \tag{7.29}$$

The *algorithms cardinality constraints* make sure that the number of chosen algorithms of a portfolio will be k, when we take *true=1* and *false=0*:

$$\sum_{i=1}^{n} A_i = k \tag{7.30}$$

Such constraints are not allowed in QF_LRA, but they can be reduced to propositional logic, which is allowed in QF_LRA. We used the encoding suggested in [Sin05] for this purpose.

In the above SMT representation, 7.28–7.30 allow the choice of exactly k algorithms, with values from M. When combined with 7.27, the maximum sum of k-subsets will be produced when $V_i = \max_{A \in s} M(A, i)$, thus the model expresses measure 7.3.

7.4.2 Modeling the K-Algorithms Min-Max-Gap Problem with QF_LRA

The following encoding is the SMT representation of the K-Algorithms Min-Max-Gap Problem.

The *real variables* are V_i for $i \in I$,as in subsection 7.4.1.
In addition, the single real variable V is used to hold the maximum of V_i's.

The *Boolean decision variables* are A_i for $i \in [1..n]$, as in subsection 7.4.1.

The *objective* is to minimize the worst gap across benchmarks, marked by V:

$$\min V \tag{7.31}$$

Constraints (7.32)-(7.35) below are connected by a *logical and*($\wedge$).

The *maximum constraints* ensure that V is the maximal (worst) gap across benchmarks:

$$\forall i \in I : V \geq V_i \tag{7.32}$$

The *value choice constraints* role and encoding is identical to (7.28):

$$\forall i \in I : (V_i = Gap(A_1, i, S)) \vee (V_i = Gap(A_2, i, S)) \vee \ldots \vee (V_i = Gap(A_n, i, S)) \tag{7.33}$$

The *implied algorithm constraints* role and encoding is identical to (7.29):

$$\forall i \in I, V \in \{Gap(A_j, i, S)\}_{j \in \{1..n\}} : (V_i = V) \rightarrow \bigvee_{A \in S : Gap(A,i,S) = V} A \tag{7.34}$$

The *algorithm cardinality constraints* role and encoding is identical to (7.30):

$$\sum_{i=1}^{n} A_i = k \tag{7.35}$$

In the above SMT representation, 7.33–7.35 allow the choice of exactly k algorithms, with values defined by 7.5. When combined with 7.32, V becomes the maximum of V_i values. The minimum of maximum gaps from the optimal solution over all k-subsets algorithms values will be produced when $V_i = \min_{A \in s} Gap(A, i, S)$, thus the model expresses the measure 7.4.

Chapter 8

Empirical Results: Portfolios

8.1 SMT Modeling

We implemented an SMT modeling program with the following interface, as described in chapter 7:

- Input:
  - A matrix M, where M_{ij} represents the quality of the solution that algorithm $i \in [1..n]$ returns over input j
  - A number $k < n$ of algorithms to choose
- Output:
  - SMT encoding of the k-Algorithms Max-Sum problem
  - SMT encoding of the k-Algorithms Min-Max-Gap problem

8.2 SMT Solving

Some of the most competitive SMT solvers are *CVC4* [BCD+11], *Yices* [Dut14] and *Z3* [dMB08]. We chose to use *Z3*. *Z3* is a high-performance SMT solver developed by Microsoft Research. It is used in various software verification applications [CDH+09], [Lei10], [WPF+10]. *Z3* supports various theories, including the *quantifier-free linear real arithmetic* (QF_LRA) theory in our setting.

8.3 Portfolios Construction – Tuned Algorithms

We built the matrix M from the results of the 24 tuned algorithms over the inputs that we described in chapter 6, for both portfolio measures that we described in chapter 7.

8.3.1 Three Portfolio Models

We built three portfolios, for each $k \in [1..24]$:

1. Optimal – Uses the program-generated SMT encoding for k and solves with *Z3*
2. Greedy – Chooses the best algorithm to complete a partial portfolio, for k iterations
3. K-Best – Sorts the algorithms by quality, and takes the first k algorithms.

The greedy portfolio is a natural choice since we have a special case of the subset selection problem. The K-Best portfolio is one which does not use the information about the quality of each algorithm per-instance, thus we expect it to be inferior to the two other portfolios. Obviously any method of portfolio construction eventually reaches the optimum if $k = n$.

8.3.2 Results

The results of the three portfolios are shown in figures 8.1-8.4. In the Max-Sum figures the qualities sum, which have to be maximized, is normalized to the best single algorithm's quality. In the Min-Max-Gap the qualities max-gap, which have to be minimized, is normalized to the best non-zero quality. This normalization influences the improvement factors that we describe, since we are using percents. Each figure describes the number of benchmarks, the timeout for a single algorithm's run, and L, the hardness parameter that we used and described in chapter 6.

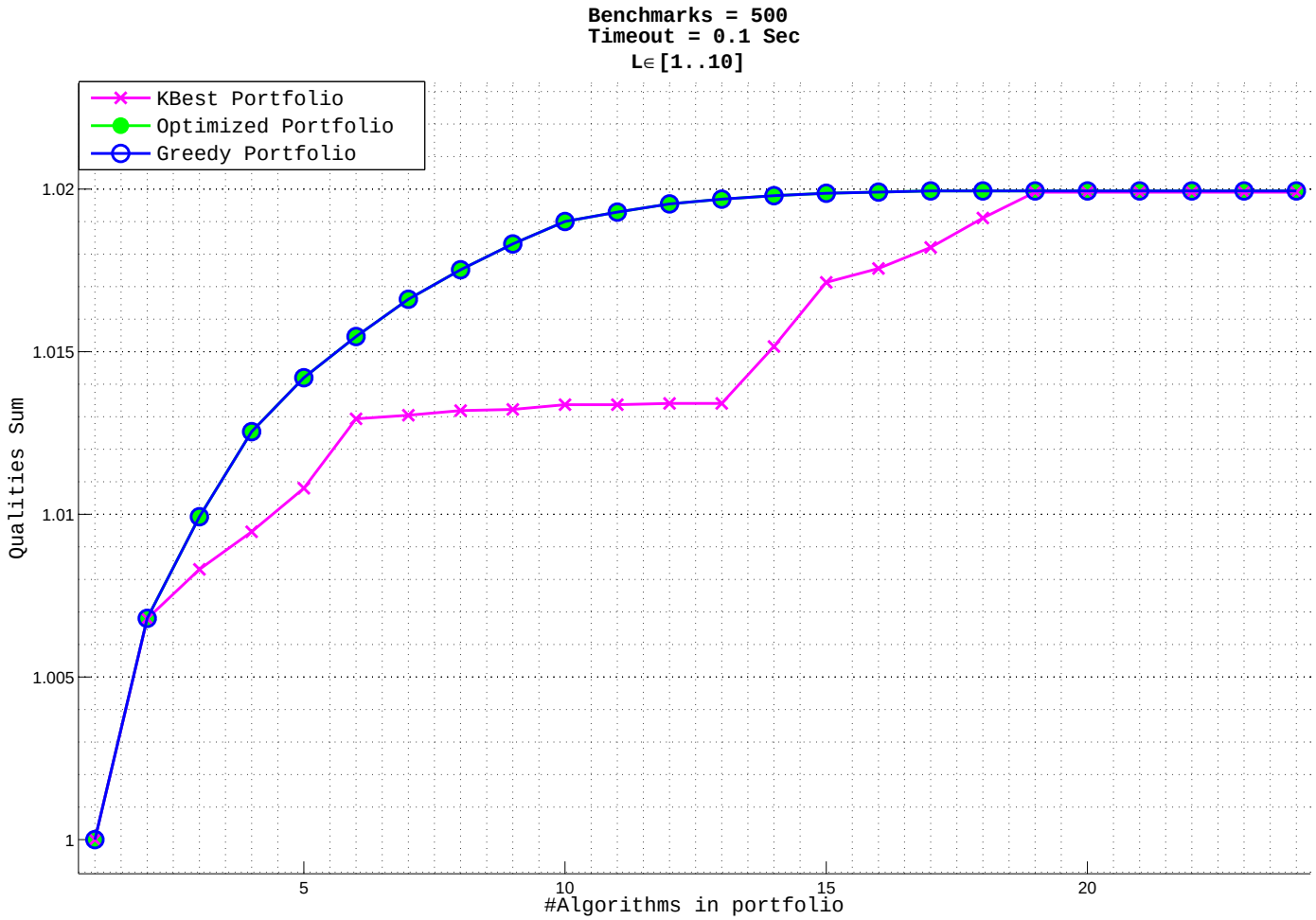


Figure 8.1: Max-Sum portfolios. The optimal portfolio reaches its maximum improvement of about 2% at a portfolio size of 17, with a decreasing rate of improvement. This does not guarantee convergence, and as far as we know convergence is identified only when a partial portfolio reaches the quality of the full portfolio. The greedy portfolio's quality is almost identical to the optimal portfolio. The K-Best portfolio's quality is lower as expected, and it reaches the optimum quality at a portfolio size of 19.

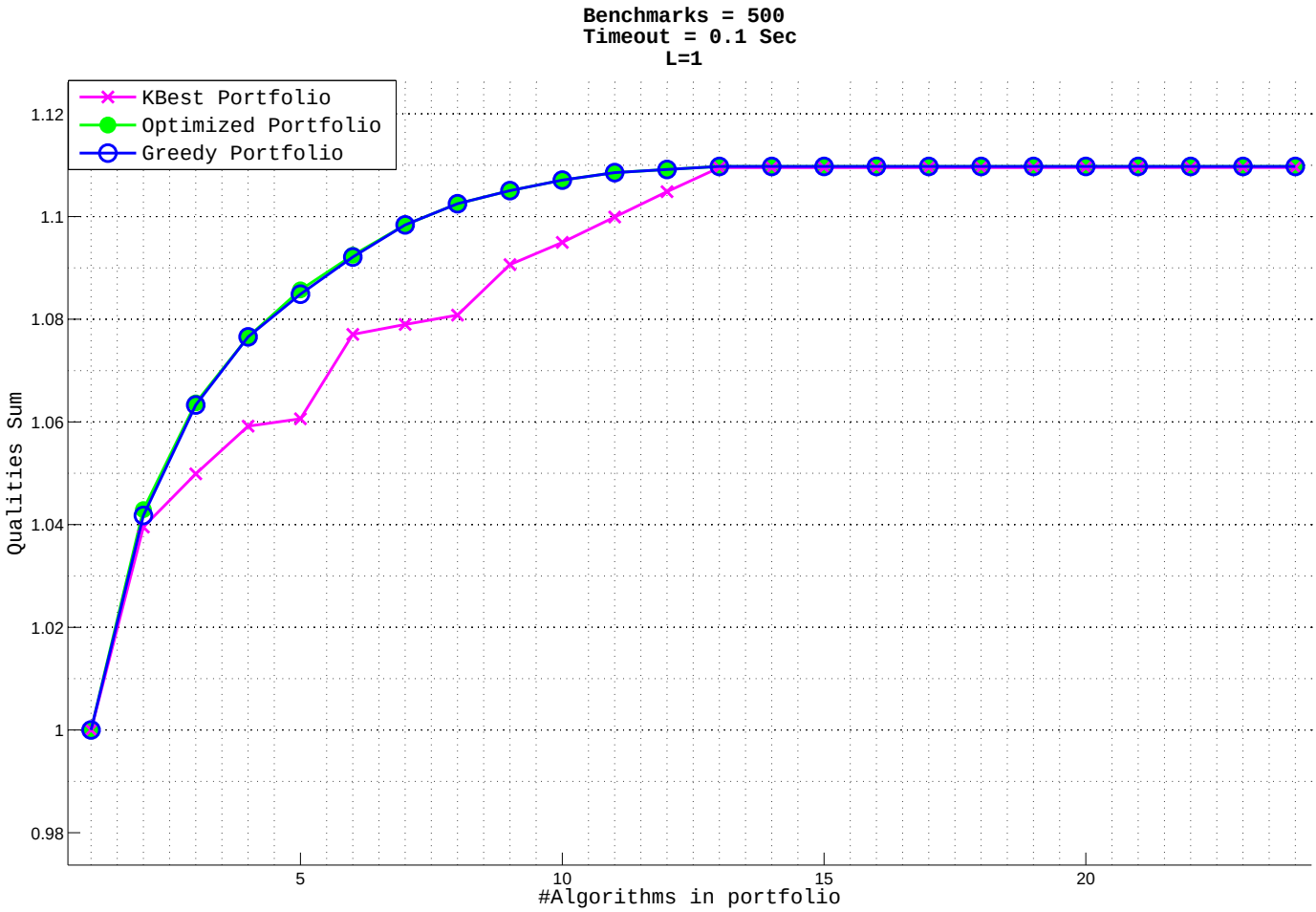


Figure 8.2: Max-Sum portfolios for hard problems (L=1).The optimal portfolio reaches its maximum improvement of 11% at a portfolio size of 15. This is not apparent from the graph but can be seen in the numerical results. The 11% improvement is higher than the improvement we saw in the portfolio over general problems. Again, the greedy portfolio almost coincide with the optimal portfolio, and the K-best portfolio is inferior, and it reaches the optimal portfolio only at a portfolio size of 13.

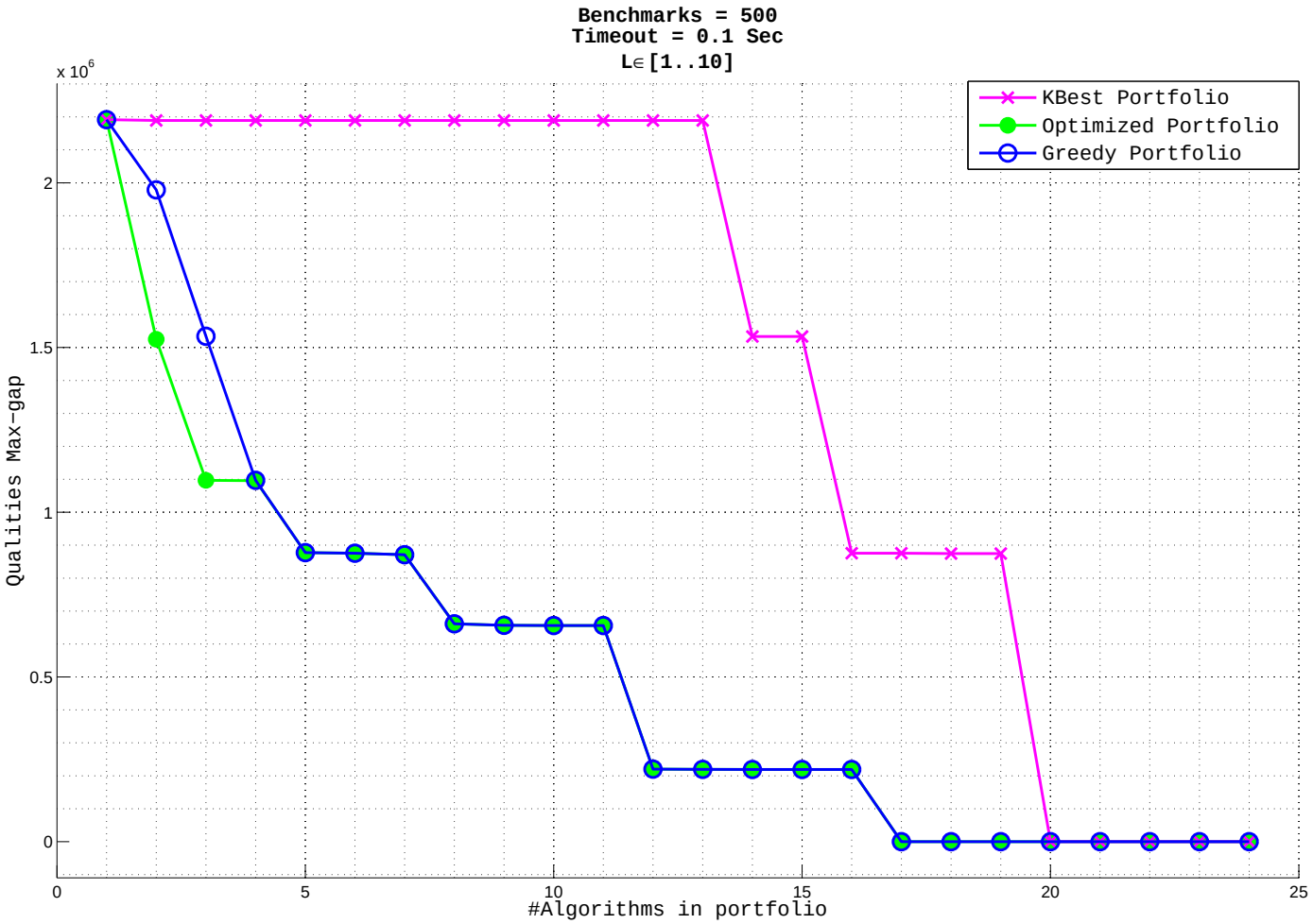


Figure 8.3: Min-Max-Gap portfolios. The optimal portfolio is of size 17. The best portfolio with non-zero quality gap is the one in size 16 and it is more than 2 million times better than using the single best algorithm, as we can see at a portfolio size of 1, for all the portfolio construction methods. In the optimum size of the optimal portfolio (17), the cost is 0, by definition. This shows that using less cores than the minimum number for optimal performance has a high cost in the min-max-gap measure, more than it has in the max-sum measure. We can also see that the greedy portfolio is not meeting the optimal cost for two portfolio sizes, and it fits less to this measure. Last, we can see that the K-best portfolio is far from the optimal performance, reaching the optimum only at a portfolio size of 20.

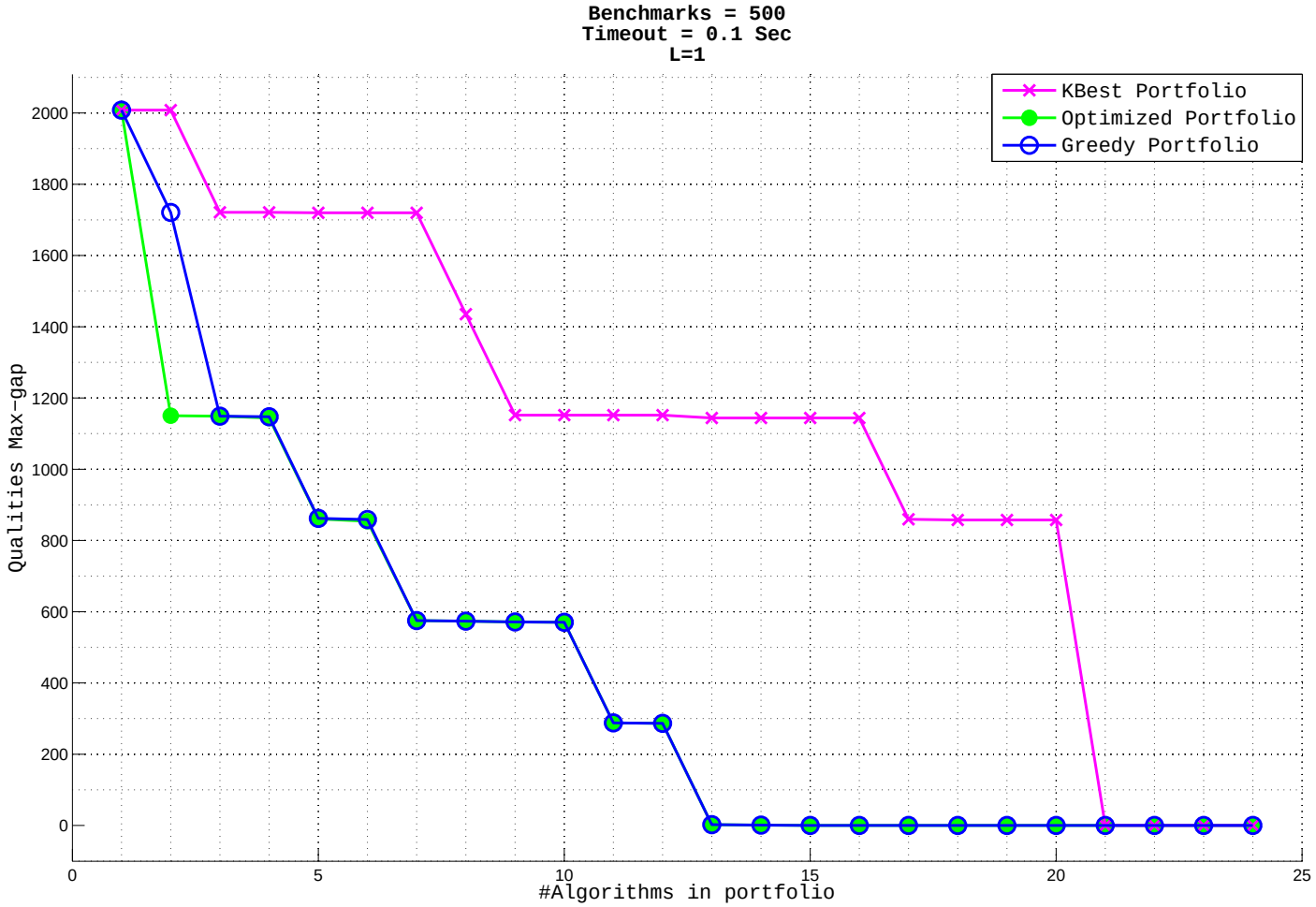


Figure 8.4: Min-Max-Gap portfolios for hard problems (L=1). The optimal portfolio is reaching optimum in a portfolio size of 15. This is not apparent from the graph but can be seen in the numerical results. For the min-max-gap over hard problems, the best portfolio with non-zero quality gap is about 2000 times better than the best single algorithm, which is a very significant result. The greedy portfolio is following the optimal portfolio, except for a portfolio size of 2. The K-best portfolio is far from the optimum and reaching it only in a portfolio size of 21.

Tables 8.1-8.10 describe the algorithmic components of the optimal portfolio, for the max-sum and min-max-gap measures, for general and hard problems. One example for the lack of monotonicity in the optimal portfolios can be found in portfolio's sizes of 1 and 2 in table 8.4. In Appendix A we describe a computation of an optimal max-sum portfolio that we conducted using random matrices. The results in the appendix are not comparable to the results that we showed in this chapter, since the quality axis in the appendix is normalized to the range [0,1].

Portfolio Size	1	2	3	4	5	6	7
Portfolio members	$Greedy + SA$	$Greedy + SA$ $Greedy + CE_2$	$Greedy + SA$ $Greedy + CE_2$ SA_2	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA

Table 8.1: Max-Sum Optimal Portfolios - Part 1

Portfolio Size	8	9	10	11	12	13
Portfolio members	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC SHC_2	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC SHC_2 $Greedy + TS_2$

Table 8.2: Max-Sum Optimal Portfolios - Part 2

Portfolio Size	14	15	16	17
Portfolio members	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC SHC_2 $Greedy + TS_2$ $Greedy + CE$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC SHC_2 $Greedy + TS_2$ $Greedy + CE$ $Greedy + RW$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC SHC_2 $Greedy + TS_2$ $Greedy + CE$ $Greedy + RW$ $Greedy + RS$	$Greedy + SA$ $Greedy + CE_2$ SA_2 $Greedy_{LOOP}$ TS $Greedy + SA_2$ SA $Greedy + SHC$ TS_2 $Greedy + SHC_2$ SHC SHC_2 $Greedy + TS_2$ $Greedy + CE$ $Greedy + RW$ $Greedy + RS$ $Greedy + RW_2$

Table 8.3: Max-Sum Optimal Portfolios - Part 3

Portfolio Size	1	2	3	4	5	6	7	8
Portfolio members	$Greedy_{LOOP}$	SA_2 $Greedy + SHC_2$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2	$Greedy_{LOOP}$ $Greedy + SA_2$ TS_2 SA	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS

Table 8.4: Max-Sum Optimal Portfolios, Hard Problems - Part 1

Portfolio Size	9	10	11	12	13	14	15
Portfolio members	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$ SHC	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$ SHC $Greedy + SHC$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$ SHC $Greedy + SHC$ $Greedy + SA$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$ SHC $Greedy + SHC$ $Greedy + SA$ $Greedy + CE_2$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$ SHC $Greedy + SHC$ $Greedy + SA$ $Greedy + CE_2$ $Greedy + RS$	$Greedy_{LOOP}$ $Greedy + SHC_2$ TS_2 SHC_2 SA SA_2 $Greedy + SA_2$ TS $Greedy + CE$ SHC $Greedy + SHC$ $Greedy + SA$ $Greedy + CE_2$ RS CE_2

Table 8.5: Max-Sum Optimal Portfolios, Hard Problems - Part 2

Portfolio Size	1	2	3	4	5	6	7
Portfolio members	$Greedy + CE_2$	$Greedy + SA$ SA_2	$Greedy + SA$ SA_2 SA	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2

Table 8.6: Min-Max-Gap Optimal Portfolios - Part 1

Portfolio Size	8	9	10	11	12	13
Portfolio members	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$ SHC_2	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$ SHC_2 $Greedy + TS_2$

Table 8.7: Min-Max-Gap Optimal Portfolios - Part 2

Portfolio Size	14	15	16	17
(r)1-7 Portfolio members	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$ SHC_2 $Greedy + TS$ $Greedy + RW_2$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$ SHC_2 $Greedy + TS_2$ $Greedy + RW_2$ $Greedy + CE$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$ SHC_2 $Greedy + TS_2$ $Greedy + RW_2$ $Greedy + CE$ $Greedy + RS$	$Greedy + SA$ SA_2 SA $Greedy_{LOOP}$ $Greedy + CE_2$ $Greedy + SHC_2$ TS_2 SHC TS $Greedy + SA_2$ $Greedy + SHC$ SHC_2 $Greedy + TS_2$ $Greedy + RW_2$ $Greedy + CE$ $Greedy + RS$ $Greedy + RW$

Table 8.8: Min-Max-Gap Optimal Portfolios - Part 3

Portfolio Size	1	2	3	4	5	6	7	8
Portfolio members	$Greedy + CE$	$Greedy_{LOOP}$ SHC	$Greedy_{LOOP}$ SHC $Greedy + CE$	$Greedy_{LOOP}$ $Greedy + SHC_2$ SHC TS_2	$Greedy_{LOOP}$ $Greedy + SHC$ SHC TS SA	$Greedy_{LOOP}$ $Greedy + CE_2$ SHC SHC_2 TS_2 SA	$Greedy_{LOOP}$ $Greedy + CE_2$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS

Table 8.9: Min-Max-Gap Optimal Portfolios, Hard Problems - Part 1

Portfolio Size	9	10	11	12	13	14	15
Portfolio members	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$ SA_2	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$ SA_2 $Greedy + SA_2$	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$ SA_2 $Greedy + SA_2$ $Greedy + CE_2$	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$ SA_2 $Greedy + SA_2$ $Greedy + CE_2$ $Greedy + SA$	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$ SA_2 $Greedy + SA_2$ $Greedy + CE_2$ $Greedy + SA$ $Greedy + RS$	$Greedy_{LOOP}$ $Greedy + CE$ SHC SHC_2 TS_2 SA $Greedy + SHC_2$ TS $Greedy + SHC$ SA_2 $Greedy + SA_2$ $Greedy + CE_2$ $Greedy + SA$ $Greedy + RS$ CE_2

Table 8.10: Min-Max-Gap Optimal Portfolios, Hard Problems - Part 2

Chapter 9

Conclusion

We summarize the thesis with a concise list of its contributions:

9.1 Contributions

- We described the problem of solving computationally hard problems in a short and fixed amount of time, and defined the notion of a *fixed-time variant* of a hard computational problem.
- We defined a particular optimization problem called *Resource Allocation with Forbidden Pairs* (RAFP) and showed that its decision variant is NP-complete. The rest of the thesis was dedicated to solving its fixed-time variant.
- We defined *Fixed-Time Search* as a heuristic framework for solving RAFP and similar discrete optimization problems in real-time. We presented over ten different known algorithms, most of which are based on local search (but also cross-entropy and a deterministic greedy algorithm), and their adaptation for solving RAFP.
- We used automatic tuning of parameters to get the best quality in a real-time setting, and showed empirically that it significantly improves the quality of all the algorithms.
- We defined *Algorithms Cover* problems. The first is the problem of choosing k out of n algorithms, $k < n$, to create the best parallel static portfolio with k cores, where a given set of inputs serve as the training set. The second is the problem of finding the smallest k for the best parallel static portfolio. These problems can be instantiated with a measure. We defined and used the *Max-Sum* and *Min-Max-Gap* measures.
- We suggested an SMT encoding for solving all the *Algorithm Cover* Problems optimally. We also compared the optimal result to one that is computed via a greedy polynomial method, which turned out to be, with our benchmarks, close to optimal for the max-sum measure, but sub-optimal for the min-max-gap measure.

- All the algorithms, their tuned versions and optimized portfolios, were tested empirically with hundreds of random instances with varying difficulty. The thesis contains a comprehensive empirical study of their effectiveness in solving RAFP in a short amount of time.

9.2 Future Work

We intend to explore several directions for improving the empirical results:

9.2.1 Individual Algorithms

Adding search algorithms From the algorithms that we implemented and described in Chapter 5, the *Cross-Entropy Method* and the *Tabu Search* can benefit from a re-design of the implementation. Specifically, the cross-entropy method may converge before time-out, and so far we did not use the extra time for restarting it. In the tabu search, the current conditions for a forbidden solution are still naive, and we intend to explore other variants. In addition, there are more anytime algorithms than we implemented, which are relevant for solving our problem, such as *Variable Neighborhood Search* [MH97], *Ant-Colony Optimization* [DMC96], *Genetic Algorithms* [Mit98] and *Monte Carlo Tree Search* [BPW$^+$12]. An anytime method that was used for *Weighted Constraint Satisfaction Problems* (see in appendix A) and is worth exploring is *Depth-First Branch and Bound* [Zha00]. Using this method will require a fast way to compute upper bounds on the quality of solutions that agree with a given partial solution.

9.2.2 Automatic Parameters Tuning

- **Tuning for a portfolio** So far we tuned each algorithm separately, despite the fact that our end goal is to find an optimal portfolio. To that end, we can let the choice of algorithms be a tunable parameter as well. This will require us to use the conditional parameters of ParamILS in order to tune only the parameters that correspond to the currently chosen algorithm.

- **Improving parameters tuning** In Chapter 6 we described our process of tuning the parameters of the algorithms. We use the same tuning time of 1500 seconds for each algorithm in *ParamILS*, with the *BasicILS* option of *ParamILS*. The *BasicILS* option compares between two configurations by their quality over the same number of runs. The objective of this uniform tuning is a fair tuning across algorithms. In order to achieve a better performance, it might be useful to use a tuning time which is proportional to the number of configurations. This way a large configuration space will get more tuning time. Moreover, the *FocusedILS* option of *ParamILS* might improve the results of the tuning process. The *FocusedILS* option compare between two configurations using a dominance concept which allows a different number of runs for each configuration. A last improvement might be achieved by a longer tuning period.

- **Tuning for anytime**: In Chapter 5 we described a tuning of parameters which is based on the quality at a specific timeout. Another approach for tuning will be to tune in order to improve the anytime behavior. One such tuning method is described in [RLIS13], where a measure similar to the area under the performance profile is the scalar value which guides the tuning process. This process cannot yield better performance for a specific timeout, but it might improve the overall behavior in a real-time interval of 1 second, for example.

- **Tuning & constructing portfolios** In Chapter 7 we described our serial process of tuning the algorithms and then constructing a portfolio. The greedy portfolio construction in [HLBSS12], which we also described in chapter 7, might lead to a better performance in our real-time setting.

9.2.3 Better Portfolio Construction

- **Dynamic portfolios** In Chapter 7, we described a construction of a static portfolio. A dynamic portfolio, i.e., a per-instance portfolio [MS12], might be better than a static one. This requires identifying features of problem instances, that on the one hand predict well the performance of various algorithms, and on the other are cheap to compute. To that end we may use machine learning methods in order to identify such features.

- **Collaborative portfolios** In Chapter 7, we focused on algorithms that do not communicate during their run. Communicating algorithms might be efficient in portfolios [BSS15], [SS12]. In our real-time setting, this can be relevant only if the cost of communication is low. Algorithms can share their best solutions, allowing others to use them as a starting point for further improvements.

9.2.4 Exploring More Real Time Issues

- **Flexible timeouts** Our algorithms can be tuned and analyzed with several different timeouts. Moreover, the model can be replaced with another reasonable model: We can add to the input of the problem a function which defines the utility of a specific computation time. This function can be combined with the quality of the solution to create a new combined quality. The combined quality has a maximum value, the point of the best trade-off between quality and its computation time. We can tune the algorithms in order to maximize the combined quality.

- **Considering Input's Noise** To consider uncertainty and inaccuracy of inputs, we suggest two ways: In the first, instead of representing the inputs as real numbers, we represent them as ranges of real numbers. Then we might be able to solve a worst-case scenario of each problem. The second way of dealing with noise in the inputs is to get a probability distribution of each input, and trying to maximize the expectation of the utility function.

Appendix A

Appendix

A.1 Defining RAFP using Weighted Constraint Satisfaction Problems

In what follows, we define RAFP using the *Weighted Constraint Satisfaction Problem*. We follow the definitions of *CSP, WCSP* and *VCSP* of [Lar02] and [SFV$^+$95] with slight changes for a unified format.

A.1.1 Constraint Satisfaction Problem

Definition A.1.1 (Constraint Satisfaction Problem)**.** A *constraint satisfaction problem* is a triple $P = \langle V, D, C \rangle$, where:

- $V = \{V_1, ..., V_n\}$ is a set of n variables
- D is a set of values for the variables
- $D_i \subseteq D$ is the set of values for variable V_i, $i \in [1..n]$
- C is a set of constraints, which defines the allowable values that the variables can have simultaneously.

Definition A.1.2 (CSP Tuple)**.** Given a CSP $P = \langle V, D, C \rangle$, An *assignment tuple* t is an ordered set of values assigned to the ordered set of variables $V^t \subseteq V$

Definition A.1.3 (CSP Tuple Consistency)**.** Given a CSP $P = \langle V, D, C \rangle$, A tuple t is *consistent* if it satisfies all constraints whose scope is included in V^t.

Definition A.1.4 (CSP Tuple Global Consistency)**.** Given a CSP, A tuple is *globally consistent* if it can be extended to a consistent complete assignment.

Definition A.1.5 (CSP Solution)**.** Given a CSP, A *solution* is a consistent complete assignment

Definition A.1.6 (Binary Constraint Satisfaction Problem)**.** A *binary constraint satisfaction problem* is a triple $P = \langle V, D, C \rangle$, where:

- $V = \{V_1, ..., V_n\}$ is a set of n variables
- D is a set of values for the variables
- $D_i \subseteq D$ is the set of values for variable V_i, $i \in [1..n]$
- C is a set of unary and binary constraints:
  - An unary constraint $C_i \subseteq D_i$ contains the permitted assignments to V_i
  - A binary constraint $C_{ij} \subseteq D_i \times D_j$ contains the permitted simultaneous assignments to V_i and V_j

A.1.2 Valued Constraint Satisfaction Problem

Definition A.1.7 (Valuation Structure)**.** A *valuation structure* is a triple $S = \langle E, \circledast, \succeq \rangle$, where:

- E is a set of costs, totally ordered by $\succeq$, with maximum element noted $\top$, and minimum element noted $\bot$
- $\circledast$ is a commutative, associative closed binary operator on E, used to combine costs, with the following properties:
  - Identity: $\forall a \in E,\ a \circledast \bot = a$
  - Monotonicity: $\forall a, b, c \in E, (a \succeq b) \implies ((a \circledast c) \succeq (b \circledast c))$
  - Absorbing element: $\forall a \in E, (a \circledast \top) = \top$

Definition A.1.8 (Valued Constraint Satisfaction Problem)**.** A *valued constraint satisfaction problem* is a 5-tuple $P = \langle V, D, C, S, \varphi \rangle$, where:

- $\langle V, D, C \rangle$ is a constraint satisfaction problem
- $S = \langle E, \circledast, \succeq \rangle$ is a valuation structure
- $\varphi : C \to E$ is a valuation function

Definition A.1.9 (VCSP Tuple Valuation)**.** Given a VCSP $P = \langle V, D, C, S, \varphi \rangle$ and a tuple t, the *valuation* of t with respect to the VCSP is defined by:

$$\mathcal{V}(t) = \underset{\substack{c \in C \\ \text{t violates c}}}{\circledast} (\varphi(c)) \tag{A.1}$$

Definition A.1.10 (VCSP Tuple Consistency)**.** Given a VCSP $P = \langle V, D, C, S, \varphi \rangle$, a tuple t is *consistent* if $\mathcal{V}(t) < \top$

Definition A.1.11 (VCSP Solution)**.** Given a VCSP, a *solution* is a consistent tuple

Definition A.1.12 (VCSP Objective)**.** Given a VCSP, the *objective* is $\min_t \mathcal{V}(t)$

A.1.3 Weighted Constraint Satisfaction Problem

Definition A.1.13 (S(k) Valuation Structure)**.** $S(k)$ is the valuation structure $S = \langle [0, \ldots, k], \oplus, \geq \rangle$, where:

- $k \in [1, \ldots, \infty]$
- $\oplus$ is the sum over the valuation structure defined as $a \oplus b = \min\{k, a + b\}$
- $\geq$ is the standard order among naturals

Definition A.1.14 (Weighted Constraint Satisfaction Problem)**.** A *weighted constraint satisfaction problem* is a valued constraint satisfaction problem, with $S(k)$ as a valuation structure, i.e. $P = \langle V, D, C, S(k), \varphi \rangle$

Definition A.1.15 (WCSP Valuation Function)**.** Given a WCSP $P = \langle V, D, C, S(k), \varphi \rangle$ and a tuple t, the valuation function is defined using the $S(k)$ valuation structure:

$$\mathcal{V}(t) = \bigoplus_{\substack{c \in C \\ \text{t violates c}}} (\varphi(c)) \tag{A.2}$$

Definition A.1.16 (Binary Weighted Constraint Satisfaction Problem)**.** A *binary weighted constraint satisfaction problem* is a weighted constraint satisfaction problem $P = \langle V, D, C, S(k), \varphi \rangle$, where $\langle V, D, C \rangle$ is a binary constraint satisfaction problem

Definition A.1.17 (BWCSP Valuation Function)**.** Given a BWCSP $P = \langle V, D, C, S(k), \varphi \rangle$ and a tuple t, The valuation function is defined using the $S(k)$ valuation structure:

$$\mathcal{V}(t) = \sum_{\substack{c_i \in C \\ \text{t violates } c_i}} \varphi(c_i) \oplus \sum_{\substack{c_{ij} \in C \\ \text{t violates } c_{ij}}} \varphi(c_{ij}) \tag{A.3}$$

A.1.4 The Resource Allocation with Forbidden Pairs Problem

Definition A.1.18 (The Resource Allocation with Forbidden Pairs Problem)**.** RAFP is a weighted constraint satisfaction problem $P = \langle V, D, C, S(\infty), \varphi \rangle$ with the following properties:

- $V = \{V_1, ..., V_n\}$ is a set of n variables
- D is a set of l values for the variables. Each value $d \in D$ is a 2-tuple $\langle a, r \rangle$, where:
  - $a \in [1..l]$ is the tactic in use
  - $r \in [1..m]$ is the resource in use
- $D_i \subseteq D$ is the set of values for variable V_i, $i \in [1..n]$
- C is a set of constraints, φ is a valuation function where:

– An unary constraint $C_i \subseteq D_i$ contains costly $(0 < \varphi(c_i) < \infty)$ assignments to V_i

– A binary constraint $C_{ij} \subseteq D_i \times D_j$ contains costly $(0 < \varphi(c_{ij}) < \infty)$ simultaneous assignments to V_i and V_j

– A *resource constraint* C_R is a constraint with a non-constant arity defining forbidden resource consumption:

$$(\langle a_1, r_1\rangle, \ldots, \langle a_s, r_s\rangle) \in C_R \implies \exists j \in [1..m] \, |\{r_i | r_i = j\}| > B_j$$

where $B_j \in \mathbb{N}$ for $j \in [1..m]$ are bounds on the resources

– Valuation function

$$\varphi(c) = \begin{cases} 0 < g < \infty & c \in C_i \cup C_{ij} \\ 0 < h < \infty & c \in C_R \\ 0 & \text{otherwise} \end{cases}$$

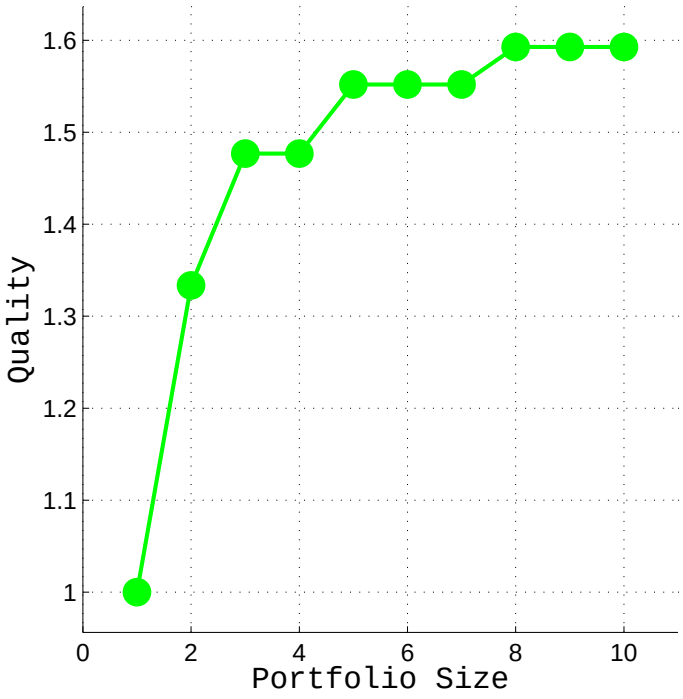
Definition A.1.19 (RAFP Valuation)**.** Given a RAFP $P = \langle V, D, C, S(k), \varphi\rangle$ and a tuple t, The valuation is defined using the $S(k)$ valuation structure:

$$\mathcal{V}(t) = \sum_{\substack{c_i \in C \\ \text{t violates } c_i}} \varphi(c_i) \oplus \sum_{\substack{c_{ij} \in C \\ \text{t violates } c_{ij}}} \varphi(c_{ij}) \oplus \sum_{\substack{c_R \in C \\ \text{t violates } c_R}} \varphi(c_R) \tag{A.4}$$

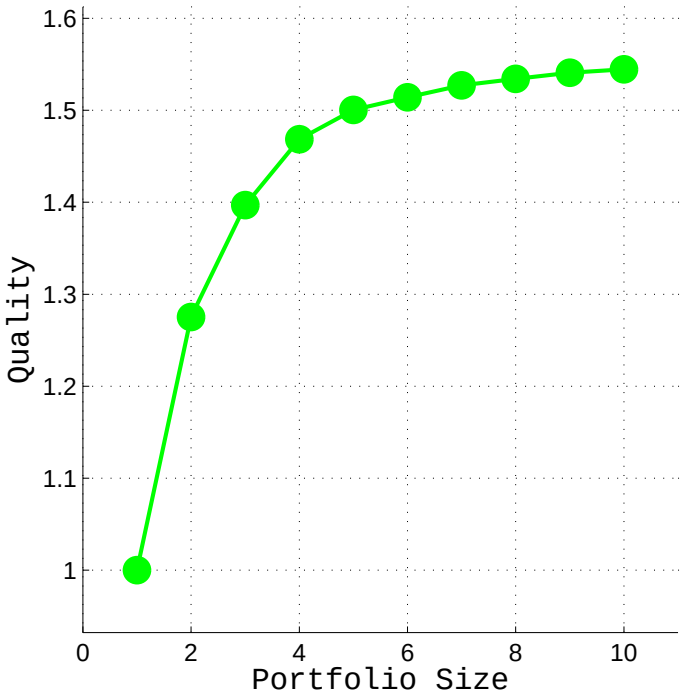
A.2 Portfolios Construction – Random Matrices

In what follows, we describe an empirical evaluation that we conducted using random matrices as an input for the optimal construction of max-sum portfolios.

We created two 10-row matrices with uniformly distributed random numbers in [0..1]. We used these matrices as the input to the modeling program we described in section 8.1, with $k \in [1..10]$. This setting simulates a quality matrix of 10 algorithms, with a low correlation. We solved the SMT modeling using $Z3$. The results are shown in figure A.1. In A.1a we simulated 15 inputs (matrix columns) and we can see several examples that a non-increasing portfolio does not imply convergence (e.g. points 3,4 and 5,6,7). In A.1b we simulated 40 inputs and we can see a smoother graph, with a slowing rate of increase. Notice that the quality axis is not comparable to the one in figure 8.1, where the results are normalized to the 1-portfolio quality.



(a) 10 Algorithms, 15 Inputs



(b) 10 Algorithms, 40 Inputs
Figure A.1: Max-Sum Portfolios, Random Matrices

Bibliography

[Ack] Wilhelm Ackermann. Solvable cases of the decision problem.

[AST09] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 142–157. Springer, 2009.

[BBNP04] Edmund Burke, Yuri Bykov, James Newall, and Sanja Petrovic. A time-predefined local search approach to exam timetabling problems. *IIE Transactions on Operations Engineering*, 36:1–19, 2004.

[BCD+11] Clark Barrett, Christopher Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Computer aided verification*, pages 171–177. Springer, 2011.

[BDL98] Clark W Barrett, David L Dill, and Jeremy R Levitt. A decision procedure for bit-vector arithmetic. In *Proceedings of the 35th annual Design Automation Conference*, pages 522–527. ACM, 1998.

[BPW+12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Bro58] Samuel H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6:244–251, 1958.

[Bro11] Jason Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[BSS15] Tomas Balyo, Peter Sanders, and Carsten Sinz. Hordesat: a massively parallel portfolio sat solver. *arXiv preprint arXiv:1505.03340*, 2015.

[CDH+09] Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. *VCC: A Practical System for Verifying Concurrent C*, pages 23–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Cla99] Jens Clausen. Branch and bound algorithms-principles and examples. *Dept. Comput. Sci., Univ. Copenhagen, [Online]*, 1999.

[CZ06] Sharlee Climer and Weixiong Zhang. Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence*, 170(8-9):714–738, 2006.

[DB88] T.L Dean and M.S. Boddy. An analysis of time-dependent planning. *AAAI*, 17:49–54, 1988.

[DDM06] Bruno Dutertre and Leonardo De Moura. A fast linear-arithmetic solver for dpll (t). In *International Conference on Computer Aided Verification*, pages 81–94. Springer, 2006.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. *Z3: An Efficient SMT Solver*, pages 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[Dut14] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[FM93] Stephanie Forrest and Melanie Mitchell. Relative Building-Block Fitness and the Building-Block Hypothesis. *Foundations of Genetic Algorithms*, 2:109–126, 1993.

[GJ79] M. R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of np-completeness. 1979.

[Glo86] Fred Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[GS01] Carla P Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[HBHH07] Frank Hutter, Domagoj Babic, Holger H. Hoos, and Alan J. Hu. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer Aided Design, 2007. FMCAD '07*, pages 27 –34, nov. 2007.

[HHLB10] Frank Hutter, HolgerH. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer Berlin Heidelberg, 2010.

[HHLB11] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[HHLBS09] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[HLBSS12] Holger Hoos, Kevin Leyton-Brown, Torsten Schaub, and Marius Schneider. Algorithm configuration for portfolio-based parallel sat-solving. In *Workshop on Combining Constraint Solving with Mining and Learning*, 2012.

[HLH97] Bernardo A Huberman, Rajan M Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[HS04] Holger H. Hoos and Thomas Stutzle. Stochastic Local Search: Foundations and Applications. *Morgan Kaufmann*, 2004.

[JV83] Scott Kirkpatrick , C. Daniel Gelatt Jr. and Mario P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[KBS10] D. Kroening, R.E. Bryant, and O. Strichman. *Decision Procedures: An Algorithmic Point of View.* Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010.

[Lar02] Javier Larrosa. Node and arc consistency in weighted csp. In *Proceedings of the National Conference on Artificial Intelligence*, pages 48–53, 01 2002.

[Lei10] K Rustan M Leino. Dafny: An automatic program verifier for functional correctness. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 348–370. Springer, 2010.

[LIDLC+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Lou03] Samir Loudni. Solving constraint optimization problems in anytime contexts. *IJCAI*, pages 251–256, 2003.

[McC62] John McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, volume 62, pages 21–28, 1962.

[MH97] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[Mit98] Melanie Mitchell. *An introduction to genetic algorithms*. The MIT Press, 1998.

[MS12] Yuri Malitsky and Meinolf Sellmann. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 244–259, 2012.

[OD12] Lars Otten and Rina Dechter. Anytime and/or depth-first search for combinatorial optimization. *AI Communications*, 25(3):211–227, 2012.

[PZ06] Marek Petrik and Shlomo Zilberstein. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence*, 48(1):85–106, 2006.

[QYZ15] Chao Qian, Yang Yu, and Zhi-Hua Zhou. Subset selection by pareto optimization. *Advances in Neural Information Processing Systems*, pages 1774–1782, 2015.

[RK04] Reuven Rubinstein and Dirk Kroese. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning. *Springer-Verlag, New York*, 2004.

[RLIS13] Andreea Radulescu, Manuel López-Ibánez, and Thomas Stützle. Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 825–840. Springer, 2013.

[SBLI12] Thomas Stützle, Mauro Birattari, and Manuel López-Ibáñez. *Anytime Local Search for Multi-Objective Combinatorial Optimization: Design, Analysis and Automatic Configuration.* PhD thesis, Citeseer, 2012.

[SFV+95] Thomas Schiex, Helene Fargier, Gerard Verfaillie, et al. Valued constraint satisfaction problems: Hard and easy problems. *IJCAI (1)*, 95:631–639, 1995.

[Sin05] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. *CP*, 3709:827–831, 2005.

[SS12] Holger Hoos , Kevin Leyton-Brown , Torsten Schaub and Marius Schneider. Algorithm Configuration for Portfolio-based Parallel SAT-Solving. *CoCoMile*, 2012.

[VFG+11] M. Vallati, C. Fawcett, A. Gerevini, H.H. Hoos, and A. Saetti. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proceedings of the Eighth RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2011.

[WC00] Benjamin W Wah and Yi Xin Chen. Optimal anytime constrained simulated annealing for constrained global optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 425–440. Springer, 2000.

[WPF+10] Yi Wei, Yu Pei, Carlo A Furia, Lucas S Silva, Stefan Buchholz, Bertrand Meyer, and Andreas Zeller. Automated fixing of programs with contracts. In *Proceedings of the 19th international symposium on Software testing and analysis*, pages 61–72. ACM, 2010.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[XHHLB12] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. *Theory and Applications of Satisfiability Testing–SAT 2012*, pages 228–241, 2012.

[Yan10] Xin-She Yang. Nature-Inspired Metaheuristic Algorithms, Second Edition. *Luniver Press*, pages 12–13, 2010.

[Zha00] Weixiong Zhang. Depth-first branch-and-bound versus local search: A case study. In *AAAI/IAAI*, pages 930–935, 2000.

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996.

הממצאים מראים כי עבור מדד סכום האיכויות, ביצועי השיטה החמדנית כמעט זהים לביצועים האופטימליים, עבור שתי הקבוצות של הבעיות. ביצועי שיטת בחירת k האלגוריתמים הטובים ביותר נחותים לעומת שתי השיטות האחרות. בקבוצת הבעיות הכלליות אנו מקבלים שיפור איכות מקסימלי של 2 אחוזים לעומת האלגוריתם הטוב ביותר, במערך שבו 17 אלגוריתמים מתוך ה־24. בקבוצת הבעיות הקשות אנו מקבלים שיפור איכות של 11 אחוזים לעומת האלגוריתם הטוב ביותר, במערך שבו 15 אלגוריתמים מתוך ה־24. עבור מדד המרחק הגדול ביותר מהאופטימום, נראה שהשיטה החמדנית אינה מגיעה לביצועי השיטה האופטימלית, והשיטה של k האלגוריתמים הטובים ביותר גרועה ממנה. עבור קבוצת הבעיות הכלליות המערך האופטימלי הגדול ביותר בו עדיין יש פער מול האופטימום משפר את האיכות פי 2 מיליון לעומת האלגוריתם היחיד הטוב ביותר. במערכים גדולים מכך המרחק לאופטימום הוא 0, על־פי ההגדרה. עבור קבוצת הבעיות הקשות, המערך האופטימלי הגדול ביותר בו עדיין יש פער מול האופטימום משפר את האיכות פי 2000 לעומת האלגוריתם היחיד הטוב ביותר. שתי התוצאות האחרונות הן משמעותיות ומדגישות את המחיר האפשרי של שימוש בפחות ליבות מהנדרש לצורך פיתרון בעיה.

בניית מערך מקבילי של אלגוריתמים

בחלק זה אנו דנים בבעיה של בניית מערך מקבילי של אלגוריתמים, כדי לנצל משאבי מיחשוב מקבילי ולשפר עוד יותר את הביצועים בזמן אמת באופן כללי. בהנתן n אלגוריתמים, מספר k<n של ליבות חישוביות, ותוצאות ריצות האלגוריתמים על קלטים שונים, אנו מעוניינים לבחור קבוצה של k אלגוריתמים מתוך n שביצועיהם יחד כמערך מקבילי יהיו טובים ביותר. אנו מראים כיצד ניתן למדל את הבעיה באמצעות Satisfiability Modulo Theories (SMT), הרחבה עשירה של בעיית הספיקות SAT. לצורך המידול אנו משתמשים בתיאוריה של Quantifier-Free Linear Real Arithmetic (QF_LRA) המאפשרת נוסחאות בוליאניות של פרדיקטים הבנויים מאי-שיוויונים מעל משתנים ממשיים. אנו מציעים שני מדדים טבעיים לביצועי מערך אלגוריתמים. המדד הראשון לטיב מערך אלגוריתמים הוא סכום איכויותיו על הקלטים, אותו יש להביא למקסימום. אנו מכנים בעיה זו K-Algorithms Max-Sum. המדד השני הוא המקסימום על-פני הקלטים, של המרחק בין איכות המערך לאיכות האלגוריתם הטוב ביותר. מדד זה יש להביא למינימום. אנו מכנים בעיה זו K-Algorithms Min-Max-Gap. בעיות דומות אותן נרצה לפתור הן מציאת ה-k המינימלי עבורו ניתן לקבל ביצועים אופטימליים, עבור שני המדדים שהזכרנו. פיתרון בעיות אלה יאפשר לנו להבין כמה ליבות נדרשות למיצוי היכולת מקבוצת אלגוריתמים נתונה, או קבלת מידע שיש לפתח עוד אלגוריתמים כדי לנצל כמות ליבות נתונה.

ניתוח ביצועים השוואתי של מערך האלגוריתמים המקבילי

אנו מממשים תכנית לקידוד SMT, שהקלט שלה הוא מטריצת איכויות של כל אלגוריתם על כל קלט, ומספר k, והפלט שלה הם הקידודים של שתי הבעיות שהזכרנו בפסקה הקודמת. לצורך פתרון מודלי ה-SMT אנו משתמשים בפותרן Z3. אנו מנתחים מערכי אלגוריתמים שאותם אנו בונים מתוצאות 24 האלגוריתמים מול 500 הקלטים בהם דנו קודם. לצורך השוואה מול מערך האלגוריתמים האופטימלי אנו מממשים שתי שיטות פשוטות נוספות לבניית מערך אלגוריתמים. הראשונה היא שיטה חמדנית בה אנו מתחילים ממערך אלגוריתמים ריק ובוחרים בכל איטרציה את האלגוריתם שישלים בצורה הטובה ביותר את מערך האלגוריתמים הקיים, למשך k איטרציות. בשיטה השניה אנו ממיינים את האלגוריתמים על-פי טיבם ובוחרים את k האלגוריתמים הטובים ביותר. אנו בונים מערכי אלגוריתמים אופטימליים, חמדניים ו-k הראשונים עבור קבוצת הבעיות הכלליות ועבור קבוצת הבעיות הקשות יותר שדנו בהן קודם.

אלגוריתמים מתאימים לפיתרון הבעיה

אנו מגדירים סכמה אלגוריתמית אותה אנו מכנים Fixed-Time Search כסכמת החישוב המתאימה לדרישות ה-anytime שלנו. ראשית אנו כוללים בסכמה מספר אלגוריתמי חיפוש מקומי ידועים כגון Stochastic Hill-Climbing, Simulated Annealing, Tabu Search ומתאימים אותם לבעיה. בנוסף, אנו מממשים בסכמה חיפוש חמדני פשוט לבעיה, ואת חיפוש המונטה קרלו The Cross-Entropy Method. אנו מממשים מספר גרסאות כלאיים, שהן חיפוש חמדני שלאחריו מפעילים את אחד האלגוריתמים שהוזכרו. כנקודות השוואה אנו משתמשים באלגוריתמים האקראיים הנאיביים Random Search ו-Random Walk. אנו מממשים את האלגוריתמים ומנתחים את ביצועיהם עבור זמן ריצה של שניה אחת, כממוצע על 50 קלטים אקראיים שיצרנו עבור RAFP. פרופיל האיכות של האלגוריתמים כפונקציה של הזמן מראה שאלגוריתמי הכלאיים הם בעלי איכות דומה וטובה לאחר שניה. לאלגוריתמים האחרים, שאינם מבוססים על ריצה מקדימה של האלגוריתם החמדני, יש ביצועים שונים ופחותים מביצועי אלגוריתמי הכלאיים לאחר שנייה. באופן כללי, לאורך זמן ריצה של שניה אחת משתנה דירוג האלגוריתמים כך שההמלצה באיזה אלגוריתם בודד להשתמש תלויה בזמן הריצה המדויק שנבחר בתוך המרווח של שניה אחת.

כיוונון פרמטרים אוטומטי וניתוח ביצועים

כיוונון פרמטרים של אלגוריתמים הוא גורם משמעותי בשיפור ביצועיהם, והכרחי לצורך השוואה הוגנת ביניהם. כיוונון פרמטרים אוטומטי עמיד יותר כנגד הטיות שאינן רצויות. אנו משתמשים בכלי אוטומטי לכיוונון פרמטרים מן השורה הראשונה שנקרא ParamILS לצורך כיוונון פרמטרי האלגוריתמים שמימשנו. ראשית אנו מכוונים את פרמטרי האלגוריתמים לביצועים הטובים ביותר לאחר שניה אחת, ורואים כיצד ביצועי אלגוריתמים שאינם מבוססים על פיתרון חמדני משתפרים משמעותית וחלקם עולים על ביצועי אלגוריתמי הכלאיים. לאחר מכן אנו מבצעים תהליך כיוונון הפרמטרים באמצעות 500 בעיות אקראיות מסוג RAFP למשך 1500 שניות וזמן ריצת אלגוריתם בודד של עשירית השניה. אנו מראים כי איכות הפיתרונות של האלגוריתמים לאחר כיוונון הפרמטרים משתפרת בעד 35 אחוזים לעומת ביצועיהם עם הערכים ההתחלתיים של הפרמטרים. כמו כן, הדירוג ההשוואתי של האלגוריתמים משתנה במהלך תהליך כיוונון הפרמטרים, מה שמדגיש עוד יותר את חשיבות התהליך. אנו מבצעים כיוונון פרמטרים וניתוח תוצאות בנפרד גם עבור 500 בעיות אקראיות קשות יותר. הבעיות הקשות נבנו על-ידי בחירת הסתברות של 0.5 לכך שכל זוג ערכים למשתנים יהיה חוקי. ניתוח ביצועי האלגוריתמים על הבעיות הקשות יותר מראה דרוג אלגוריתמים שונה במעט וניכר שהביצועים של כלל האלגוריתמים טובים פחות על הבעיות הקשות. לאחר כיוונון הפרמטרים אנו מבצעים הערכה של ביצועי האלגוריתמים בתצורתם הסופית על-ידי הרצתם מול 500 בעיות אקראיות אחרות מאשר אלה ששימשו לצורך כיוונון הפרמטרים. אנו מראים שעל-ידי כיוונון מונחה seed שונה, ניתן ליצור כמה גרסאות של כל אלגוריתם שהוא בר-כיוונון, ואנו יוצרים שתי גרסאות לכל אלגוריתם כזה. כך מתקבלים 24 אלגוריתמים שונים לפיתרון RAFP.

תקציר

הקדמה

מערכות זמן־אמת נדרשות לעיתים לפתור בעיות קשות חישובית בזמן קבוע וקצר מאד. לצורך קיום דרישה שאפתנית זו, לא נוכל להעזר באלגוריתמים הפותרים את הבעיה אופטימלית כיוון שאלו דורשים זמן ריצה אקספוננציאלי בגודל הקלט. אלגוריתמי קירוב מסוג PTAS מספקים פיתרון בעל חסם עליון על מרחקו מהפיתרון האופטימלי. זאת הם עושים בזמן פולינומיאלי בגודל הקלט, ולכן גם הם עלולים לחרוג מזמן הריצה הקבוע העומד לרשותינו. אנו מתמקדים בבעיות קשות חישובית מסוג NP־optimization, שלהן ניתן לבצע רדוקציה לבעיית Constraints Optimization Problem וכך נוכל לדון באילוצים אותן הן מכילות. האילוצים בבעיות אלה נחלקים לאילוצים קשים, אותם יש לקיים ללא פשרות, ואילוצים רכים שאת ערכם יש למקסם. בנוסף תתכן לבעיה פונקציית מטרה אותה יש למקסם, אך היא שקולה לאילוצים רכים ולכן לא נדון בה עוד. בהנתן בעיה, משקלים על האילוצים וזמן ריצה נדרש, אנו מגדירים את 'גרסת זמן־קבוע' של הבעיה כבעיה שבה האילוצים הקשים הופכים לאילוצים רכים עם המשקלים הנתונים, ואותה יש לפתור תוך זמן הריצה הנתון. כדי לפתור את גרסת הזמן הקבוע של בעיה קשה חישובית ניתן להשתמש באלגוריתמי Anytime המחזירים סדרה של פתרונות באיכות עולה, כך שלפחות אחד מהם יוחזר תוך זמן הריצה הנתון.

בעיית אופטימיזציה מוחשית

לצורך לימוד הנושא אנו מגדירים בעיה מוחשית שאנו קוראים לה RAFP. בעיה זו עוסקת בבחירת ערכים למשתנים בדידים, כך שכל ערך מייצג דרך־פעולה ספציפית. לכל משתנה מוגדר מספר מסוים של ערכים חוקיים, ולכל זוג משתנים קיימים אילוצים על הערכים שניתן לשים בהם בו־זמנית, ונקראים אילוצים בינאריים. בנוסף, כל השמה של ערך למשתנה גוררת הקצאת משאב מתכלה מתוך אוסף משאבים שסוגיהם וכמויותיהם נתונים, ואין לחרוג מהם. אנו מדגימים ייצוג של שתי בעיות אופטימיזציה בדידה באמצעות RAFP: קבלת החלטות לשיבוץ שירותי חירום רפואיים וקבלת החלטות למסחר אוטומטי. אנו מוכיחים כי גרסת ההכרעה של הבעיה שהגדרנו היא NP־Complete על־ידי רדוקציה פולינומיאלית אליה מגרסה של בעיית צביעה, הידועה כקשה חישובית.

המחקר בוצע בהנחייתו של פרופסור עופר שטרייכמן, בפקולטה להנדסת תעשייה וניהול.

תודות

אני רוצה להודות למנחה שלי, שהראה לי את הדרך הנכונה. תודה להוריי, שעודדו אותי ללמוד. תודה לילדיי, אשתי והוריה שאפשרו את השלמת התזה, בסבלנותם.

פיתרון בזמן אמת
של בעיות אופטימיזציה דיסקרטית

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת ניהול מידע

יאיר נוף

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
אדר התשע"ח חיפה מרץ 2018

פיתרון בזמן אמת
של בעיות אופטימיזציה דיסקרטית

יאיר נוף