AN IMPROVED RESTART STRATEGY FOR RANDOMIZED BACKTRACK SEARCH

by

Venkata Praveen Reddy Guddeti

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Berthe Y. Choueiry

Lincoln, Nebraska

December, 2004

AN IMPROVED RESTART STRATEGY FOR RANDOMIZED BACKTRACK SEARCH

Venkata Praveen Reddy Guddeti, M.S.

University of Nebraska, 2004

Advisor: Berthe Y. Choueiry

When solving combinatorial problems with backtrack search, *randomization* of the path explored by search and frequent *restarts* of the search mechanism have been proposed as an effective way to allow the exploration of wider areas of the search space than otherwise possible. In these strategies, the choice of a *cutoff value*, a point after which the search is restarted, remains an open issue. Previous restart strategies rely, for computing this cutoff value, either on the availability of an overall profile of the cost of search [Gomes *et al.*, 1998], or on a predetermined restart schedule. An example of the latter is the Randomization and Geometric Restart (RGR) strategy proposed by Walsh [1999], which computes the cutoff value as a function of the number of variables in the problem and a constant parameter given as input. We propose Randomization and Dynamic Geometric Restart (RDGR), an improved restart strategy of Walsh's RGR. Unlike previous strategies, which have fixed restart schedules, our technique *dynamically* adapts the value of the cutoff parameter to the results of the search process.

We evaluate empirically the behavior of our technique and compare its performance to that of other search techniques. We use the cumulative distribution of the solutions, and consider different run-time durations, values of the constant parameter used to compute the cutoff value, and problem types (i.e., a real-world resource allocation problem and randomly-generated binary constraint satisfaction problems). We show that distinguishing between solvable and over-constrained problem instances yields new insights on the rela-

tive performance of the search techniques tested. On over-constrained problem instances and random instances at the phase transition, RDGR statistically outperforms other search techniques. While on under-constrained problems, RDGR is second to a multi-agent-based search [Liu *et al.*, 2002; Zou, 2003]. We propose to use this characterization as a basis for building new strategies of cooperative, hybrid search.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Berthe Y. Choueiry, for her support, guidance, and advice during my work on this thesis. I am indebted to my committee members, Dr. F. Choobineh (Department of Industrial & Management Systems Engineering at UNL), Dr. S. Dunbar (Department of Mathematics at UNL), and Dr. W. Srisa-an (Department of Computer Science & Engineering at UNL), for their enlightening suggestions and comments, which helped me improve the content and presentation of this thesis. I would like to thank Dr. K. Xu (Beijing University of Aeronautics and Astronautics), and Dr. J.I. van Hemert and Dr. B.G.W. Craenen (Napier University) for insightful email discussions on generating random CSP instances and for the use of the RandomCSP software of Dr. J.I. van Hemert. Also, I would like to thank Mr. B. Danner (Department of Statistics), Mr. M. Abdoli (Department of Industrial & Management Systems Engineering at UNL), and Dr. H. Hoos (University of British Columbia) for help with the statistical analysis. I am very grateful for the opportunity to be a member of the Constraint Systems Laboratory for the past two years. I enjoyed pursuing my research interests and obtaining a wealth of invaluable experiences in many aspects of my academic life. I also thank the members of the lab, in particular Hui Zou, Anagh Lal, Ryan Lim, and Daniel Buettner for their support and for our many interesting discussions.

Finally, I gratefully acknowledge the constant support to my family and friends, in particular my late grandfather G. Subba Reddy and my parents G. RamaLakshmi and G. Lokanatha Reddy, who sent me on my way and provided a stable and stimulating environment for my personal and intellectual development.

This research was supported by NSF grants #EPS-0091900 and CAREER Award #0133568. The experiments were conducted utilizing the Research Computing Facility of UNL.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We propose an improved restart strategy for randomized backtrack search. Randomized backtrack search with restarts borrow the backtracking feature of systematic backtrack search and the stochastic characteristic of local search. Our research was motivated by a real-world application, which is the assignment of Graduate Teaching Assistants (GTA) to academic tasks. In practice, this problem is large, tight, and sometimes over-constrained. Since 2001, various members of the Constraint Systems Laboratory have designed and implemented a set of interactive and automated search techniques to solve this problem. We compare the performance of our new strategy to that of the various automated solvers developed in our group. The long-term goal of this project is to provide a robust portfolio of search algorithms to solve complex decision-making problems. This chapter presents the motivations for our work, related works, the questions addressed, and our contributions.

1.1 Motivations

A great deal of theoretical and empirical research has focused on developing and improving the performance of algorithms for solving Constraint Satisfaction Problems (CSP). Because CSPs are in general **NP**-complete, search remains a key mechanism for solving them.

Search algorithms for solving CSPs are usually classified into two main categories: systematic backtrack search and iterative-improvement search. Generally, systematic search techniques have been almost always deterministic in nature and iterative improvement search techniques are stochastic. Systematic search, as the name suggests, operates by exhaustively examining the solution space, which makes it complete and sound. In contrast, iterative-improvement search starts from a random solution, which may or may not be consistent, and tries to reach better solutions by visiting neighboring solutions (thus, the name local search). This makes it incomplete.

On large problems the performance of systematic search degrades with the size of the problem (thrashing), while the performance of iterative-improvement search is impaired by local optima and livelocks. Variable and value ordering heuristics and techniques like backjumping and backmarking are employed to improve the performance of systematic search, but cannot totally eliminate thrashing. Moreover, the memory requirements for the data structures necessary for backjumping and backmarking may be a cause of concern with increasing problem size. Iterative-improvement search can be improved with heuristics (min-conflict [Minton *et al.*, 1992]), random restarts, and random walk to avoid and recover from local optima. However, its inherent incompleteness remains a major concern. This situation motivated us to explore randomized backtrack search with restarts as designed by Gomes et al. [1998]. One important feature of this search is the restart strategy. In this thesis we propose a restart strategy that improves on the one proposed by Walsh [1999] and compare the performance of the resulting search with that of Walsh another and other search techniques implemented in the Constraint Systems Laboratory.

1.2 Graduate Teaching Assistants Assignment Problem

The Graduate Teaching Assistants Assignment Problem (GTAAP) is a critical and arduous task that our department's, Computer Science and Engineering (CSE), administration has to drudge through every semester. Given a set of courses, a set of graduate teaching assistants (GTAs), and a set of constraints, the goal is to find a consistent and satisfactory assignment of GTAs to courses. Glaubius and Choueiry modeled the GTAAP as a Constraint Satisfaction Problem (CSP) [2001; 2002a; 2002b]. The constraints specify allowable assignments such as the availability and proficiency of a graduate student for being a teaching assistant. In a consistent solution there are no broken constraints. And, satisfactory solution attempts to optimize the quality of a solution in terms of the preferences expressed by the GTAs.

In the CSP model of the GTAAP, the courses are modeled as variables and the GTAs as the domain values of these variables. By focusing our investigations on this particular real-world application, we have been able to identify and compare the advantages and shortcomings of the various search strategies that have implemented to solve this problem. Such an insight is unlikely to be gained from testing toy problems, and surely difficult from testing randomly generated problems. We show that the identified behaviors apply beyond our application, and hold on randomly generated binary CSPs.

1.3 Related works

There are three main works on randomized backtrack search with restarts:

1. *Randomization and rapid restarts* (RRR): Gomes et al. proposed the RRR restart strategy [1998]. RRR employs a fixed optimal cutoff value. Search is restarted on reaching the cutoff value. The estimation of the optimal cutoff value requires a priori knowledge of the cost distribution of the problem instance, which is not known in

most settings and must be determined by trial-and-error. This is clearly not practical in general.

2. *Randomization and geometric restarts* (RGR): In the absence of an optimal cutoff value, the RGR restart strategy of Walsh can be used [1999]. Unlike RRR, the cutoff value in RGR is not fixed, but is geometrically increasing. On reaching the cutoff value, search is restarted and the cutoff value for the next restart is geometrically increased by a constant factor r regardless of the progress of search. Like RRR, RGR is static in the sense that it does not take into account the outcome of the search during a run to compute the cutoff value for the following run. Note that the cutoff value in RGR is strictly monotonically increasing and, in theory, the resulting search mechanism is complete[1].

3. *Bayesian approach*: Kautz et al. introduced an optimal policy for dynamic restarts [2002]. They employ Bayesian methods to build predictive models of the run-time distribution. Utilizing this model, their restart strategy considers the predictions about run time to choose the cutoff value for restarts. This approach is based on the use of machine learning techniques, and, unlike the restart strategies, increases the complexity of the implementation and deployment.

1.4 Questions addressed

In this thesis, we address the following questions:

1. How effective are variable and value orderings in backtrack search for solving the GTAAP?

   Answer: Research has shown that variable and value orderings are effective techniques for improving the performance of backtrack search. We show that on GTAAP,

[1] Probabilistically approximately complete Las Vegas algorithm.

while dynamic selection of variables consistently outperforms static selection, variable orderings that utilize the least domain (LD) and domain-to-degree (DD) ratio heuristics result in similar performances. Further, the various value-ordering heuristics we tested do not result in qualitative improvements.

2. How serious is thrashing on large problem instances?

   Answer: We show in Section 3.2 that thrashing is serious enough to warrant the study of restart strategies.

3. What are the best values for the ratio used to increase the cutoff value in geometric-restart strategies?

   Answer: We confirm that for RGR, a value of r=1.1, as devised by Walsh, is the best value of the ratio, and show that this holds for both the GTAAP data and for randomly generated problems. For RDGR, r=1.1 yields best results, however on randomly generate problems, a higher value (i.e., r=2.0) is a better choice. This discrepancy of the value of r on random problems can be explained by the fact that the cutoff value increases more quickly in RGR than in RDGR.

4. What are the characteristics of the different search techniques implemented with respect to the different problem types?

   Answer: We show that distinguishing between solvable and over-constrained problem instances yields new insights on the relative performance of the search techniques tested. On over-constrained problem instances and problem instances at the phase transition, RDGR statistically outperforms other search techniques. While on under-constrained problems, RDGR is second to the multi-agent-based search known as ERA [Liu *et al.*, 2002; Zou, 2003].

1.5 Contributions

In this thesis, we propose an improved restart strategy for randomized backtrack search, which we denote RDGR. We compare its performance to four other search mechanisms (denoted BT, LS, ERA, and RGR) in order to study and characterize their behavior. We conduct our investigations in the context of a real-world application, which is the assignment of Graduate Teaching Assistants (GTA) to academic tasks. We also conduct tests on randomly generated binary CSPs. Our main contributions can be summarized as follows:

1. Testing of various ordering heuristics in BT for solving GTAAP.

2. Proposition of a new improved restart strategy (RDGR) for randomized backtrack search.

3. Empirical evaluation of RDGR by comparing its performance with that of BT, LS, ERA, and RGR, both on GTAAP data and randomly generated instances.

Finally, we identify two directions for future research, namely the development of progress-aware restart strategy and cooperative, hybrid search techniques.

1.6 Outline of the thesis

This thesis is structured as follows. Chapter 2 briefly reviews the Constraint Satisfaction Problem, the Graduate Teaching Assistants Assignment Problem, and the various search techniques developed for solving GTAAP. Chapter 3 explains the empirical evaluation of the ordering heuristics in backtrack search in the context of GTAAP. Chapter 4 presents RDGR, our proposed dynamic restart strategy for randomized backtrack search, and its empirical evaluation. Finally, Chapter 5 concludes the thesis and provides directions for future research.

Appendix A lists the results of running all the currently available search techniques on the collected GTAAP data.

Chapter 2

Background

This chapter draws the background of our work. After a brief introduction to the Constraint Satisfaction Problem (CSP), we introduce a real-world application, the Graduate Teaching Assistants assignment problem (GTAAP), which is at the motivation for our investigations. We briefly review how it was modeled by Glaubius and Choueiry as a CSP and solved using systematic backtrack search [2001; 2002a; 2002b]. Finally, we review the search techniques developed for solving GTAAP.

2.1 Constraint Satisfaction Problem (CSP)

Constraints are ubiquitous in everyday life. A constraint is a relation that restricts the set of allowable combinations of values among a set of variables. Some examples of everyday constraints are qualifications and requirements for a job or for college admission, speed limits for driving, and monetary constraints for buying a new car. Such problems can be modeled as Constraint Satisfaction Problems (CSPs), where a set of decisions, each with a set of options, must be made under a set of constraints restricting the allowable combinations of options for decisions. Constraint Processing, a sub-area of Artificial Intelligence, is a set of set of representation and processing techniques concerned with solving such prob-

lems. CSPs can be decision or optimization problems (in which case an objective function must be specified). They are useful for modeling and solving many complex problems such as scheduling, resource allocation, planning, temporal reasoning [Dechter *et al.*, 1991], and constraint databases [Revesz, 2002]. More formally, a CSP is often defined as follows:

> A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ the set of their respective domains, and $\mathcal{C}$ is a set of constraints that restricts the acceptable combinations of values for variables. Solving a CSP requires assigning a value to each variable such that all constraints are simultaneously satisfied, which is in general **NP**-complete.

The CSP framework provides a common platform to researchers for developing application-independent solvers and study the behavior of different search techniques, which have yielded important industrial benefits A CSP can be characterized by a number of parameters used to describe and compare problem instances. Below we list the main features:

1. *Number of variables*: number of assignments to be made.

2. *Domain size*: size of the largest domain.

3. *Tightness of a given constraint*: is defined as the ratio of number of tuples disallowed by the constraint over that of all possible tuples, which is the size of the cross Cartesian product of the domains of the variables in the scope of the constraint.

4. *Proportion of constraints*: is the ratio of number of constraints in the CSP to the total number of possible constraints.

5. *Problem size*: number of all possible combinations, calculated as $\Pi_{v \in \mathcal{V}} |\mathcal{D}_v|$. As the number of the variables increases, the number of possible combinations increases exponentially.

6. *Constraint arity*: the number of variables in the scope of a constraint. Increasing the constraint arity increases the complexity of checking the consistency of the constraint.

7. *Number of solutions sought*: depending on the application, we may need to find one solution, all solutions (entire search space needs to be explored), or an optimal solution.

Many real-life applications are over-constrained. Over-constrained problems have no complete assignment of values to variables such as all constraints are satisfied. The following ways are used to handle over-constrained problems [Zou, 2003]:

1. Relax the problem by removing some constraints. Finding the smallest set of constraints that need to be removed is thought to be **NP**-hard.

2. Express preferences between constraints or allocate weights to allowed tuples of a constraint (e.g., preference-based CSP [Junker, 2002] and soft constraints [Bistarelli *et al.*, 1995]).

3. Maximize the number of satisfied constraints (e.g., the MAX-CSP framework of [Freuder, 1989; Freuder and Wallace, 1992]).

4. Accept partial, consistent solutions (i.e., do not cover all variables) that maximize the number of assigned variables.

In GTAAP, the objective is to provide support to as many courses as possible while satisfying all the constraints. Thus, we are interested in finding the *maximal consistent partial solutions*. For the rest of this thesis, a solution may be partial, but must always be consistent.

2.2 Graduate Teaching Assistants Assignment Problem

Glaubius and Choueiry define GTAAP as follows: “Given a set of graduate teaching assistants (GTAs), a set of courses, and a set of constraints that specify allowable assignments of GTAs to courses, the goal is to find a consistent and satisfactory assignment” [2001; 2002a; 2002b]. In a consistent solution there are no broken constraints. And, in a satisfactory solution maximizes the preferences expressed by the GTAs. Hard constraints (e.g., a GTA’s competence, availability, and employment capacity) must be met, and GTA’s preferences for courses (expressed on a scale from 1 to 5) must be maximized. It is a critical and time-consuming process to be done by our department and likely other institutions across the world. Typically, every semester, the department has about 70 different academic tasks and can hire between 25 and 40 GTAs. Instances of this problem, collected since Spring 2001, are consistently tight and often over-constrained. However, this is not known a priori. The objective is to ensure GTA support to as many courses as possible by finding a *maximal consistent partial-assignment*. Because the hard constraints cannot be violated, the problem cannot be modeled as a MAX-CSP [Freuder and Wallace, 1992].

Glaubius and Choueiry proposed a constraint-based model for this problem where the courses are represented by variables, the GTAs by domain values, and the assignment rules by a number of unary, binary, and non-binary constraints [2001; 2002a; 2002b]. They define the problem as the task of finding the longest assignment, as a primary criterion, and maximizing GTAs’ preferences, as a secondary criterion. (We model the latter as the value of the geometric mean of GTAs’ preferences in an assignment.)

In the Constraint Systems Laboratory, we have implemented a number of search strategies for solving this problem, which we summarize below. These are a heuristic backtrack search (BT) with various ordering heuristics, a greedy local search (LS), a multi-agent-based search (ERA), and a randomized backtrack search with two restart strategies (RGR

and RDGR). The latter are the topic of this dissertation. All strategies implement the above two optimization criteria, except ERA, which models the GTAAP as a satisfaction problem. Below we list the characteristics of the GTAAP data sets.

Problem size: There are 14 data sets of GTAAP: eight original (these are real data, collected from academic semesters in our department) and six boosted data sets (there are over constrained sets to which we have added dummy resources to make them solvable). We used the eight original data sets in our experiments. Table 2.1 lists the characteristics of these eight instances in terms of original or boosted data sets, solvable or unsolvable, number of courses (i.e., number of variables), number of GTAs (i.e. domain size), total capacity of the GTAs, total load of the courses, and number of constraints. Each GTA has

Table 2.1: Characteristics of the GTAAP data sets.

Data Sets	**Data Sets reference**	**Original/Boosted**	**Solvable?**	**#Half semester courses**	**#Courses/#variables**	**#GTAs/Domain size**	**Total capacity**	**Total load**	**Ratio** = $\frac{\text{Total Capacity}}{\text{Total Load}}$	**Constraints**		
										#Unary	**#Binary**	**#Non-Binary**
Spring2001b(O)	1	O	$\times$	12	69	26	26	29.6	0.88	277	1179	52
Fall2001b(O)	2	O	$\surd$	14	65	34	30	29.3	1.02	267	1676	68
Fall2002(O)	3	O	$\times$	10	31	28	11.5	13	0.88	233	1124	56
Fall2002(O)-NP	4	O	$\times$	10	59	28	27	29.5	0.91	233	1124	56
Spring2003(O)	5	O	$\surd$	10	54	34	27.5	27.4	1.00	250	622	68
Spring2003(O)-NP	6	O	$\surd$	12	64	34	31	30.2	1.02	250	622	68
Fall2003(O)	7	O	$\surd$	0	25	27	22	12.8	1.71	235	45	27
Spring2004(O)	8	O	$\surd$	0	41	35	26.5	19.3	1.37	208	32	35

a capacity factor corresponding to the number of hours he/she is hired for and restricting the maximum course weight he/she can be assigned during the semester. The sum of the

capacities of all GTAs represents the *total capacity*. Each course has a load that indicates the weight of the course. For example, a value of 0.5 means the course requires half the capacity of a GTA. Some courses may be only offered during one-half of the semester. The *total load* of a semester is the cumulative load of the individual courses. A ratio of the total capacity to the total load that is strictly less than one indicates that the problem instance is necessarily not solvable.

Types of constraints: In GTAAP, a number of unary, binary, and non-binary constraints specify the allowable assignments. The capacity of each GTA is modeled as a non-binary constraint, called the capacity constraint. We summarize the constraints as follows:

- Unary constraints: English certification, enrollment, overlap, and zero preference constraints.
- Binary constraints: mutex and equality constraints.
- Non-binary constraints: capacity, equality, and confinement constraints.

A detailed description of the problem and the constraints can be found in [Glaubius and Choueiry, 2002a]. Table 2.2 lists the number of unary constraints, number and type of binary constraints, and number of non-binary (capacity) constraints. For the non-binary constraints, we list the maximum, minimum, and mean values of the arity. Also, we list the average and standard deviation of the degree of the variables. Note that our problem typically has a large number of non-binary constraints and their average arity is almost equal to the number of variables. This observation shows that the non-binary constraints are almost global. This fact constitutes the main difficulty in solving this problem.

Quality of solutions: The primary criterion is the number of assigned courses. And, maximizing GTAs' preferences is the secondary criterion. This is calculated as the geometric mean of GTAs's preferences (between 1 and 5 for each course) in an assignment.

Table 2.2: Characteristics of the GTAAP constraints.

Data set	Number of constraints							Degree of variables	
	Unary	Binary		Non-Binary				Mean	Standard deviation
					Arity				
		Mutex	Equality	Capacity	Maximum	Mean	Minimum		
1	277	1146	33	52	63	63	63	85.66	24.77
2	267	1631	45	68	59	58	57	116.35	14.36
3	233	1098	26	56	55	54	53	96.74	14.28
4	233	1098	26	56	55	54	53	93.30	13.37
5	250	575	47	68	58	58	58	85.20	23.73
6	250	575	47	68	58	58	58	84.96	23.26
7	235	6	39	27	61	61	61	31.96	2.04
8	208	5	27	35	52	52	52	40.12	2.23

Partial solution: Some instances of GTAAP are over-constrained and do not have a complete solution. For such instances, only a partial solution can be obtained. Here we need to note that, strictly speaking, GTAAP is not a MAX-CSP. In MAX-CSP, all constraints are soft and the goal is to maximize the number of satisfied constraints. Thus, in the solution of a MAX-CSP problem all variables are assigned, but the solution is not necessarily consistent. In GTAAP, however, it is not permissible for any constraint to be broken, but some variables may remain unassigned.

2.3 Heuristic backtrack search

A deterministic backtrack (BT) search, implemented as a depth-first search, was the first search technique implemented to solve GTAAP by Glaubius and Choueiry [2002a]. Their implementation integrated forward checking [Haralick and Elliott, 1980] and a branch-and-bound mechanism to seek the optimal solutions.

2.3.1 Basic mechanism

A backtrack search instantiates the variables of a CSP incrementally from values present in its current domain. When a variable is instantiated, the search looks ahead towards the future variables, and removes incompatible values from their current domain. When the next variable is instantiated, we can be sure that it is compatible with the past variables. A full look-ahead strategy would drastically increase the number of constraint checks while effectively yielding little filtering since the GTA application has many mutex and global constraints (it is a resource allocation problem). As depth-first search expands nodes along a search path, the search checks if the expansion of the search path can improve on the current best solution. Once the current best solution cannot be improved, backtrack occurs. Because the problem may be over-constrained, Glaubius and Choueiry modified the backtrack mechanism to allow null assignments and proceed toward the longest solution in a branch-and-bound manner (i.e., backtracking is not performed when a domain is wiped-out as long as there are future variables with no empty domains). The implementation is described in detail in [Glaubius and Choueiry, 2002a].

2.3.2 Variable ordering heuristics

Glaubius and Choueiry implemented two variable-ordering heuristics for choosing the most constrained variable first: *least domain* (LD) and *domain degree* (DD) ratio. In LD, we choose as current variable the one with the smallest current domain-size. In DD, we choose as current variable the one with the minimum ratio of domain size to the degree. This is based on the intuition that the most constrained variable (i.e., the smallest domain and the largest degree) would reduce the branching factor and implement the well-known fail-first principle. Ties are broken lexicographically.

The two variable-ordering heuristics, LD and DD, are applied both statically and dynam-

ically. In a static ordering, the order is specified before the search begins and is not changed thereafter. In a dynamic ordering, the order changes after each instantiation. Extensive previous experiments and analysis done by the research community, have demonstrated that dynamic ordering substantially reduces the cost of any search. Table 2.3 shows the four variable-ordering heuristics available for GTAAP.

Table 2.3: Variable ordering heuristics.

		Variable ordering	
		Least Domain	Domain Degree ratio
Strategy	Static	SLD	SDD
	Dynamic	DLD	DDD

2.3.3 Value ordering heuristics

In general, the choice of the value to be assigned to the current variable is orthogonal to the choice of the variable to be instantiated. The intuition is that we should assign first the value most likely to yield a solution. This is true and worthwhile when we are looking for the first solution only. If all the solutions are required, or when there are no solutions, then the order in which the values are considered makes no difference. Three value-ordering heuristics are available for GTAAP.

1. First in line (FIL): FIL is an arbitrary method, which depends on how the domains are stored. The first value that is present in the domain of the current variable is assigned to the current variable.

2. Highest preference (PREFERENCE): This heuristic considers only the current variable. The value (GTA) assigned to the current variable (course) is the one having the

highest preference for that course. If there are more than one value available then the choice is made by the order of appearance.

3. Least occurring (`OCCURRENCE`): This heuristic considers the current domain of all the future variables. The value that appears least frequently in the domain of future variables is assigned to the current variable. This is based on the assumption that selecting the value that occurs the least number of times in the future variables will increase the number of options for instantiating future variables, and thus is the least likely to lead to a conflict. If there are more than one value available then the choice is made by the order of appearance.

Glaubius and Choueiry implemented `FIL` and `PREFERENCE` while we implemented `OCCURRENCE`. The four variable-ordering heuristics of Table 2.3 and the three value-ordering heuristics listed above result in 12 combinations, which we evaluate in Chapter 3.

2.4 Local search

Zou and Choueiry designed and implemented a greedy, local search (LS) technique for solving GTAAP [2003a; 2003; 2003b]. LS is a hill-climbing search using the min-conflict heuristic for value selection [Minton *et al.*, 1992]. In a CSP, a state is an assignment of values to all variables. This may be inconsistent with the constraints. Local search starts from an initial state, usually chosen randomly, and explores neighboring states until it reaches an optimal state. The neighboring states are those that can be reached by the changing the assignment of one variable. The name local comes from the fact that it only moves from a state to its neighboring state. The evaluation value of a state is its number of constraint violations. A hill-climbing strategy allows only moves that reduce the evaluation value. For value selection, the min-conflict heuristic orders the values according to the number of constraints violations after each move. A variable is said to be in conflict if it

violates any constraint. At each iteration any variable that is in conflict is assigned a value that minimizes the number of conflicts, breaking ties randomly.

LS continues until the value of current state is better than the values of all the states adjacent to it. At this point, the current state is either an optimum or a local optimum. A weakness of a hill-climbing search is that it may stagnate either on a *local optimum* or on a *plateau* (Figure 2.1). A local optimum is a state that is the best amongst its neighbor

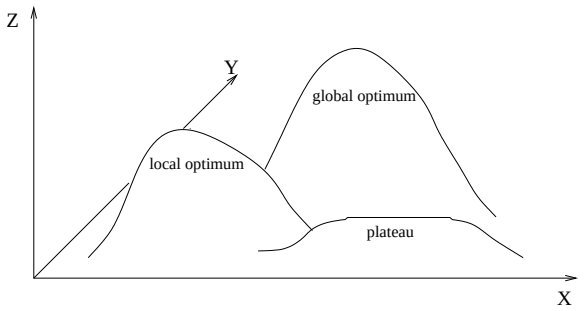

Figure 2.1: Local optimum and plateau with hill-climbing [Zou, 2003].

states but is not the optimum. A plateau is a state whose evaluation value is equal to all the neighboring states. Since the neighboring states are not better than the states of local optimum and plateau, local search stagnates.

Random walk is a strategy utilized to avoid local optima. In random walk, the value of a variable chosen using the min-conflict heuristic is with a probability $1 - p$, and with probability p this value is chosen randomly. Preliminary studies on the influence of random walk is presented in [Wallace and Freuder, 1995; Wallace, 1996]. Following the indications of [Barták, 1998] and after testing, Zou and Choueiry use a value of $p = 0.02$. Finally, they use random restarts to break out of local optima. In random restarts, search is started from a new randomly selected state while keeping track of the best solution obtained so far, thus giving the resulting algorithm an anytime flavor.

2.5 Multi-agent-based search

Inspired by swarm intelligence, Liu et al. [2002] proposed a multi-agent-based search, called ERA (i.e., Environment, Reactive rules, and Agents) for solving CSPs. Zou and Choueiry adapted and implemented ERA algorithm to solve GTAAP [2003a; 2003b; 2003].

An ERA system has three components: an Environment (E), a set of Reactive rules (R), and a set of Agents (A). The environment records the number of constraint violations of the current state for each value in the domains of all variables. Each variable is an agent, and the position of the agent corresponds to the value assigned to this variable. Each agent moves according to its reactive rules. First, ERA places the agents randomly in their allowed positions in the environment, then it considers each agent in sequence. For a given agent, it computes the constraint violations of each agent's position. Each agent moves to occupy a position (`zero position`) that does not break any of the constraints that apply to it. If the agent is already in a `zero position`, no change is made. Otherwise, the agent chooses a position to move to, the choice being determined stochastically by the reactive rules. The agents keep moving until they all reach a `zero position` (i.e., a full, consistent solution) or a certain time period has elapsed. After the last iteration, only the CSP variable corresponding to agents in `zero position` are effectively instantiated. The remaining ones remain unassigned (i.e., unbounded).

This algorithm acts as an 'extremely' decentralized local search, where any agent can move to any position, likely *forcing other agents to seek other positions*. This extreme mobility of agents in the environment is the reason for ERA's unique immunity to local optima, as uncovered by the experiments by Zou and Choueiry [2003a; 2003; 2003b]. It is indeed the only search technique to solve instances that remain unsolved by any other technique we tested. Zou and Choueiry also uncovered the weakness of ERA on over-constrained problems, where a livelock[1] phenomenon undermines its stability resulting in particularly

[1] Although they called it a 'deadlock,' livelock is a more appropriate term because search is not halted.

short solutions. However, they show that this phenomenon can be advantageously used to isolate, identify, and represent conflicts in a compact manner. In their implementation, agents move in sequence, but the technique can also be asynchronous.

Although ERA can be viewed as an extension to LS, it differs from LS in some subtle ways. LS moves from one state to another by changing the assignment of one (or two) variables, while in ERA any number of variables can change positions at each move; each agent chooses its most convenient position (i.e., value). The evaluation function to assess the quality of a given state in LS is a global account of the quality of the state (typically the total number of broken constraints). In ERA, no such global evaluation function is used. ERA appears as an extremely decentralized version of LS and where the selection of the next state is determined, locally, by the individual agents.

2.6 Randomized backtrack search with restarts

Unlike ERA and local search, general backtrack (BT) search is, in principle, complete and sound. However, the performance of heuristic BT heavily depends on the accuracy of the ordering heuristic, which at shallow levels in the search tree is often myopic. This results in BT being unpredictable in practice over a set of problem instances, even within the same problem type. The performance of BT also depends on the branching factor of the problem. Greater the branching factor of a problem, greater is the effort required to undo incorrect heuristic choices made early in the search process. This results in BT being seriously undermined by thrashing (i.e., searching unpromising parts of the search space). As the problem size increases, the effects of thrashing become more serious.

Gomes et al. demonstrated that randomization of heuristic choices combined with restart mechanisms is effective in overcoming the effects of thrashing and in reducing the total execution time of BT [1998]. Thrashing in BT indicates that search is stuck exploring an

unpromising part of the search space, and thus incapable of improving the quality of the current solution. It becomes apparent that there is a need to interrupt search and to explore other areas of the search space. The restarted search must begin from a different portion of the search space; otherwise it will end up traversing the same paths. Randomization of branching during search is used to this end. Randomness can be introduced in the variable and/or value ordering heuristics, either for tie-breaking or for variable and/or value selection. After choosing a randomization method, the algorithm designer must decide on the type of restart mechanism. This restart mechanism determines when to abandon a particular run and restart the search. Here the tradeoff is that reducing the cutoff time reduces the probability of reaching a solution at a particular run.

2.6.1 Restart strategies

Several restart strategies have been proposed with different cutoff schedules. Some of the better known ones are the fixed-cutoff strategy and Luby et al.'s universal strategy [1993], the randomization and rapid restart (RRR) of Gomes et al. [1998], and the randomization and geometric restarts (RGR) of Walsh [1999]. Among the above listed restart strategies, RRR and RGR have been studied and empirically tested in the context of CSPs. All of these restart strategies are static in nature, i.e. the cutoff value for each restart is independent of the progress made during search. Some restart strategies (e.g., fixed-cutoff strategy of [Luby *et al.*, 1993] and RRR [Gomes *et al.*, 1998]) employ an optimal cutoff value that is fixed for *all* the restarts of a particular problem instance. The estimation of the optimal cutoff value requires a priori knowledge of the cost distribution of that problem instance, which is not known in most settings and must be determined by trial-and-error. This is clearly not practical for real-world applications. There are other restart strategies that do not need any a priori knowledge (e.g., Luby et al.'s universal strategy [1993] and Walsh's RGR [1999]). They utilize the idea of an increasing cutoff value in order to ensure the

completeness of search. However, if these restart strategies do not find a solution after the initial few restarts, then the increasing cutoff value leads to fewer restarts, which may yield thrashing and diminishes the benefits of the restart.

2.6.2 Randomization and geometric restarts

Walsh proposed the Randomization and Geometric Restarts (RGR) strategy to automate the choice of the cutoff value [1999]. According to RGR, search proceeds until it reaches a cutoff value for the number of nodes visited. The cutoff value for each restart is a constant factor, r, larger than the previous run. The initial cutoff is equal to the number of variables n. This fixes the cutoff value of the i^{th} restart at $n.r^i$ nodes. The geometrically increasing cutoff value ensures completeness with the hope of solving the problem before the cutoff value increases to a large value. We implemented RGR, studied various values of r, and compared it with our proposed restart strategy (RDGR). In our implementation, RGR was combined with BT of Section 2.3 and the randomized selection of variable-value pairs of Section 3.3.

2.6.3 Dynamic restarts

Unlike the static restart strategies of Section 2.6.1, Kautz et al. introduced an optimal policy for dynamic restarts [2002]. They employ Bayesian methods to build predictive models for run-time distribution. Utilizing this model, their restart strategy considers the predictions about the running time to choose the cutoff value for restarts. This approach is based on the use of machine learning techniques. Machine learning techniques require large collections of problem instances to train the predictive models. This is clearly not practical for real-world applications. And, unlike other restart strategies, Bayesian methods increase the complexity of the implementation and deployment.

Given these drawbacks, we propose a restart strategy that dynamically adapts the cutoff value for each restart based on the performance of previous restarts and does not require any complex computations. We do this at the expense of completeness, which, anyway, is not achievable on large problems.

2.7 Las Vegas algorithms

BT is deterministic and the other three search techniques (i.e., LS, ERA, and RGR) are stochastic. These stochastic search techniques can be further classified in terms of Las Vegas algorithms. A Las Vegas algorithm always yields correct solutions and its run-time is a random variable. According to Hoos, an algorithm A for a problem class Π is a *Las Vegas Algorithm* (LVA) if it has the following properties [1998]:

- If for a given problem instance $\pi \in \Pi$, algorithm A returns a solution s, s is guaranteed to be a correct solution of π.
- For each given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable $RT_{A,\pi}$.

The solutions returned by Las Vegas algorithms are guaranteed to be correct. However, Las Vegas algorithms are not guaranteed to be complete. Based on the property of completeness, Hoos classifies Las Vegas algorithms into the following three categories [1998]. Consider a Las Vegas algorithm A for a problem class Π, and let $P_s(RT_{A,\pi} \leq t)$ denote the probability that A finds a solution for a soluble instance $\pi \in \Pi$ in time less than or equal to t. A is said to be:

1. *Complete*, if and only if for each soluble instance $\pi \in \Pi$ there exists some t_{max} such that $P_s(RT_{A,\pi} \leq t_{max}) = 1$;

2. *Probabilistically approximately complete (PAC)*, if and only if for each soluble instance $\pi \in \Pi$, $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) = 1$;

3. *Essentially incomplete*, if it is not PAC, i.e. if there exists a soluble instance $\pi \in \Pi$, for which $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) < 1$.

Table 2.4 shows the classification of LS, ERA, and RGR as Las Vegas algorithms. The

Table 2.4: Las Vegas algorithms.

Search method	**Las Vegas algorithms**
RGR	*Probabilistically approximately complete*
RDGR	*Essentially incomplete*
ERA	*Essentially incomplete*
LS	*Essentially incomplete*

increasing cutoff parameter in RGR ensures that it will traverse, in theory, the whole search space. This makes RGR a probabilistically approximately complete algorithm. In our proposed restart strategy, RDGR, completeness is not guaranteed because the cutoff parameter may remain constant. Local search algorithms are essential incomplete due to search getting stuck in local optima. Even if some techniques such as random restarts, random walk, or tabu search are applied to escape from local optima, the local search algorithms cannot be guaranteed to achieve completeness. ERA also does not guarantee completeness.

Summary

CSP provides a framework that allows researchers to study and solve problems. GTAAP is a real-world resource allocation problem that offers an opportunity to develop and test new search techniques. In practice, this problem is tight, even over-constrained. Various search techniques (i.e., BT, LS, ERA, and randomized backtrack search with restarts) have been implemented and developed for solving GTAAP.

Chapter 3

Study of backtrack search

Deterministic backtrack (BT) search was the first search technique implemented to solve GTAAP. Although BT is theoretically sound and complete, the large size of the search space and thrashing *make such guarantees meaningless in practice*. This chapter addresses the influence of variable/value selection on the performance of backtrack search. First, we study the effect of various heuristics for this selection, then we study the effect of randomization.

3.1 Study of deterministic ordering heuristics

In Section 2.3 we summarized the four variable-ordering heuristics and the three value-ordering heuristics implemented for BT. We tested these 12 combinations on the GTAAP data sets of Table 2.1. The parameters for the experiments were the GTAAP data set, variable-ordering heuristic, and value-ordering heuristic. Given the large size of the problem instances, we had to limit the time of each experiment to 5 minutes. The detailed results for each heuristic can be found in Appendix A.2. Table 3.1 summarizes the best results obtained by using all the variable-value ordering heuristics on all the GTAAP data sets. Table 3.1 shows the number of unassigned courses, the solution quality in terms of

Table 3.1: Best results of BT using deterministic ordering heuristics: GTAAP (5 minutes).

Data Sets	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	Time [sec]	CC ($\times 10^8$)	Ordering Heuristic
1	12	2.31	0	4.5	94	1.39	DDD-OCCURRENCE
2	2	2.65	0	2.7	22	0.92	DLD-OCCURRENCE
3	3	3.73	0	1.5	2	0.04	DLD-PREFERENCE
4	7	2.86	0	3.5	162	2.27	DDD-FIL
5	0	3.22	0	1.5	25	0.75	DLD-OCCURRENCE
6	2	4.15	0	2.3	25	1.07	DLD-PREFERENCE
7	6	2.88	0	4.2	253	0.92	DDD-FIL
8	1	3.89	3	4.6	102	0.50	DLD-FIL

the geometric mean of the assigned GTAs preferences, the number of unused GTAs, available resources, time taken for finding the best solution, the number of constraint checks needed to find the best solution, and the combination of variable-value ordering heuristics that obtained the best solution for each GTAAP data-set.

Observation 3.0.1. *Dynamic variable-ordering heuristics outperform static variable-ordering heuristics.*

Table 3.1 shows that for all the data sets of the GTAAP, either DLD or DDD variable-ordering heuristic yields the best solution. None of the static variable-ordering heuristics find better solutions than dynamic variable-ordering heuristics.

Observation 3.0.2. DD *and* LD *show similar performances.*

Table 3.1 shows that LD yields the best result for five instances of the GTAAP (i.e., data sets 2, 3, 5, 6, and 8), while DD yields the best result for the remaining 3 instances. Clearly, neither DD nor LD outperforms the other.

Observation 3.0.3. *All value-ordering heuristics show similar performances.*

Table 3.1 shows that the FIL yields the best solution for data sets 4, 7, and 8. PREFERENCE yields the best solution for data sets 3 and 6. And, OCCURRENCE yields the best solution for data sets 1, 2, and 5. Clearly, no value-ordering heuristic is the winner. Table 3.2 summarizes the relative performance of the various deterministic ordering heuristics.

Table 3.2: Relative performance of the deterministic ordering heuristics.

Dynamic selection strategy $\succ$ Static selection strategy
DD $\approx$ LD
OCCURRENCE $\approx$ PREFERENCE $\approx$ FIL

3.2 Thrashing

Thrashing is the phenomenon of BT searching unpromising parts of the search space. It is severe in problems of large size, such as the GTAAP. Due to the high branching factor of the problem, BT is unable to backtrack to the place of bad heuristic choice. Figure 3.1 illustrates thrashing for data set 1 with 69 variables and 26 values. In this figure, the percentage

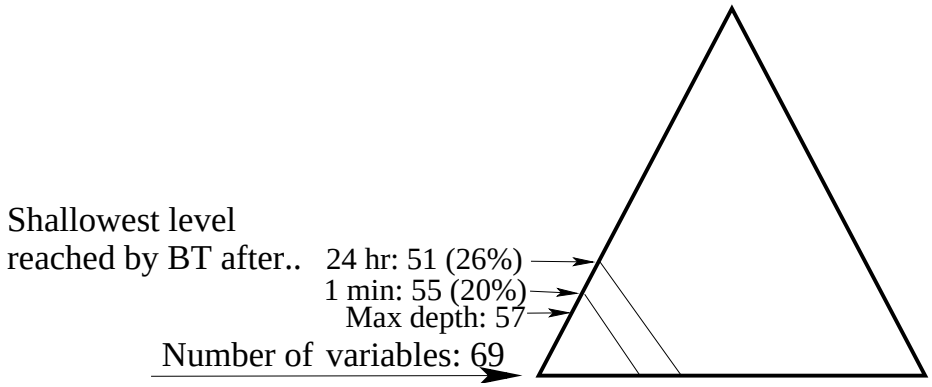

Figure 3.1: BT search thrashing in large search spaces.

denotes:

$$\frac{\text{number of variables} \; - \; \text{shallowest level reached by backtracking}}{\text{number of variables}}. \qquad (3.1)$$

Indeed, the shallowest level of backtrack achieved after 24 hours (26%) is not significantly better than that reached after 1 minute (20%) of search, never revising the initial assignment of 74% of the variables. Table 3.3 shows that this phenomenon is also present in other data sets. Table 3.3 shows, for each data set, the number of variables, the longest solution

Table 3.3: BT search thrashing.

Data set	**# Vars**	**BT running for..**					
		5 min			6 hours		
		Max depth	Shallowest level	%	Max depth	Shallowest level	%
1	69	57	53	23%	57	51	26 %
2	65	63	55	15%	63	54	16 %
3	31	28	13	58%	28	3	90 %
4	59	49	48	18%	50	45	23 %
5	54	52	44	18%	54	41	24 %
6	64	62	54	15%	62	47	26 %
7	25	19	7	72%	21	0	100 %
8	41	40	19	53%	40	17	58%

(maximum depth), and the shallowest BT level (and the corresponding percentage) attained by backtracking after 5 minutes and 6 hours.

To confirm that this phenomenon is not solely due to the short run-time of search, we conducted further experiments by allowing search to run for longer time periods. Table 3.4 shows the performance of BT on data set 1 for various running times, up to 24 hours. We notice that increasing the running time of BT does not yield significant improvements in terms of the length of the solution found or its quality. Indeed, the number of unassigned variables does not decrease, the quality of the solution measured by its geometric mean does not significantly increase, and the shallowest level of backtracking is not sig-

Table 3.4: Performance of BT for various running times.

Data set 1 (69 variables, over-constrained)						
CPU run-time	30 sec	5 min	30 min	1 hour	6 hours	24 hours
Shallowest BT level	54	53	52	52	51	51
Longest solution	57	57	57	57	57	57
Geometric mean of preference values	2.15	2.17	2.17	2.21	2.27	2.27
# Backtracks	1835	47951	261536	532787	3274767	13070031
# Nodes visited	3526	89788	486462	989136	6059638	24146133
# Constraint checks	8.50E+07	3.17E+08	1.81E+09	3.58E+09	2.16E+10	8.70E+10

nificantly reduced (first 51 variables in the ordering is never undone). Further, the cost of search, in terms of number of backtracks, nodes visited, and constraint checks is significantly increased. Clearly, BT is being undermined by severe thrashing. We conclude that the backtrack mechanism is operating on the deeper levels of the search tree and seems to be unable to undo early choices and there is no benefit from letting BT run over longer period of time.

3.3 Randomized backtrack search

We implemented two random-ordering heuristics:

1. Random variable-ordering heuristic (`Rvar`) and

2. Random value-ordering heuristic (`Rval`).

`Rvar` randomly selects the current variable from the set of future variables. And, `Rval` instantiates the current variable with a randomly chosen value from its current domain. Both these heuristics are applied at every instantiation (i.e., dynamically). The resulting randomized backtrack search can be characterized as a *complete Las Vegas algorithm*. We test the randomized backtrack search on the GTAAP data sets of Table 2.1 and show the results in Table 3.5. Given the stochastic nature of the randomized backtrack search,

Table 3.5: Randomized backtrack search: GTAAP (5 minutes).

Data-sets	Unassigned variables	Time sec	CC ($X10^8$) best solution	CC ($X10^8$) 5 mins	NV ($X10^3$) best solution	NV ($X10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($X10^3$) to best solution	BT ($X10^3$) in 5 mins
	7	181	12.78	14.43	155.40	172.17	69	54	19.42	33.03
Data set 1	13	19	13.78	18.19	202.22	219.51	69	45	0.52	13.36
	10	17	21.82	25.87	226.08	262.92	69	51	0.97	30.71
	1	207	3.96	5.33	15.68	21.51	65	54	10.73	14.59
Data set 2	5	19	0.70	4.84	1.03	22.09	65	56	0.74	17.03
	3	20	0.70	4.85	1.03	22.13	65	56	0.74	17.07
	8	5	0.07	4.62	0.81	30.92	31	8	0.65	23.96
Data set 3	2	5	0.07	4.63	0.81	30.88	31	8	0.65	23.93
	5	5	0.07	4.63	0.81	30.82	31	10	0.65	23.88
	7	160	2.25	3.05	27.00	50.05	59	47	12.03	30.37
Data set 4	8	150	1.75	3.08	40.15	98.07	59	48	13.07	35.80
	10	16	0.52	3.39	2.07	74.62	59	48	1.13	41.06
	11	262	4.40	5.01	12.16	13.59	54	35	9.29	10.31
Data set 5	1	264	4.40	4.97	12.16	13.52	54	39	9.29	10.25
	0	264	4.40	4.98	12.16	13.53	54	43	9.29	10.26
	13	113	2.25	5.42	6.11	14.11	64	43	2.89	6.80
Data set 6	15	80	1.97	5.34	5.75	20.07	64	49	2.70	13.03
	2	25	1.07	4.64	0.08	13.92	64	55	0.05	5.28
	7	30	0.35	0.52	78.35	301.20	25	6	20.02	190.03
Data set 7	8	158	0.28	0.54	168.73	316.88	25	7	97.51	183.67
	8	150	0.30	0.55	165.66	315.45	25	7	95.55	184.43
	13	198	0.78	1.24	83.82	122.45	41	24	21.22	34.56
Data set 8	6	217	0.32	0.43	88.02	164.36	41	27	30.20	35.04
	1	102	0.50	1.64	33.17	78.88	41	19	10.00	28.55

Time: CPU run-time needed to reach best solution.

CC: Number of constraint checks.

NV: Number of nodes visited.

BT: Number of backtracks.

we conducted three experiments for each GTAAP data set, each experiment running for 5 minutes. Table 3.5 shows the number of unassigned variables, time taken to find the best solution, number of constraint checks for finding the best solution and during 5 minutes, number of nodes visited for finding the best solution and during 5 minutes, depth of the search tree, shallowest level reached by backtracking, and number of backtracks to finding the best solution and during in 5 minutes.

Observation 3.0.4. *Random-ordering heuristics may or may not yield better solutions than deterministic-ordering heuristics.*

The best result for data set 1 using random-ordering heuristics is 7 unassigned variables (Table 3.5), while it is 12 for deterministic-ordering heuristics (Table 3.1). However, random-ordering heuristics can also yield worse results (13 unassigned variables for data set 1) than deterministic-ordering heuristics. This is due to the stochastic nature of randomized backtrack search. This variation in the results can be mitigated by the use of restart strategies, as discussed in Section 2.6.

Summary

In this chapter we study the performance of the various ordering heuristics implemented for BT. As expected, our experiments show that dynamic selections are consistently superior to static selections. However, none of the variable-ordering heuristics is dominant. Similarly, none of the value-ordering heuristics outperforms any of the others. We highlight the severe thrashing in BT. We investigate randomized backtrack search as an alternative to deterministic backtrack search and show that it needs to be augmented with restart strategies.

Chapter 4

Randomization and dynamic geometric restarts

In this chapter we first introduce the design and implementation of our proposed restart strategy. Next, we present the empirical evaluation of the performance of our restart strategy by comparing it against all the search techniques (i.e., BT, LS, ERA, and RGR) developed to solve GTAAP, both on GTAAP data-sets and on randomly generated problems. We describe the evaluation methodology and present the results.

4.1 Randomization and dynamic geometric restarts

In Section 2.6, we described the various static restart-strategies and their drawbacks. RGR employs a geometrically increasing cutoff value. This ensures completeness of the search, but results in fewer restarts, thus increasing the likelihood of thrashing and diminishing the probability of finding a solution. Our proposed strategy, Randomization and *Dynamic* Geometric Restarts (RDGR), aims to attenuate this effect. It operates by not increasing the cutoff value for the following restart whenever the quality of the current best solution

is not improved upon. When the current restart improves on the current best solution, then the cutoff value is increased geometrically, similar to RGR. Because the cutoff value may stay constant, completeness is no longer guaranteed (i.e., essentially incomplete Las Vegas algorithm). This situation is acceptable in application domains (like ours) with large problem size where completeness is, anyway, infeasible in practice. Let C_i be cutoff value for the i^{th} restart and r be the ratio used to increase the cutoff value. In RGR the cutoff value is updated according to the equation: $C_{i+1} = r.C_i$. We use the following equation in RDGR:

$$C_{i+1} = \begin{cases} r.C_i & \text{when the solution has improved at the } i^{th} \text{ restart} \\ C_i & \text{otherwise} \end{cases} \tag{4.1}$$

In RGR, the cutoff value for each restart is determined *independently* of how search performed at the previous step. However, this is not the case for RDGR. RGR and RDGR follow the same cutoff schedules for search paths that improve solutions.

As search proceeds, the cutoff values in RGR keep on increasing. This results in RGR becoming more of a randomized backtrack search than a randomized backtrack search with restarts. In contrast, the cutoff values in RDGR remain at a smaller value compared to that of RGR. This results in more restarts taking place in RDGR than RGR. However, for $r = 1$ the cutoff values for both RGR and RDGR are similar, resulting in similar performance. The dynamic nature of RDGR arises due to the fact that the cutoff schedule of RDGR is different for each search process, and is not fixed as RRR or static as RGR.

We implemented RDGR with the BT of Section 2.3 and the randomized selection of variable-value pairs of Section 3.3.

4.2 Experimental methodology

We tested and compared the 5 search strategies, namely: BT (Section 2.3), LS (Section 2.4), ERA (Section 2.5), RGR (Section 2.6.2), and RDGR. BT is deterministic and the other 4 search techniques (i.e., LS, ERA, RGR, and RDGR) are stochastic.

4.2.1 Main experiments

We conducted the following three sets of experiments:

1. Effect of running time on RGR and RDGR.
2. The influence of the choice of the ratio r used in RGR and RDGR.
3. Relative performance of BT, LS, ERA, RGR, and RDGR.

4.2.2 Evaluation criteria

We compare the performance of the algorithms using the following criteria:

1. *Solution quality distributions* (SQD) taking as reference the longest known solution for each data set, as recommended by Hoos and Stützle in [2004]. SQD's are cumulative distributions of the solution quality, similar to the cumulative distributions of run-time in run-time distributions. The horizontal axes represents in percent the relative deviation of the solution size s from the longest known solution s_{opt}, computed as $\frac{(s_{opt}-s)100}{s_{opt}}$. Thus, the point 0% on the x-axis denotes a solution as long as the longest known solution, the point 20% denotes a solution that is 20% shorter than the longest known solution. The y-axis shows the percentage of test runs that obtain a solution in terms of deviation from the best known solution.

2. *Descriptive statistics* of all the solutions found, for all search techniques. This includes the measures: mean, median, mode, standard deviation, minimum, and maximum length of the solution.

3. *95% confidence interval* to assess the performance differences of RDGR over RGR and ERA. For single problem instance (i.e., GTAAP data sets), we use the Mann-Whitney U-test for the computation of confidence intervals. For ensembles of instances (i.e., randomly generated problems), we use the Wilcoxon matched pairs signed-rank test.

4.2.3 Data sets

We tested these search techniques on the 8 GTAAP data-sets of Table 2.1 and 4 sets of randomly generated binary CSPs. For the GTAAP data sets, we repeated our experiments 500 times for all stochastic search techniques. Naturally, a single run is sufficient for BT because it is deterministic.

4.2.3.1 GTAAP data sets

Figure 4.1 shows the moving averages for the running times of RGR and RDGR. Moving averages is an indicator that shows the average running-time over various sample sizes. We found that the average run-time for all stochastic algorithms stabilizes after 300 runs on all the GTAAP data sets, as shown in Figure 4.1 for data set 1, which justifies our decision of repeating the experiments 500 times. We report the results for the following data sets:

1. Data set 1 as a representative of an over-constrained problem.

2. Data set 5 as a representative of a tight but solvable problem.

The results for all data sets are qualitatively equivalent and can be found in Appendices A.3, A.4, and A.5.

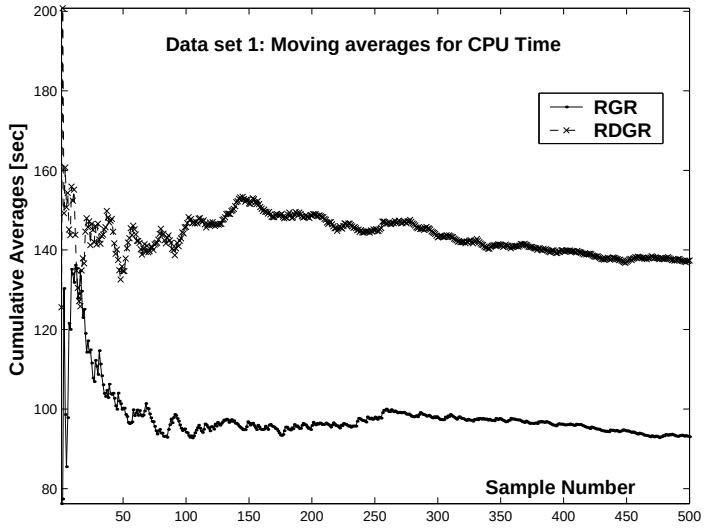

Figure 4.1: Moving average for CPU run-times for data set 1.

4.2.3.2 Randomly generated problems

We also evaluated all the search techniques on randomly generated binary CSPs. Before we started our work, all the search techniques (except RGR and RDGR) were customized for GTAAP data. We adapted all the search techniques to solve randomly generated problems. All the randomly generated binary CSPs, used for testing, are of the model B type, generated using the random generator of [van Hemert, 2004]. We generated three types of random problems as shown in Figure 4.2, where $\langle n, a, p_1, t \rangle$ denote the number of variables, the (uniform) domain size, the proportion of constraints and the (uniform) constraint tightness:

1. *Under-constrained instances*. The first type of randomly generated problems are under-constrained binary CSPs with 40 variables, uniform domain size of 20 values, 0.5 proportion of constraints, and 0.2 constraint tightness. We generated 100 instances and ran each instance for 3 minutes.

2. *Over-constrained instances*. The second type of randomly generated problems are over-constrained binary CSPs with 40 variables, uniform domain size of 20 values,

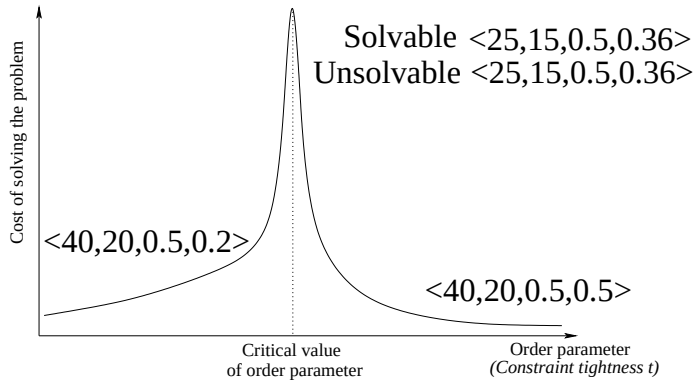

Figure 4.2: Randomly generated problem instances.

0.5 proportion of constraints, and 0.5 constraint tightness. We generated 100 instances and ran each instance for 3 minutes.

3. *Instances at the phase transition*. The third type of randomly generated problems are from the *phase transition* area. In binary CSPs, as the tightness is varied from small to larger values, a transition occurs from a region containing problems having many solutions to a region in which almost all problems have no solutions. The values of tightness where this transition occurs is known as the phase transition and acknowledged to contain the hardest problem instances for a given number of variables, domain size, and proportion of constraints. At the phase transition, we generated random binary CSPs with 25 variables, uniform domain size of 15 values, 0.5 proportion of constraints, and 0.36 constraint tightness. These problems were split into two sets. The first set contains 87 solvable instances, and the second set contains 111 unsolvable instances. All the solvable and unsolvable instances are from the random generator of [van Hemert, 2004]. We ran each instance for 3 minutes.

4.3 Effect of the running time on RGR and RDGR

To compare the performance of RGR and RDGR, we tested them on various running times for the GTAAP data sets. Figures 4.3 and 4.4 show the results in terms of SQDs.

The x-axis shows the deviation from the best known solution for RGR and RDGR with varying run-times. And, the y-axis shows the percentage of test runs that obtain a solution in terms of deviation from the best known solution. The distributions for greater run-times lie to the right of the smaller run-times. This is because, with increasing run-times, better solutions are found. In both the figures, the distribution of RDGR for 20 minutes running time is to the right of the corresponding distribution of RGR. This means that the probability of RDGR finding a particular solution quality is more than RGR. This is true for all the run-times. Thus, RDGR consistently outperforms RGR over different run-times. Clearly, increasing the running time has no affect on the relative dominance of the two algorithms.

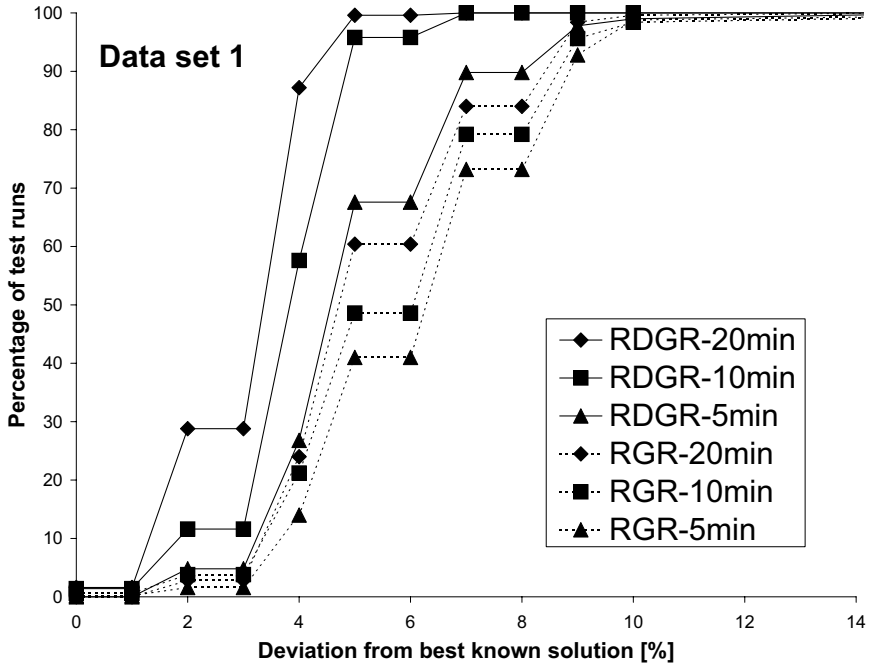

Figure 4.3: Varying run time: GTAAP, over-constrained.

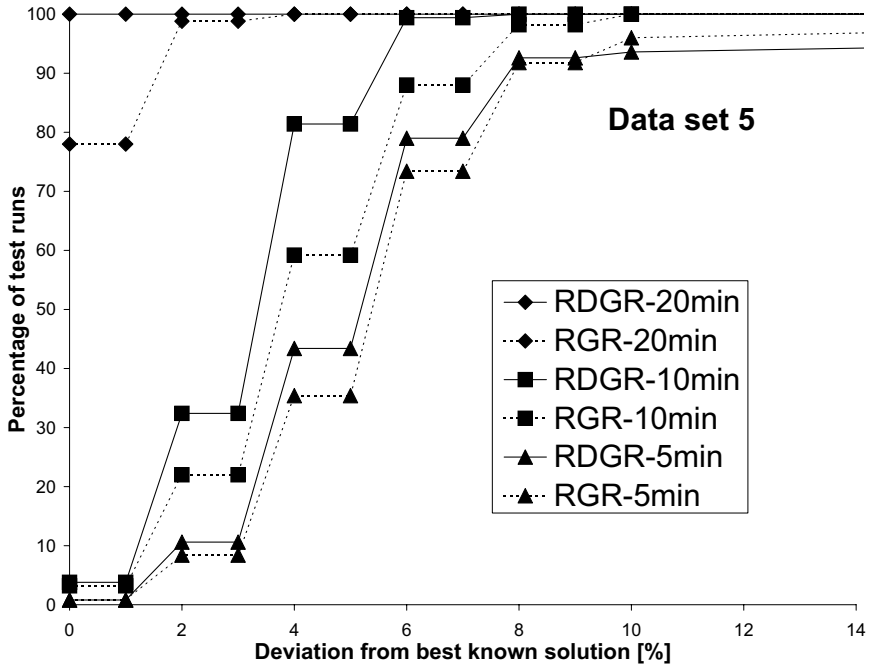

Figure 4.4: Varying run time: GTAAP, solvable.

4.4 Influence of the ratio r

We tested RGR and RDGR with different ratios, with 5 minutes running time. For the GTAAP problem we tested the values: 1, 1.1, $2^{\frac{1}{4}}$, $2^{\frac{1}{2}}$, 2, and 4. For the random CSPs we tested the values: 1, 1.1, $2^{\frac{1}{4}}$, $2^{\frac{1}{2}}$, 2, 3, and 4. Figures 4.5, 4.6, 4.7, and 4.8 show the percentage of test runs that obtain solutions for the different values of the ratio r used to increase the cutoff value in RGR and RDGR.

For example, in Figure 4.5, for data set 1 using a value of r=1.1, nearly 90% of the runs obtain solutions that are 6% deviant from the best known solution. While other values of r yield poorer results. Thus, in accordance with [Walsh, 1999], Figures 4.5 and 4.7 show that a value of r=1.1 is the best among the values tested for RGR. While this optimal ratio does not change with the problem type (i.e., GTAAP vs. random problem) for RGR, it does for RDGR. For the GTAAP, it is r=1.1 (Figure 4.6). For randomly generated problems, it is r=2 (Figure 4.8). We conducted further tests on larger randomly generated binary CSPs having 60 variables, uniform domain size of 30 values, 0.5 proportion of constraints, and 0.5 constraint tightness. Figure 4.8 shows that even with a larger problem size, the best value for RDGR is r=2. This ensures that the best value of r is not dependent on the problem size.

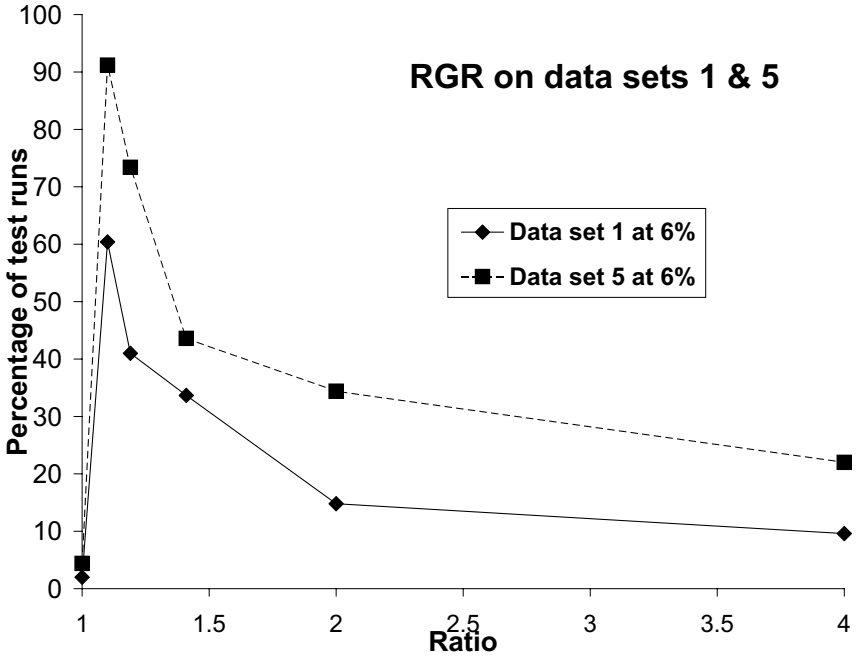

Figure 4.5: Effect of r: RGR on GTAAP.

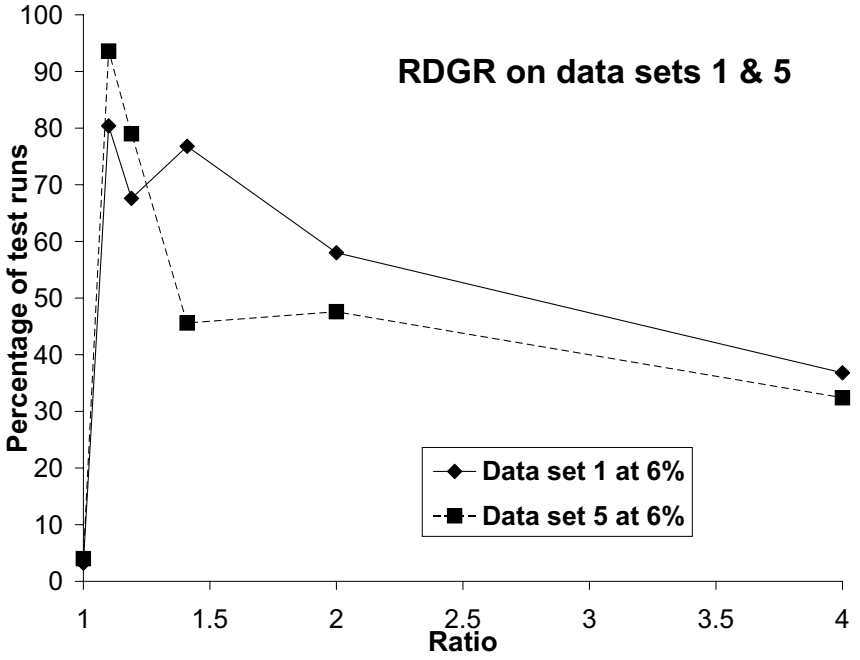

Figure 4.6: Effect of r: RDGR on GTAAP.

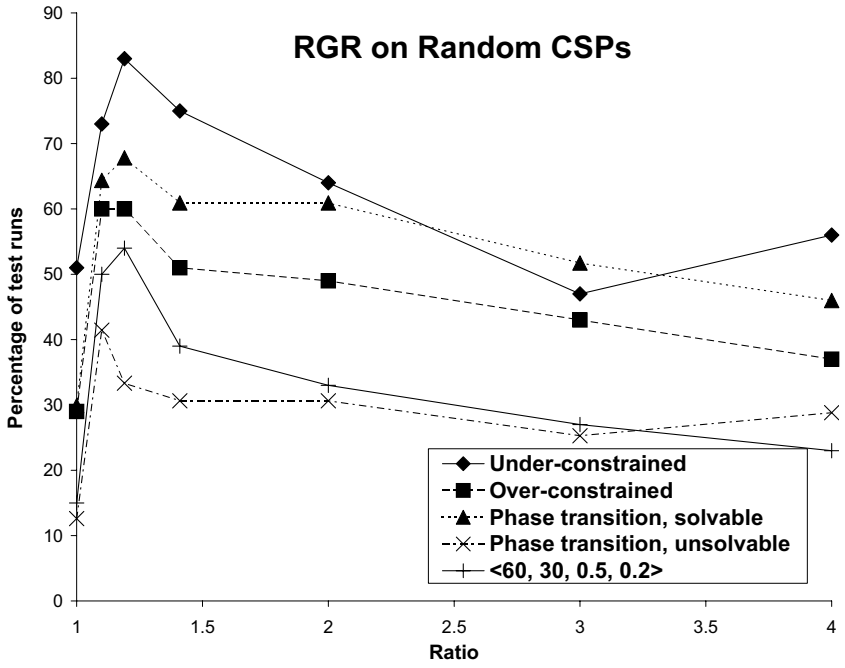

Figure 4.7: Effect of r: RGR on random CSPs.

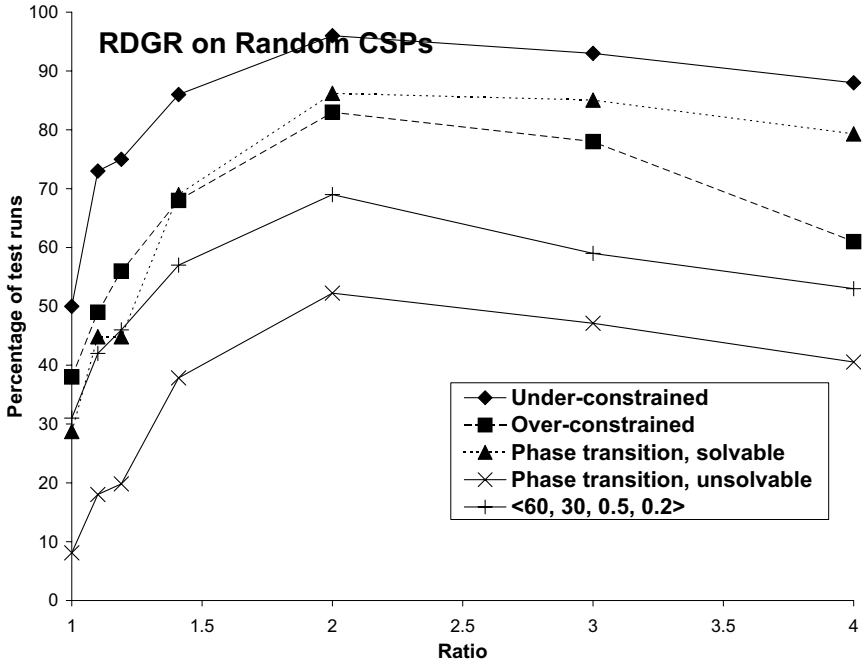

Figure 4.8: Effect of r: RDGR on random CSPs.

This discrepancy in the value of r for RDGR on random problems can be explained by the fact that the cutoff value increases more quickly in RGR than in RDGR, as shown in Figure 4.9.

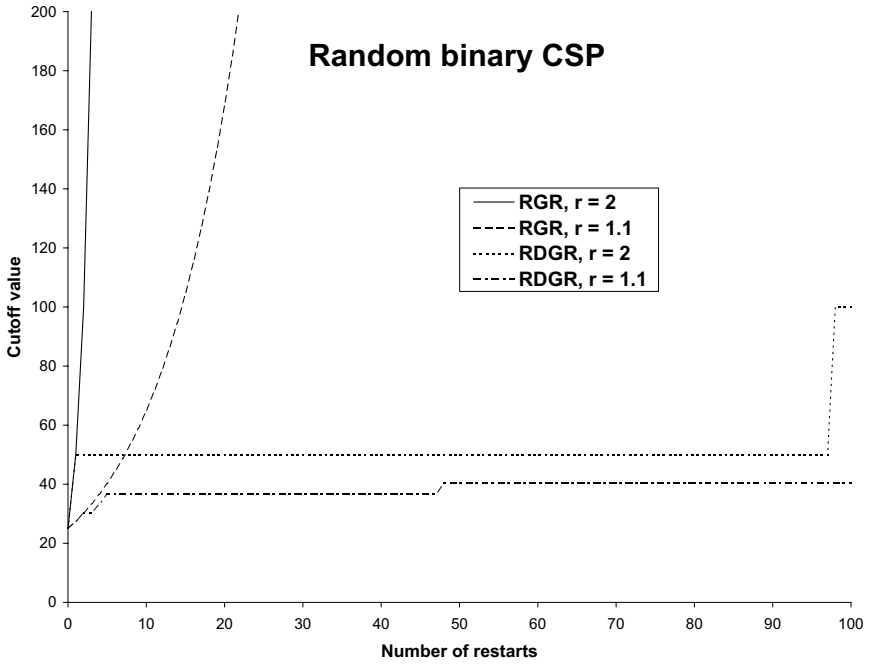

Figure 4.9: Increasing rate of the cutoff value (3 minutes).

4.5 Relative performance of BT, LS, ERA, RGR, and RDGR

In this section we compare the relative performance of all the search techniques. Each stochastic algorithm was run 500 times of 10 minutes each on the GTAAP data-set and on ensembles of instances of randomly generated problems with each instance run for 3 minutes. Figures 4.10 and 4.11 show the distributions of LS, ERA, RGR, and RDGR on data set 1 and data set 5.

In both the figures, the percentage of test runs finding solutions in the range of 0%-10% deviation from best known solution is more for RDGR than RGR. In Figure 4.10, ERA is unable to find a solution better than 25% deviation from best known solution. While in Figure 4.11, ERA finds complete solutions for all the test runs.

Figures 4.12, 4.13, 4.14, and 4.15 show the relative performance on the random problems. We do not show LS and ERA in Figure 4.13 because they go off the scale.

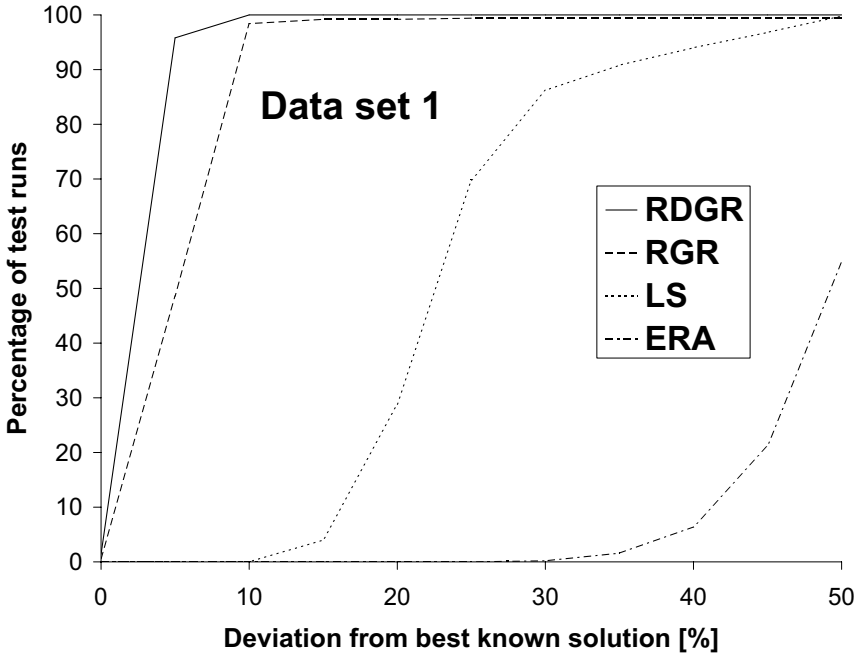

Figure 4.10: SQDs: GTAAP, over-constrained.

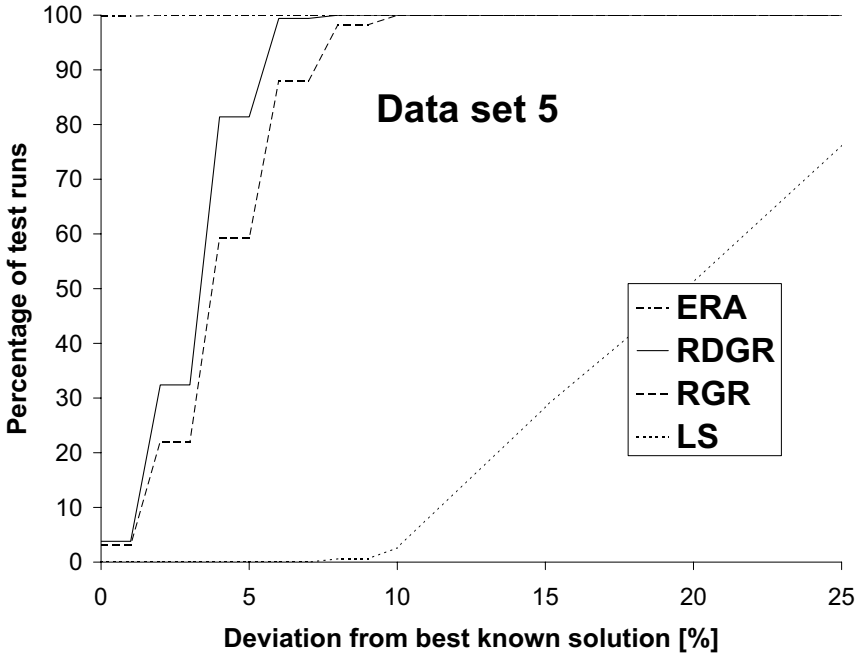

Figure 4.11: SQDs: GTAAP, solvable.

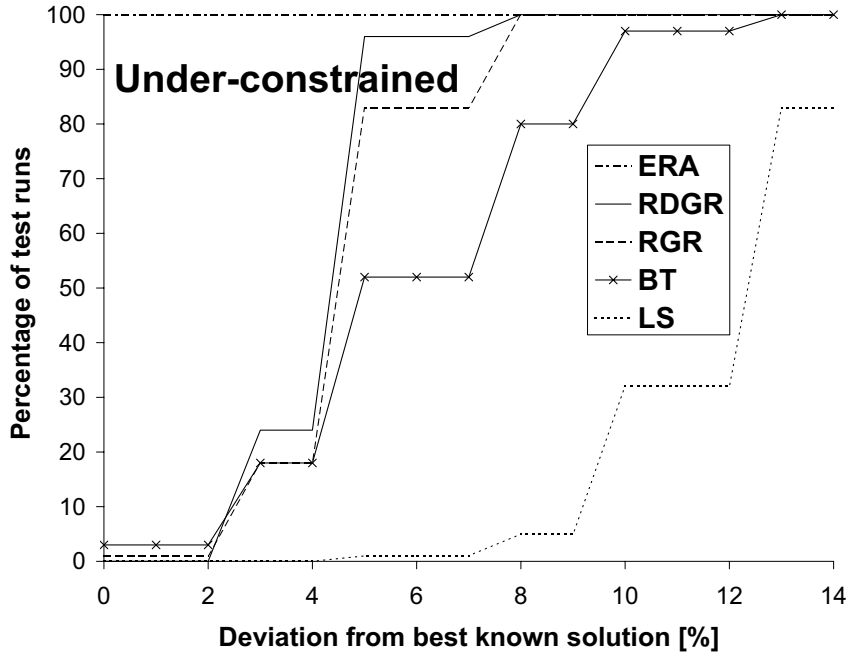

Figure 4.12: SQDs: under-constrained, random CSPs.

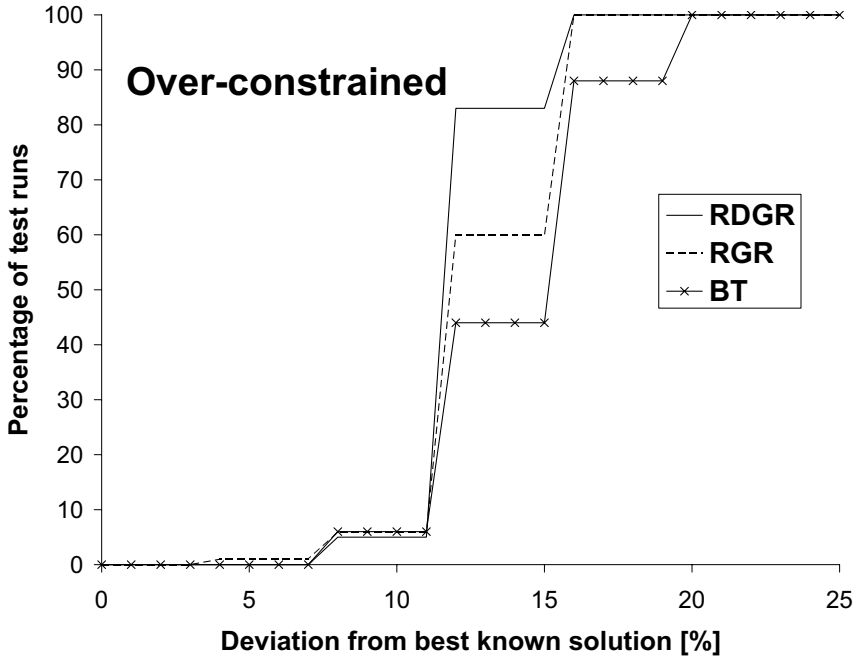

Figure 4.13: SQDs: over-constrained, random CSPs.

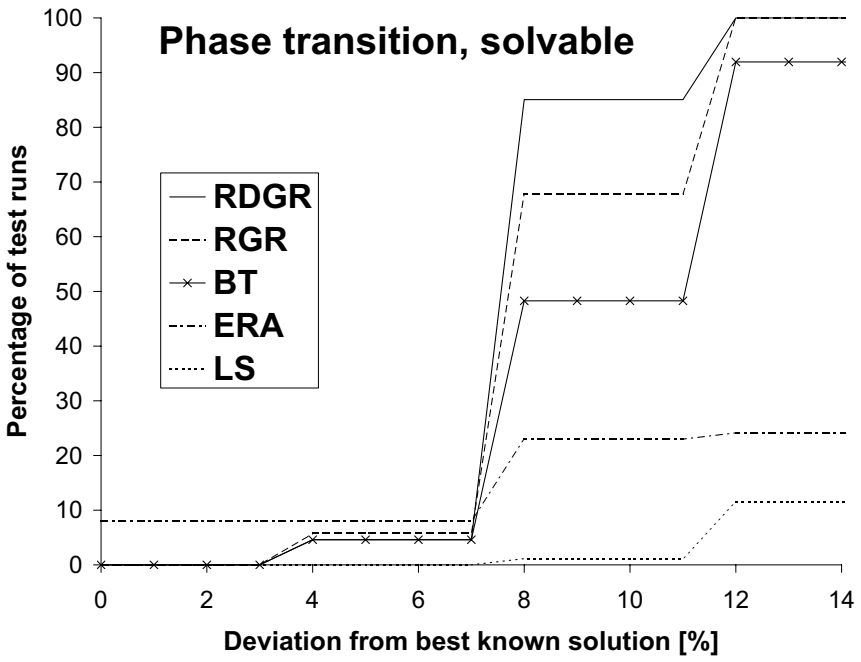

Figure 4.14: SQDs: solvable random CSPs, at phase transition.

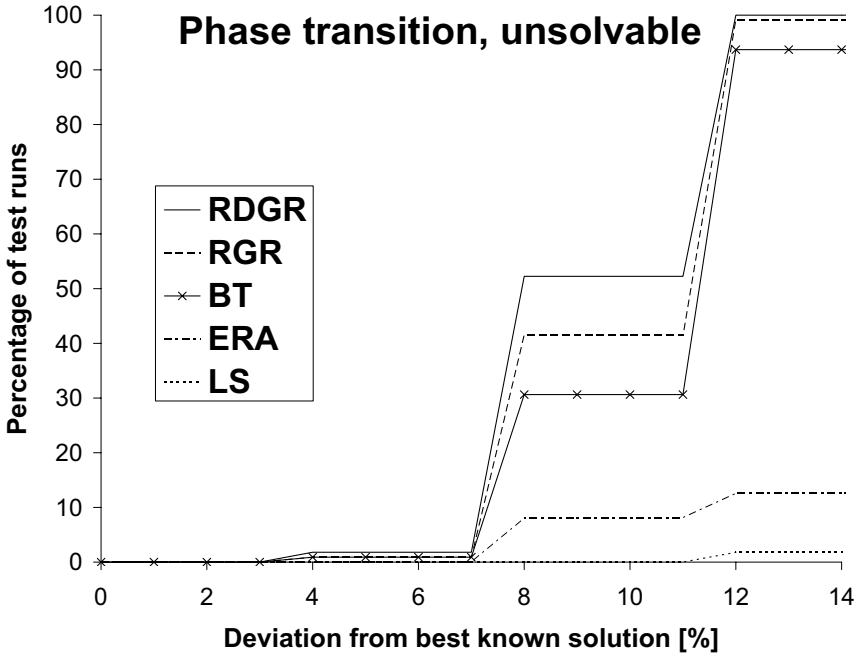

Figure 4.15: SQDs: unsolvable random CSPs, at phase transition.

Tables 4.1 and 4.2 show the lower limit and the upper limit of the confidence intervals of the performance differences of RDGR over RGR and ERA.

Table 4.1: Improvements of RDGR with 95% confidence level, GTAAP data-sets.

Solvable data sets				
Data set	Improvements over RGR		Improvements over ERA	
	Lower limit	Upper limit	Lower limit	Upper limit
2	1.53	1.61	-6.15	-6.15
5	0	1.85	-3.7	-3.7
6	1.56	1.56	-6.25	-6.25
7	4.00	4.00	-3.99	-3.99
8	2.44	4.87	0	0
Over-constrained data sets				
Data set	Improvements over RGR		Improvements over ERA	
	Lower limit	Upper limit	Lower limit	Upper limit
1	1.61	1.61	45.16	46.77
3	3.43	3.44	27.58	31.03
4	1.84	1.85	24.08	27.77

Table 4.2: Improvements of RDGR with 95% confidence level, randomly generated problems.

Data set	**Improvements over RGR**		**Improvements over ERA**	
	Lower limit	Upper limit	Lower limit	Upper limit
Under-constrained	0	2.49	-4.99	-3.75
Over-constrained	0	3.99	83.99	86.00
Phase transition, unsolvable	0	4.00	14.00	19.99
Phase transition, solvable, lower solution quality deviations	0	4.00	-4.00	-4.00
Phase transition, solvable, higher solution quality deviations	0	4.00	17.99	22.00

Comparing the improvement of RDGR over RGR, we see that the limits of the confidence intervals are positive, indicating that RDGR does improve over RGR. As for ERA, RDGR is superior to ERA for over-constrained problems (i.e., confidence intervals have positive values), and the opposite holds for under-constrained problems (i.e., confidence

intervals have negative values). These results hold for both GTAAP data sets and randomly generated problems. Note that this holds across the phase transition, where ERA remains the only technique capable of solving more of the instances as it is given more time.

4.5.1 Improvement of RDGR over BT

Tables 4.3, 4.4, and 4.5 show that the mean and maximum values of the solution sizes produced by RDGR are clearly greater than those of the solution sizes produced by BT.

Table 4.3: Statistics of solution size for data set 1 (500 runs, 10 minutes each).

Data set 1 (69 variables, over-constrained)						
Search	Mean	Median	Mode	Standard dev.	Minimum	Maximum
BT	57	57	57	0	57	57
LS	47.12	48	49	4.44	30	55
ERA	30.99	31	32	4.37	18	45
RGR	58.27	58	58	2.83	23	62
RDGR	59.66	60	60	0.77	58	62

Table 4.4: Statistics of solution size for data set 5 (500 runs, 10 minutes each).

Data set 5 (54 variables, tight but solvable)						
Search	Mean	Median	Mode	Standard dev.	Minimum	Maximum
BT	52	52	52	0	52	52
LS	42.88	44	46	3.94	29	50
ERA	53.99	54	54	0.04	53	54
RGR	51.70	52	52	1.04	49	54
RDGR	52.17	52	52	0.78	50	54

However, due to its stochastic nature, RDGR suffers from high instability (non-zero standard deviation) in its solution quality. Even on random problems (Table 4.5 and Figures 4.12, 4.13, 4.14, and 4.15) RDGR dominates BT. Also, on ensembles of problem instances, RDGR is more stable (lower standard deviation) than BT.

Table 4.5: Statistics of solution size for randomly generated problems

Under-constrained (40 variables)						
Search	Mean	Median	Mode	Standard dev.	Minimum	Maximum
BT	37.5	38	38	1.12	35	40
LS	35.2	35	35	0.80	34	38
ERA	40	40	40	0	40	40
RGR	38.0	38	38	0.61	37	40
RDGR	38.2	38	38	0.49	37	40
Over-constrained (40 variables)						
Search	Mean	Median	Mode	Standard dev.	Minimum	Maximum
BT	21.3	21	21	0.77	20	23
LS	19.6	20	20	0.61	19	21
ERA	0.7	1	0	0.89	0	4
RGR	21.6	22	22	0.60	21	24
RDGR	21.8	22	22	0.45	21	24
Phase transition, solvable (25 variables)						
Search	Mean	Median	Mode	Standard dev.	Minimum	Maximum
BT	22.4	22	23	0.71	21	24
LS	20.8	21	21	0.66	20	23
ERA	19.0	19	21	3.83	6	25
RGR	22.7	23	23	0.55	22	24
RDGR	22.8	23	23	0.43	22	24
Phase transition, unsolvable (25 variables)						
Search	Mean	Median	Mode	Standard dev.	Minimum	Maximum
BT	22.2	22	22	0.63	20	24
LS	20.4	20	20	0.50	20	22
ERA	18.2	19	21	3.48	7	23
RGR	22.4	22	22	0.53	21	24
RDGR	22.5	23	23	0.53	22	24

4.5.2 Superiority of RDGR over LS

Tables 4.3, 4.4, and 4.5 show that the mean and maximum length of the solution yielded by LS is smaller than RDGR. Clearly, the performance of RDGR is superior to that of LS (see also Figures 4.10, 4.11, 4.12, 4.14, and 4.15). Although the solution quality is variable for both RDGR (standard deviation of 0.77 in Table 4.3) and LS (standard deviation of 4.44 in Table 4.3), the low mean value of the solution quality of LS ensures that RDGR remains superior to LS.

4.5.3 Superiority of RDGR over ERA on over-constrained problems

In Figure 4.10, RDGR yields solutions that are within a range of 0-10% deviation from the best solution. While the best solution that ERA gives is more than 25% deviated from the best solution. Clearly, on over-constrained problems (Figure 4.10 and Tables 4.1 and 4.2), the livelock phenomenon prevents ERA from finding solutions of quality comparable to those found by the other techniques. BT, LS, RDGR, and RGR do not exhibit such a dichotomy of behavior between over-constrained cases and solvable instances.

4.5.4 Performance of ERA

As we just stated, on over-constrained problem (both GTAAP data and randomly generated problems), RDGR, RGR, BT, and LS are superior to ERA. On unsolvable problem instances around the phase transition (Figure 4.15), RDGR, RGR, and BT are still superior to ERA, but ERA outperforms LS.

On solvable GTAAP instances (Figure 4.11) and randomly generated under-constrained problem instances (Figure 4.12), ERA completely dominates all the other search techniques for all values of the deviation from the best known solution. However, on solvable problem instances around the phase transition (Figure 4.14), two cases must be distinguished. On

lower values of deviation from the best known solution, ERA dominates all the other search techniques. Thus, confirming that ERA is our best technique for solving solvable problems and the only one that can solve tight instances. However, for larger values of deviation from the best known solution, ERA performs only better than LS, while RDGR, RGR, and BT perform better than ERA. More tests are needed to understand this phenomenon.

4.5.5 RDGR is more stable than RGR

Table 4.6 shows the standard deviation of RGR and RDGR on the GTAAP data sets. Due to

Table 4.6: Standard deviation of RGR and RDGR on GTAAP data sets.

Data set	1	2	3	4	5	6	7	8
RGR	2.8	1.1	0.7	1.0	1.0	1.2	0.59	0.73
RDGR	0.7	0.8	0.6	0.9	0.7	1.1	0.43	0.47

their stochastic nature, RDGR and RGR techniques show variation in their solution quality. However, the smaller standard deviations of RDGR compared to RGR in Table 4.6 show that RDGR is relatively more stable than RGR.

4.5.6 Sensitivity of LS to local optima

The inability of LS to yield any good solutions shows that LS sensitivity to local optima makes it particularly unattractive. Even BT outperforms LS.

4.5.7 Larger number of restarts in RDGR

Table 4.7 shows the average number of restarts occurring in RGR and RDGR. This confirms our expectations stated in Section 4.1 that RDGR performs more restarts than RGR.

Table 4.7: Average number of restarts by RGR and RDGR on GTAAP data sets.

Data set	1	2	3	4	5	6	7	8
RGR	16.7	17.4	22.5	14.7	22.4	19.5	27.8	30.4
RDGR	74.5	59.9	167.4	39.1	39.1	46.2	826.2	272.0

Summary

We tested the 5 search strategies, namely: BT, LS, ERA, RGR, and RDGR on two types of problems. The first is a real-world resource allocation problem, GTAAP, which was the initial motivation for all our investigations. To validate the observations made on GTAAP data-sets, we conducted tests on randomly generated problems. All these experiments support that our proposed RDGR restart strategy improves upon RGR.

Chapter 5

Conclusions and future work

In this thesis we proposed an improved dynamic restart strategy (RDGR). We compared its performance with other search techniques, namely: BT, LS, ERA, and RGR. We showed in RDGR that making the restart strategy dependent on the search progress enhances the performance of randomized backtrack search compared to the static restart strategy of RGR. We studied these search techniques on a real-world application, GTAAP, and also on randomly generated binary CSPs. In this chapter we summarize our research and results, and propose directions for future investigations .

5.1 Summary of the research conducted

The basic backtrack search is deterministic, complete, and sound. However, on large problems its performance is seriously undermined by thrashing and large variance in its run-time on various instances of the same problem. Variable/value ordering heuristics and methods such as look-ahead and backjumping can improve the performance of search, but cannot eliminate thrashing completely. Zou and Choueiry [2003a; 2003; 2003b] showed the drawbacks of LS and ERA on GTAAP instances. Our tests showed that these drawbacks also seem to be present on randomly generated problems.

Given the disadvantages of BT, LS, and ERA, we studied randomized BT search with restarts. Gomes et al. demonstrated that randomization of heuristic choices combined with restart mechanisms is effective in overcoming the effects of thrashing and in reducing the total execution time of systematic BT search [1998]. RGR and RDGR are such algorithms that use randomized BT search with restarts. We proposed RDGR, an improvement over RGR of Walsh [1999].

5.2 Conclusions

By addressing a real-world application, we are able to identify, characterize, and compare the behavior of various search techniques. We tested various deterministic-ordering heuristics for BT. As expected, dynamic selection of variables was found to be superior to static selection of variables. However, there was no difference in performance among variable-ordering and value-ordering heuristics. Also, while BT is complete and sound, it suffers from thrashing. LS is vulnerable to local optima. ERA has an amazing ability to solve tight solvable problems (some of which we thought were unsolvable). However, ERA's performance degrades on over-constrained problems due to the livelock phenomenon. Randomization of backtrack search with restart strategies are effective in preventing thrashing. RGR operates by increasing the cutoff values at every restart, which makes it more increasingly vulnerable to thrashing. RDGR attenuates this effect by making the cutoff value depend upon the result obtained at the previous restart, thus increasing the number of restarts in comparison to RGR. Consequently, RDGR exhibits a more stable behavior than RGR while yielding at least as good solutions.

The following five statements, where $\succ$ denotes dominance of an algorithm over another, summarize the behavior of the 5 search strategies, also briefly explained in Table 5.1:

- On unsolvable instances:

  - Beyond the phase transition: RDGR $\succ$ RGR $\succ$ BT $\succ$ LS $\succ$ ERA.
  - Around the phase transition: RDGR $\succ$ RGR $\succ$ BT $\succ$ ERA $\succ$ LS.

- On solvable instances:
  - Beyond the phase transition: ERA $\succ$ RDGR $\succ$ RGR $\succ$ BT $\succ$ LS.
  - Around the phase transition: two cases must be distinguished (see Figure 4.14).

    If we focus on the percentage of problems solved (i.e., lower values of SQDs), ERA remains the dominant technique: ERA $\succ$ RDGR $\succ$ RGR $\succ$ BT $\succ$ LS.

    However, if we accept larger values of the deviation from the best solution, then RDGR statistically dominates: RDGR $\succ$ RGR $\succ$ BT $\succ$ ERA $\succ$ LS.

Table 5.1: Comparing the behavior of search strategies.

	Characteristics
ERA	**General:** Stochastic and incomplete
	Tight but solvable problems: Immune to local optima and solves tight CSPs
	Over-constrained problems: Livelock causes instability and yields shorter solutions
LS	**General:** Stochastic, incomplete, and quickly stabilizes
	Tight but solvable problems: Liable to local optima, and fails to solve tight CSPs even with random-walk and restart strategies
	Over-constrained problems: Finds longer solutions than ERA
RDGR	**General:** Stochastic, incomplete, immune to thrashing, produces longer solutions than BT, immune to livelock, reliable on unknown instances, and immune to local optima, but less than ERA
RGR	**General:** Stochastic, approximately complete, less immune to thrashing than RDGR, and yields shorter solutions than RDGR in general.
BT	**General:** Systematic, complete (theoretically, rarely in practice), liable to thrashing, yields shorter solutions than RDGR and RGR, stable behavior, and more stable solutions than stochastic methods in general

5.3 Open questions and future research directions

Our research was motivated and enabled by the GTA assignment project. However, we extended our results beyond this particular application to randomly generated problems. Below we describe the future research directions:

1. Enhance RDGR with tabu behavior across restarts.

2. Validate our findings on other real-world case-studies.

3. Design ‘progress-aware’ restart strategies, that is, strategies that can decide, *during* a given restart, whether to continue or abandon this particular execution.

4. Design new search hybrids where a solution from a given technique such as ERA is fed as a seed to another one such as heuristic backtrack search.

Appendix A

Results from the GTAAP data sets

This chapter presents results of all the experiments done on the data sets of GTAAP. Section A.1 contains the best solutions obtained by the five search techniques (i.e., BT, LS, ERA, RGR, and RDGR). Section A.2 presents the results of testing the various deterministic ordering heuristics of BT. Section A.3 presents the SQD graphs of comparing the four stochastic search techniques (i.e., LS, ERA, RGR, and RDGR). Section A.4 presents the SQD graphs of the effect of running time on RGR and RDGR. Finally, Section A.5 presents the SQD graphs for the different values of the ratio used to increase the cutoff value for both RGR and RDGR.

A.1 Best results of the GTAAP data sets

This section contains the best solutions obtained for the GTAAP data sets. All the experiments were carried out for 5 minutes. Table A.1 shows the best results obtained by BT using the 12 different deterministic ordering heuristics. We note the number of unassigned courses (i.e. unassigned variables), the solution quality in terms of the geometric mean of the assigned GTAs preferences, the number of unused GTAs, the available resources, time taken for finding the best solution, the number of constraint checks needed to find the best solution, and the combination of variable-value ordering heuristic that obtained the best solution for each GTAAP data-set.

Table A.1: Best results obtained by BT using deterministic various ordering heuristics: GTAAP (5 minutes).

Data Sets	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	Time [sec]	CC ($\times 10^8$)	Ordering Heuristic
1	12	2.31	0	4.5	94	1.39	DDD-OCCURRENCE
2	2	2.65	0	2.7	22	0.92	DLD-OCCURRENCE
3	3	3.73	0	1.5	2	0.04	DLD-PREFERENCE
4	7	2.86	0	3.5	162	2.27	DDD-FIL
5	0	3.22	0	1.5	25	0.75	DLD-OCCURRENCE
6	2	4.15	0	2.3	25	1.07	DLD-PREFERENCE
7	6	2.88	0	4.2	253	0.92	DDD-FIL
8	1	3.89	3	4.6	102	0.50	DLD-FIL

Tables A.2 and A.3 show the best solutions obtained by LS, ERA, RGR, and RDGR. Each experiment was run 500 times, with each run for 5 minutes. We note the number of unassigned courses (i.e. unassigned variables), the solution quality in terms of the geometric mean of the assigned GTAs preferences, the number of unused GTAs, and the available resources.

Table A.2: Best results of LS and ERA: GTAAP (500 runs, 5 minutes each).

Data Set	LS				ERA			
	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource
1	14	3.22	0	3.7	23	2.84	7	14.4
2	4	2.98	1	5.5	0	2.99	0	1.6
3	4	3.61	0	2.0	8	3.22	1	2.0
4	7	3.74	0	4.3	9	3.01	2	4.5
5	3	3.64	2	11.5	0	2.82	2	7.8
6	9	3.64	0	7.2	0	3.28	0	1.5
7	3	2.70	4	4.66	0	3.47	0	1.7
8	7	4.02	12	14.5	0	4.41	5	4.2

Table A.3: Best results of RGR and RDGR: GTAAP (500 runs, 5 minutes each).

Data Set	RGR				RDGR			
	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource
1	7	2.68	0	0	8	2.10	0	0.6
2	1	2.81	0	2.6	2	2.96	1	3.6
3	2	3.62	0	0	2	3.40	0	0
4	5	2.85	0	1.0	4	3.05	0	1.0
5	0	3.30	1	1.5	0	2.72	0	1.5
6	1	2.70	1	3.5	1	2.52	0	3.5
7	0	2.94	2	2.5	0	2.75	0	1.7
8	0	2.99	5	3.6	0	3.01	7	3.68

A.2 Results obtained from BT using the various deterministic ordering heuristics

This section presents the results of testing the various deterministic ordering heuristics of BT on all the data sets (i.e. 12 experiments for each data set, 96 experiments all together). Each experiment was run for 5 minutes. And, for each experiment we note the number of unassigned courses, the solution quality in terms of the geometric mean of the assigned GTAs preferences, the number of unused GTAs, the available resources, time taken for finding the best solution, the number of constrains checks needed to find the best solution and during 5 minutes, the number of nodes visited to find the best solution and during 5 minutes, the depth of the search tree (i.e., number of variables), the shallowest level reached by backtracking, and the number of backtracks for the best solution and in 5 minutes.

Table A.4: Deterministic ordering heuristics of BT: GTAAP, data set 1.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	17	2.84	0	2.5	291	3.98	4.08	67.36	69.37	69	52	44.95	46.28
SLD-PREFERENCE	16	4.28	0	3.6	136	2.02	3.93	33.19	74.71	69	51	21.35	47.78
SLD-OCCURRENCE	17	2.46	0	2.5	40	0.92	3.82	8.16	73.27	69	53	4.74	44.44
SDD-FIL	17	2.84	0	2.5	293	3.98	4.05	67.36	68.88	69	52	44.95	45.94
SDD-PREFERENCE	16	4.28	0	3.6	137	2.02	3.93	33.19	74.78	69	51	21.35	47.82
SDD-OCCURRENCE	17	2.46	0	2.5	41	0.90	3.58	8.16	71.14	69	53	4.74	43.10
DLD-FIL	15	2.46	0	4.3	230	2.59	3.19	65.57	87.14	69	55	34.79	46.34
DLD-PREFERENCE	16	3.90	0	5.0	205	2.44	3.34	55.14	81.92	69	55	29.36	43.62
DLD-OCCURRENCE	18	2.44	0	4.6	16	0.73	3.25	0.40	88.61	69	56	0.17	47.22
DDD-FIL	16	2.31	0	5.3	39	0.90	3.76	7.67	81.64	69	51	3.89	51.00
DDD-PREFERENCE	13	3.64	0	4.7	24	0.76	3.31	2.95	79.50	69	54	1.52	42.39
DDD-OCCURRENCE	12	2.31	0	4.5	94	1.39	3.22	21.19	79.85	69	53	11.12	42.32

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.5: Deterministic ordering heuristics of BT: GTAAP, data set 2.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	8	2.91	1	6.2	35	0.95	4.26	3.83	40.93	65	48	2.31	24.29
SLD-PREFERENCE	10	3.55	3	9.1	13	0.65	3.46	0.06	84.81	65	51	0	48.92
SLD-OCCURRENCE	10	2.76	0	7.5	15	0.74	3.70	0.06	56.98	65	54	0	29.69
SDD-FIL	8	2.91	1	6.2	36	0.95	4.22	3.83	40.98	65	48	2.31	24.32
SDD-PREFERENCE	10	3.55	3	9.1	13	0.65	3.45	0.06	84.65	65	51	0	48.83
SDD-OCCURRENCE	10	2.76	0	7.5	15	0.74	3.69	0.06	56.87	65	54	0	29.63
DLD-FIL	7	2.96	1	4.1	20	0.86	3.52	0.06	42.49	65	58	0	12.16
DLD-PREFERENCE	5	3.11	0	5.7	21	0.90	3.41	0.06	53.85	65	56	0	26.05
DLD-OCCURRENCE	2	2.65	0	2.7	22	0.92	3.63	0.12	43.25	65	56	0.02	18.90
DDD-FIL	6	3.08	0	3.9	52	1.18	3.87	6.59	41.08	65	56	2.84	18.54
DDD-PREFERENCE	7	3.41	0	4.9	20	0.87	3.66	0.06	46.16	65	57	0	20.78
DDD-OCCURRENCE	6	2.99	0	6.7	273	3.13	3.37	47.22	52.58	65	55	22.23	24.93

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.6: Deterministic ordering heuristics of BT: GTAAP, data set 3.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	7	3.32	0	4.5	244	3.34	4.10	36.37	43.77	31	9	22.37	26.76
SLD-PREFERENCE	6	4.10	0	2.3	45	0.64	4.47	4.70	26.47	31	8	2.74	14.98
SLD-OCCURRENCE	6	3.56	0	4.0	94	1.38	4.47	8.73	26.32	31	9	4.89	14.56
SDD-FIL	7	3.32	0	4.5	235	3.18	4.07	36.37	44.92	31	9	22.37	27.41
SDD-PREFERENCE	6	4.10	0	2.3	45	0.64	4.47	4.70	26.45	31	8	2.74	14.97
SDD-OCCURRENCE	6	3.56	0	4.0	93	1.38	4.50	8.73	26.45	31	9	4.89	14.63
DLD-FIL	5	3.37	0	2.5	11	0.14	3.99	1.82	38.66	31	13	0.99	21.70
DLD-PREFERENCE	3	3.73	0	1.5	2	0.04	4.04	0.12	35.77	31	11	0.05	20.85
DLD-OCCURRENCE	6	3.68	0	4.0	88	1.20	4.24	10.33	33.61	31	11	5.83	19.21
DDD-FIL	5	3.37	0	2.5	136	1.73	3.92	22.38	44.55	31	13	12.70	25.41
DDD-PREFERENCE	3	3.73	0	1.5	45	0.57	4.01	7.55	36.05	31	12	4.24	20.63
DDD-OCCURRENCE	6	3.75	0	4	90	0.79	2.63	8.38	28.45	31	13	4.68	16.15

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.7: Deterministic ordering heuristics of BT: GTAAP, data set 4.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	11	2.61	1	6.5	191	3.06	4.73	30.96	44.93	59	41	17.86	25.68
SLD-PREFERENCE	11	4.15	0	3.3	6	0.34	4.24	0.05	62.24	59	42	0	36.42
SLD-OCCURRENCE	11	2.90	0	5.3	8	0.38	4.18	0.12	57.31	59	43	0.03	32.18
SDD-FIL	11	2.61	1	6.5	198	3.06	4.59	30.96	43.85	59	41	17.86	25.08
SDD-PREFERENCE	11	4.15	0	3.3	6	0.34	4.14	0.05	60.18	59	42	0	35.21
SDD-OCCURRENCE	11	2.90	0	5.3	8	0.38	4.03	0.12	55.01	59	43	0.03	30.91
DLD-FIL	8	2.91	0	6.0	271	2.82	3.08	70.92	78.48	59	47	37.32	41.28
DLD-PREFERENCE	8	3.7	0	5.0	98	1.11	2.61	30.16	99.72	59	48	12.58	40.80
DLD-OCCURRENCE	9	3.20	0	4.7	32	0.65	3.17	6.94	73.50	59	48	3.23	38.10
DDD-FIL	7	2.86	0	3.5	162	2.27	3.73	26.29	53.98	59	47	13.92	28.26
DDD-PREFERENCE	11	3.78	0	7.7	56	0.82	3.18	14.41	80.32	59	47	7.09	40.53
DDD-OCCURRENCE	10	2.73	0	6.75	16	0.52	3.39	2.07	74.62	59	48	1.13	41.06

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.8: Deterministic ordering heuristics of BT: GTAAP, data set 5.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	13	2.85	3	9.2	230	2.56	3.26	6.42	8.14	54	39	2.44	3.19
SLD-PREFERENCE	14	4.04	2	8.5	17	0.51	4.29	0.96	32.12	54	41	0.47	14.63
SLD-OCCURRENCE	10	3.49	1	8.2	270	4.38	4.85	13.08	14.00	54	38	5.56	5.98
SDD-FIL	13	2.85	3	9.2	146	2.53	5.02	6.42	12.54	54	38	2.44	5.08
SDD-PREFERENCE	14	4.04	2	8.5	17	0.51	4.26	0.96	32.44	54	41	0.47	14.76
SDD-OCCURRENCE	10	3.49	1	8.2	271	4.41	4.87	13.08	13.96	54	38	5.56	5.96
DLD-FIL	3	3.22	0	3.5	278	3.54	3.82	15.18	16.38	54	44	6.29	6.81
DLD-PREFERENCE	4	3.82	2	4.2	23	0.70	4.46	0.30	10.73	54	46	0.08	3.49
DLD-OCCURRENCE	0	3.22	0	1.5	25	0.75	4.39	0.29	8.96	54	43	0.11	3.32
DDD-FIL	5	2.86	0	4.8	290	4.33	4.45	16.10	16.90	54	44	5.60	5.93
DDD-PREFERENCE	1	3.62	0	2.5	87	1.56	4.40	2.98	10.67	54	42	1.17	4.21
DDD-OCCURRENCE	2	3.33	0	3.5	42	0.95	3.70	1.64	34.91	54	44	0.68	18.89

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.9: Deterministic ordering heuristics of BT: GTAAP, data set 6.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	15	2.55	3	10.5	75	1.77	5.42	5.30	21.79	64	49	2.54	10.73
SLD-PREFERENCE	13	4.36	2	8.1	92	1.67	3.98	15.43	61.64	64	49	7.98	31.84
SLD-OCCURRENCE	18	3.24	3	12.1	189	3.99	6.02	5.20	8.08	64	46	2.05	3.22
SDD-FIL	15	2.55	3	10.5	76	1.79	5.44	5.30	21.44	64	49	2.54	10.57
SDD-PREFERENCE	13	4.36	2	8.13	93	1.70	4.01	15.43	60.93	64	49	7.98	31.46
SDD-OCCURRENCE	18	3.24	3	12.1	192	4.00	5.97	5.20	8.00	64	46	2.05	3.18
DLD-FIL	6	3.08	0	4.17	296	5.10	5.14	8.07	8.23	64	53	2.84	2.90
DLD-PREFERENCE	2	4.15	0	2.7	25	1.07	4.64	0.08	13.92	64	55	0.00	5.28
DLD-OCCURRENCE	10	3.21	0	5.5	117	2.39	5.13	3.94	9.39	64	54	1.43	3.29
DDD-FIL	6	2.92	0	4.2	191	3.44	4.93	10.24	17.97	64	50	4.67	8.45
DDD-PREFERENCE	6	3.78	0	4.2	24	1.09	5.13	0.13	8.76	64	55	0.03	3.42
DDD-OCCURRENCE	5	2.81	0	3.9	294	4.86	4.93	11.35	11.68	64	54	3.60	3.73

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.10: Deterministic ordering heuristics of BT: GTAAP, data set 7.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	8	2.87	1	2.76	95	0.36	1.29	79.19	225.05	25	4	42.94	126.14
SLD-PREFERENCE	7	2.92	1	3.26	67	0.26	1.37	52.06	207.56	25	4	28.26	117.09
SLD-OCCURRENCE	7	2.93	0	3.26	215	0.86	1.24	181.30	241.60	25	5	107.47	142.26
SDD-FIL	8	2.87	1	2.76	94	0.36	1.31	79.19	227.22	25	4	42.94	127.46
SDD-PREFERENCE	7	2.92	1	3.26	65	0.26	1.38	52.06	209.04	25	4	28.26	117.93
SDD-OCCURRENCE	7	2.93	0	3.26	216	0.86	1.24	181.30	241.11	25	5	107.47	141.98
DLD-FIL	6	2.88	0	4.2	281	1.01	1.05	258.10	278.54	25	7	151.53	162.80
DLD-PREFERENCE	6	2.91	0	4.2	284	1.00	1.04	261.42	277.86	25	7	153.27	162.30
DLD-OCCURRENCE	6	2.93	0	4.2	1	0.00	0.98	0.00	265.72	25	9	0	142.94
DDD-FIL	6	2.88	0	4.2	253	0.92	1.08	214.52	251.65	25	7	125.32	144.80
DDD-PREFERENCE	6	2.91	0	4.2	255	0.92	1.07	217.65	253.10	25	7	126.98	145.58
DDD-OCCURRENCE	6	2.93	0	4.2	1	0.00	1.08	0.47	229.53	25	9	0.21	121.03

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

Table A.11: Deterministic ordering heuristics of BT: GTAAP, data set 8.

Ordering heuristics	Unassigned Courses	Solution Quality	Unused GTAs	Available Resources	Time [sec]	CC ($\times 10^8$) best solution	CC ($\times 10^8$) 5 mins	NV ($\times 10^3$) best solution	NV ($\times 10^3$) 5 mins	Depth of search tree	Shallowest BT level	BT ($\times 10^3$) best solution	BT ($\times 10^3$) 5 mins
SLD-FIL	13	3.58	8	10.6	215	0.96	1.44	89.40	116.51	41	21	25.76	34.56
SLD-PREFERENCE	9	4.15	6	8.17	67	0.15	0.89	40.93	163.44	41	22	11.55	49.83
SLD-OCCURRENCE	13	3.26	6	10.68	198	0.78	1.24	83.82	122.45	41	24	21.22	32.08
SDD-FIL	13	3.58	8	10.67	218	0.96	1.41	89.40	114.82	41	21	25.76	34.02
SDD-PREFERENCE	9	4.15	6	8.17	69	0.15	0.88	40.93	160.39	41	22	11.55	48.95
SDD-OCCURRENCE	13	3.26	6	10.68	199	0.78	1.23	83.82	121.85	41	24	21.22	31.90
DLD-FIL	1	3.89	3	4.68	102	0.50	1.64	33.17	78.88	41	19	10.00	28.55
DLD-PREFERENCE	3	4.14	5	5.68	128	0.54	1.46	47.47	94.78	41	21	13.06	29.60
DLD-OCCURRENCE	5	3.11	4	6.68	280	1.12	1.20	112.22	119.31	41	30	31.59	33.70
DDD-FIL	5	3.44	5	7.18	296	1.36	1.38	95.96	97.17	41	25	22.70	23.02
DDD-PREFERENCE	4	4.17	5	5.68	260	1.24	1.40	82.05	96.39	41	25	21.94	25.99
DDD-OCCURRENCE	5	3.30	4	6.68	288	1.18	1.22	109.43	114.09	41	30	29.59	30.75

Time: CPU run time needed to reach best solution.
CC: Number of constraint checks.
NV: Number of nodes visited.
BT: Number of backtracks.

A.3 SQDs of LS, ERA, RGR, and RDGR

This section presents the SQDs of the four stochastic search techniques (i.e., LS, ERA, RGR, and RDGR) for all the data sets of GTAAP.

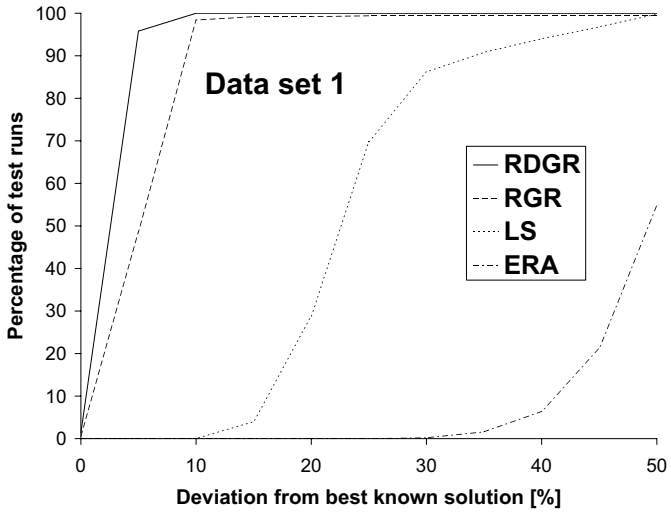

Figure A.1: SQDs: GTAAP, data set 1 (unsolvable, 500 runs, 10 minutes each).

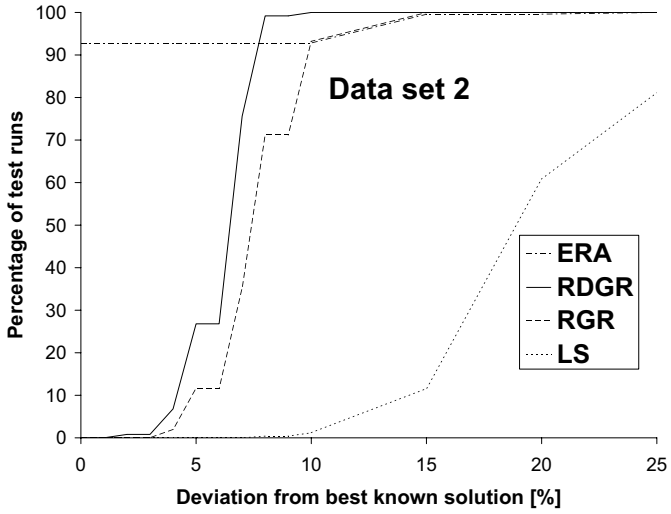

Figure A.2: SQDs: GTAAP, data set 2 (solvable, 500 runs, 10 minutes each).

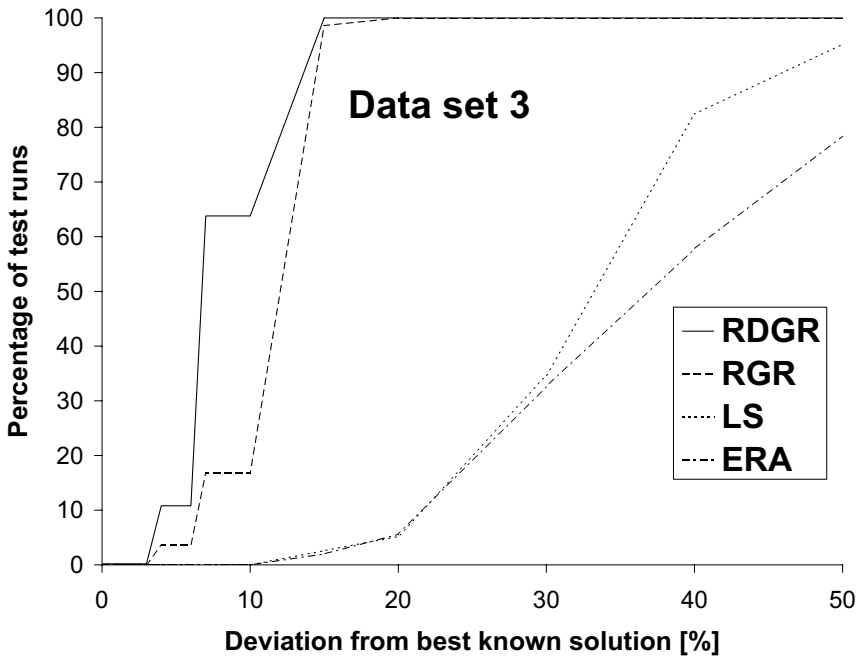

Figure A.3: SQDs: GTAAP, data set 3 (unsolvable, 500 runs, 10 minutes each).

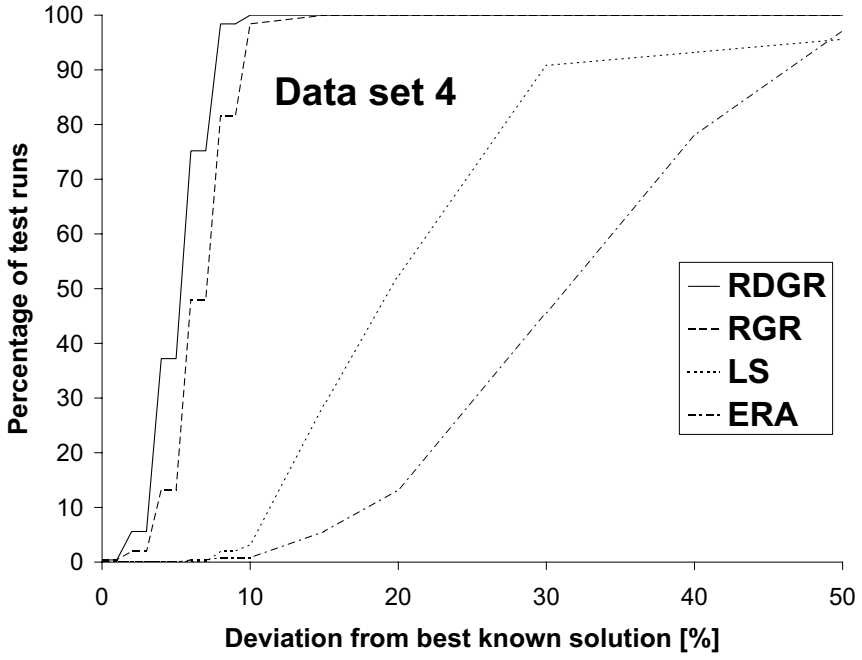

Figure A.4: SQDs: GTAAP, data set 4 (unsolvable, 500 runs, 10 minutes each).

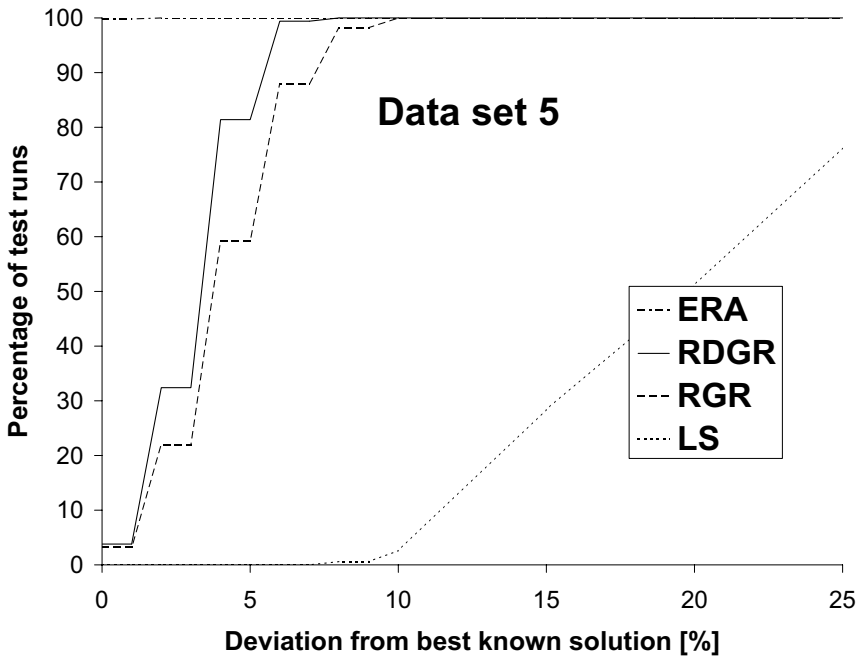

Figure A.5: SQDs: GTAAP, data set 5 (solvable, 500 runs, 10 minutes each).

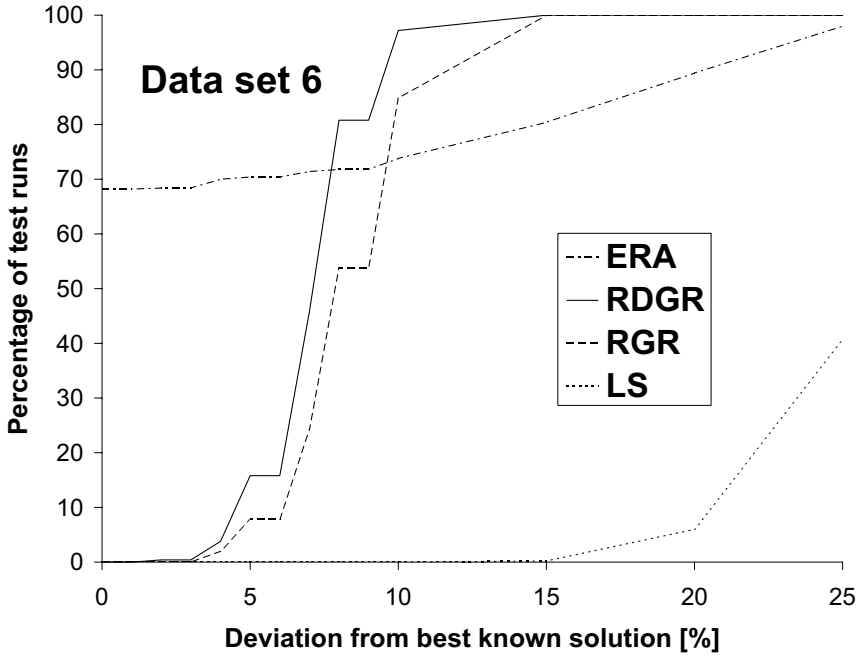

Figure A.6: SQDs: GTAAP, data set 6 (solvable, 500 runs, 10 minutes each).

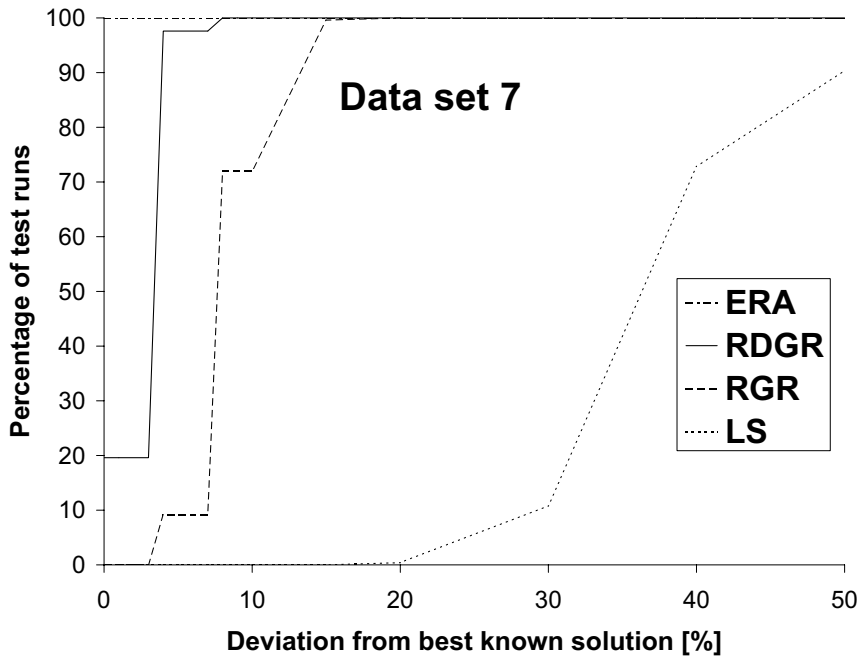

Figure A.7: SQDs: GTAAP, data set 7 (solvable, 500 runs, 10 minutes each).

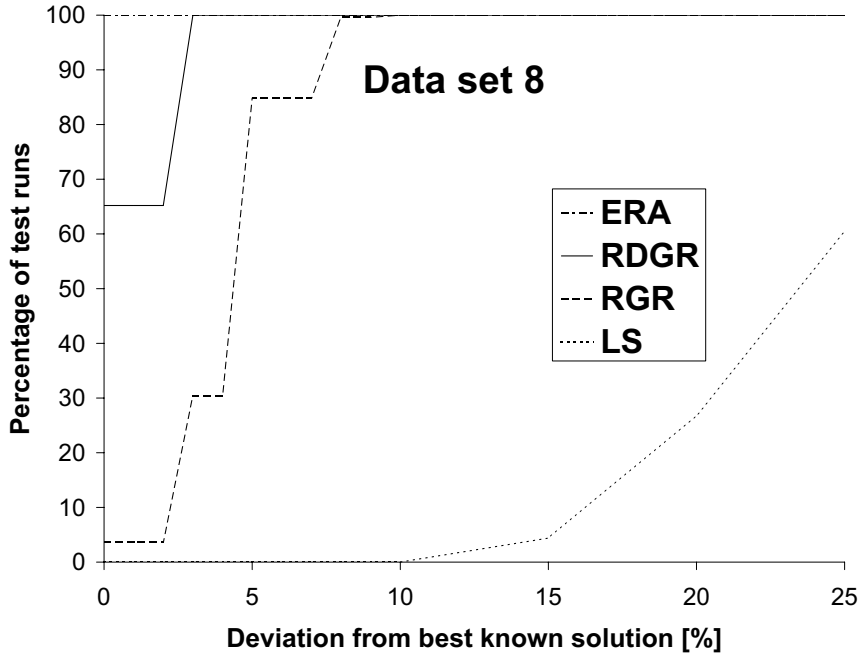

Figure A.8: SQDs: GTAAP, data set 8 (solvable, 500 runs, 10 minutes each).

A.4 RGR and RDGR over varying run time

This section presents the SQDs of comparing RGR and RDGR over different periods of times for all the data sets of GTAAP.

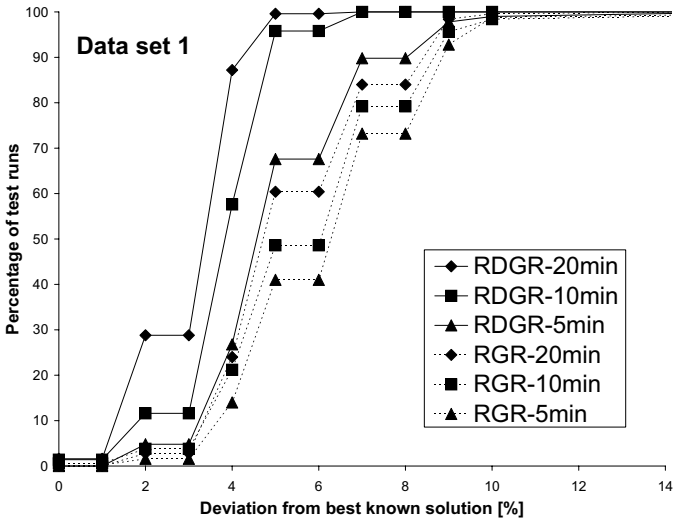

Figure A.9: Varying run time: GTAAP, data set 1 (unsolvable, 500 runs).

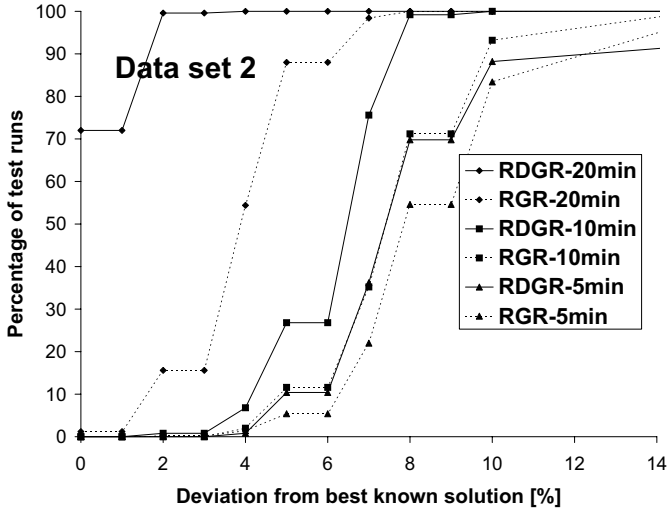

Figure A.10: Varying run time: GTAAP, data set 2 (solvable, 500 runs).

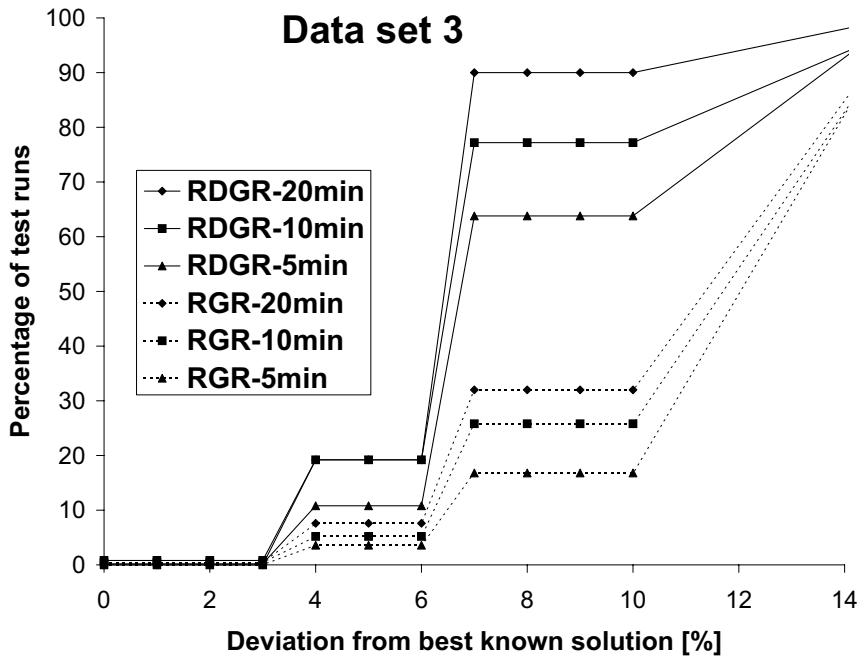

Figure A.11: Varying run time: GTAAP, data set 3 (unsolvable, 500 runs).

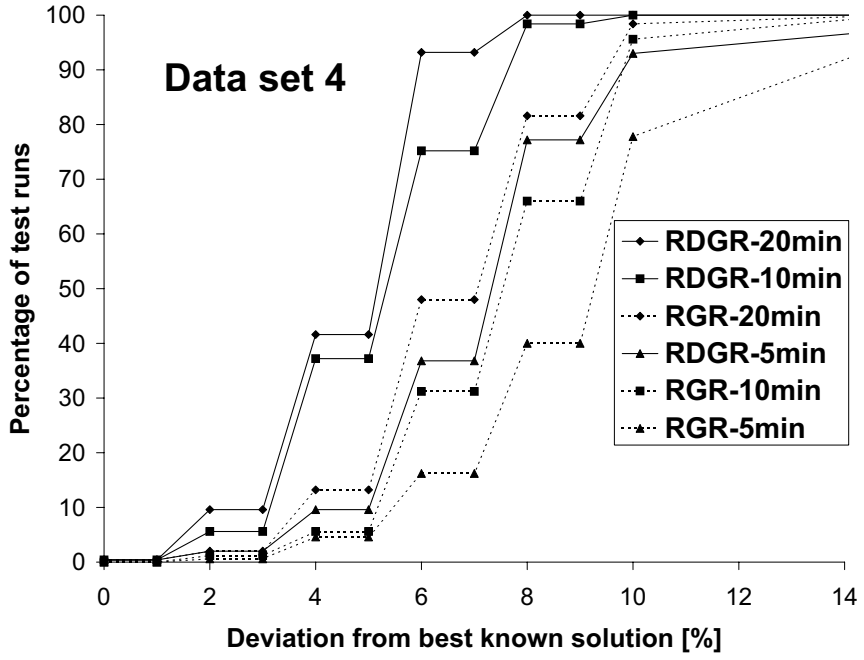

Figure A.12: Varying run time: GTAAP, data set 4 (unsolvable, 500 runs).

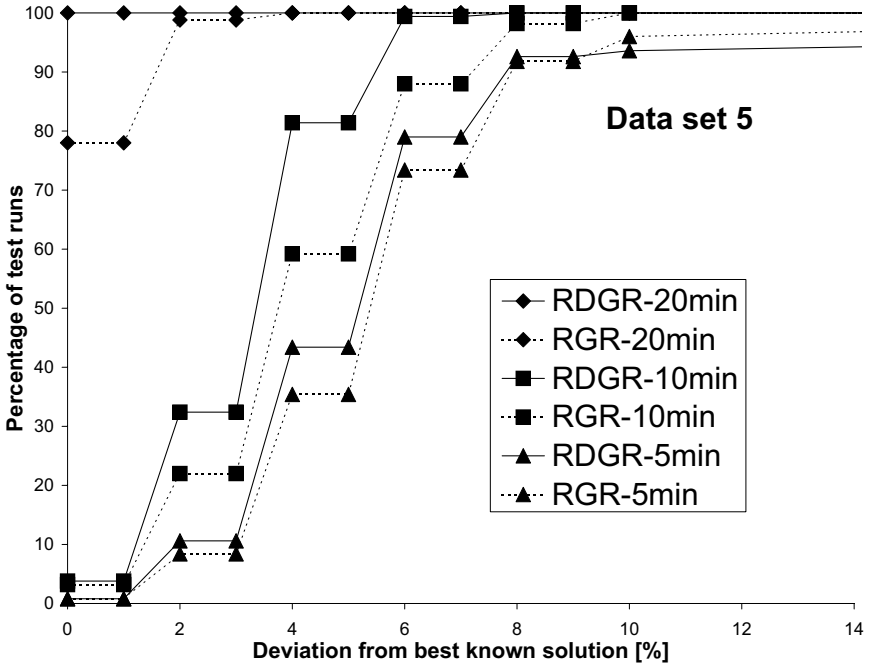

Figure A.13: Varying run time: GTAAP, data set 5 (solvable, 500 runs).

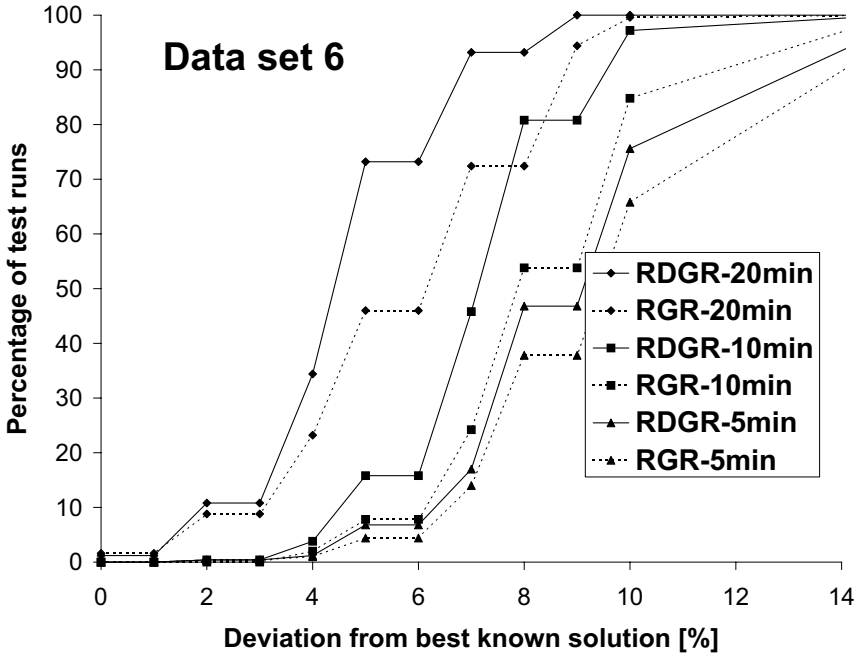

Figure A.14: Varying run time: GTAAP, data set 6 (solvable, 500 runs).

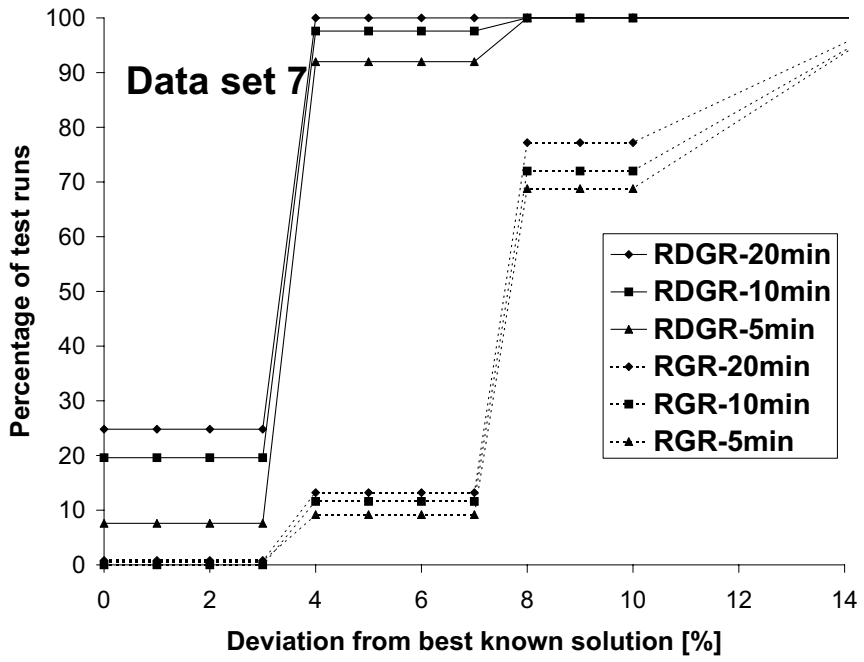

Figure A.15: Varying run time: GTAAP, data set 7 (solvable, 500 runs).

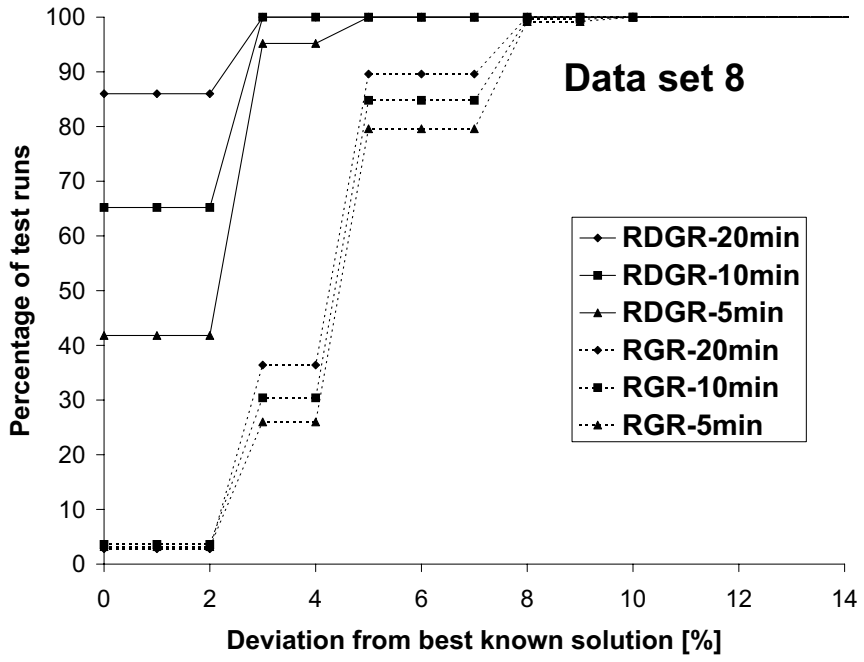

Figure A.16: Varying run time: GTAAP, data set 8 (solvable, 500 runs).

A.5 Effect of r on RGR and RDGR

This section presents the SQDs of comparing different values of the ratio used to increase the cutoff value for all the data sets of GTAAP.

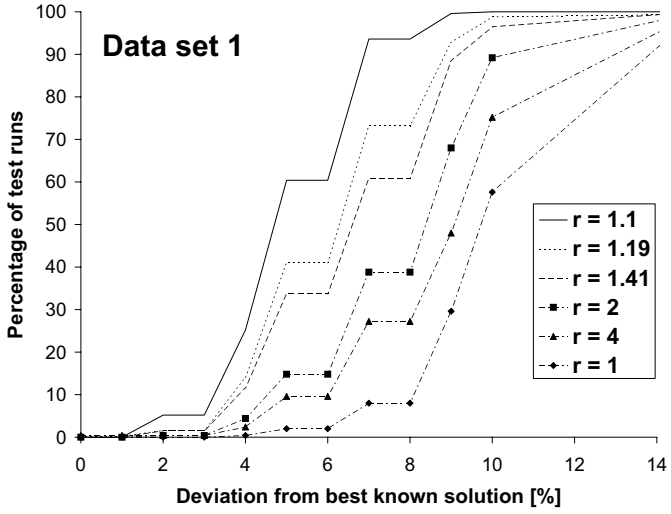

Figure A.17: Effect of r: RGR on GTAAP, data set 1 (unsolvable, 500 runs).

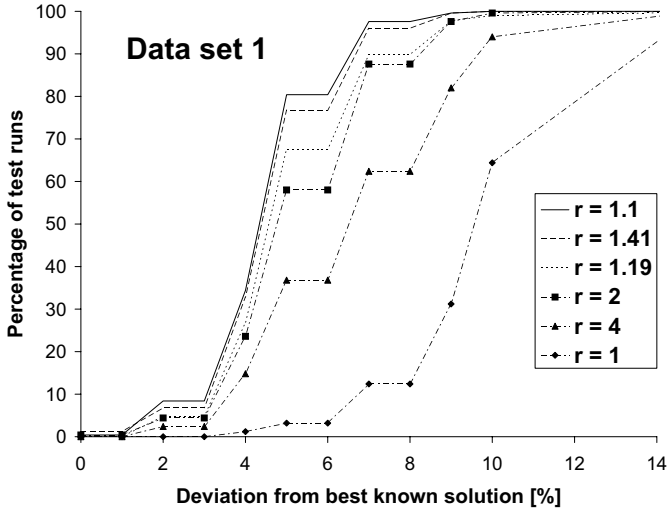

Figure A.18: Effect of r: RDGR on GTAAP, data set 1 (unsolvable, 500 runs).

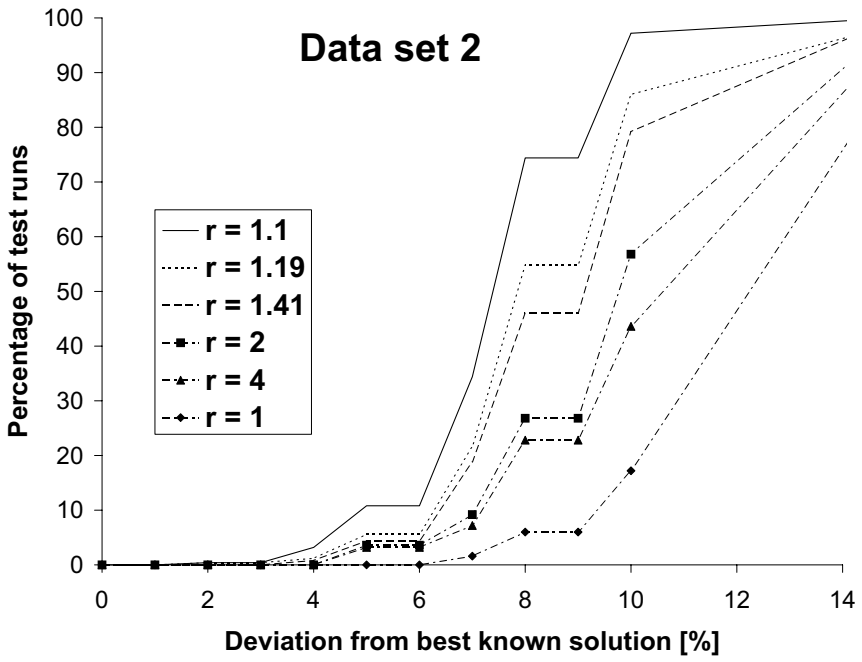

Figure A.19: Effect of r: RGR on GTAAP, data set 2 (solvable, 500 runs).

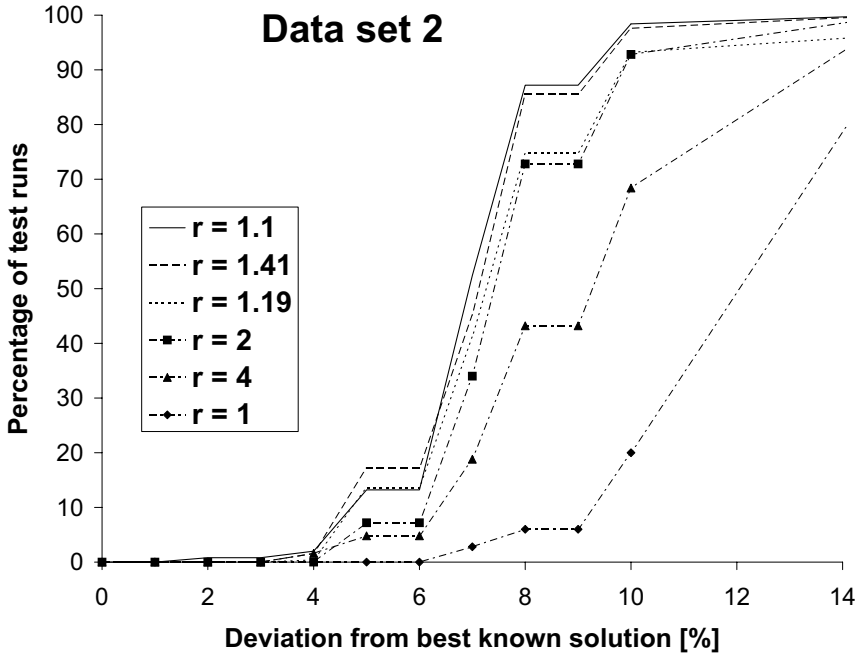

Figure A.20: Effect of r: RDGR on GTAAP, data set 2 (solvable, 500 runs).

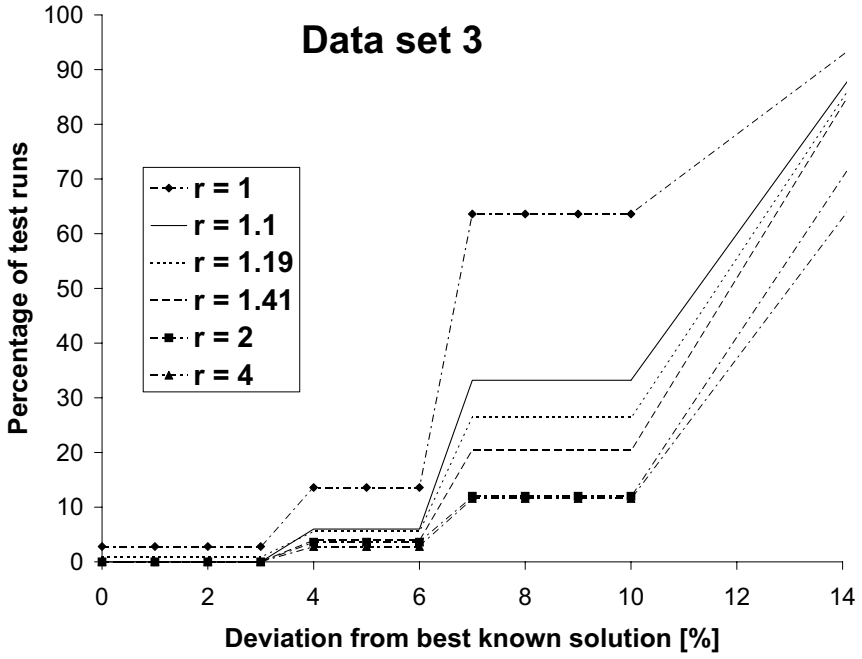

Figure A.21: Effect of r: RGR on GTAAP, data set 3 (unsolvable, 500 runs).

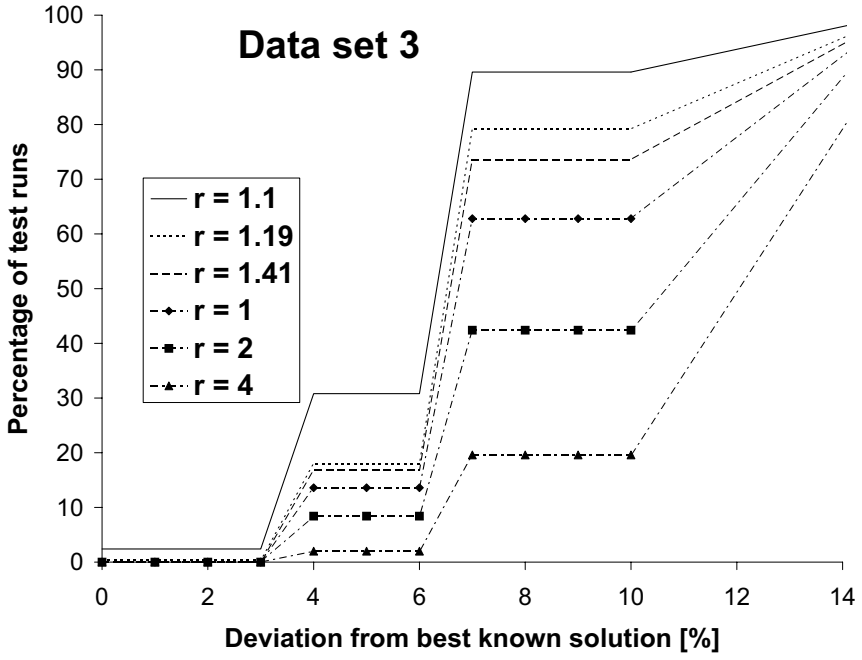

Figure A.22: Effect of r: RDGR on GTAAP, data set 3 (unsolvable, 500 runs).

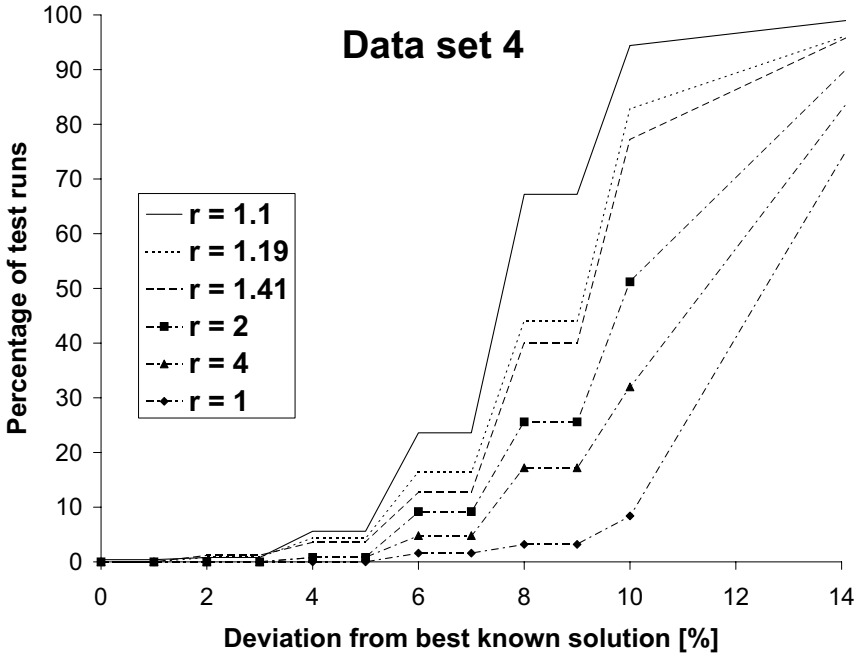

Figure A.23: Effect of r: RGR on GTAAP, data set 4 (unsolvable, 500 runs).

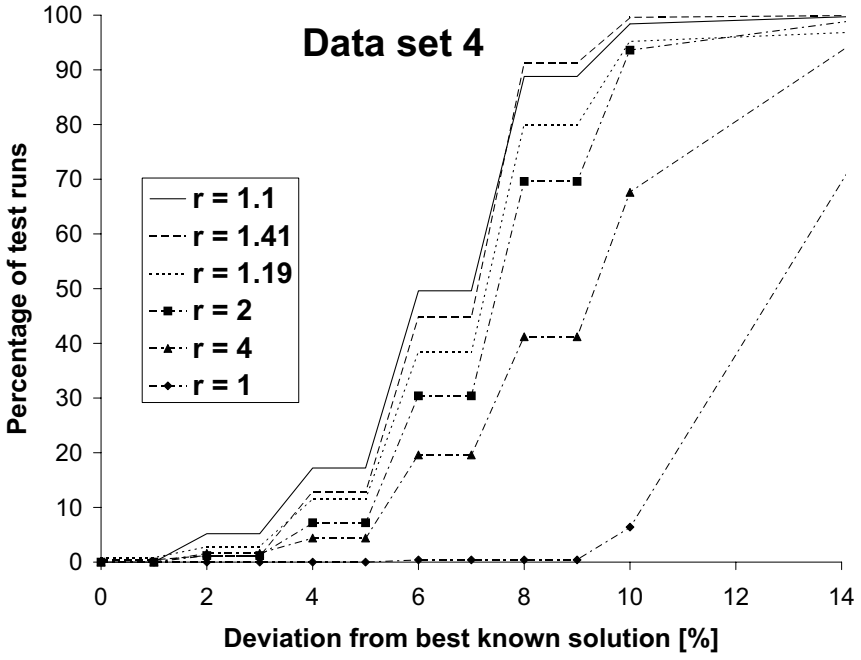

Figure A.24: Effect of r: RDGR on GTAAP, data set 4 (unsolvable, 500 runs).

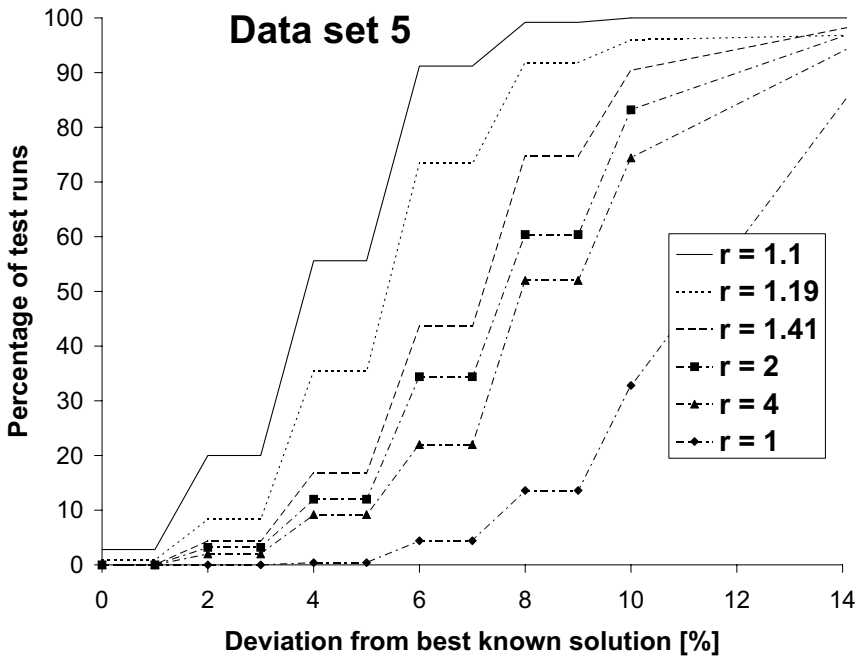

Figure A.25: Effect of r: RGR on GTAAP, data set 5 (solvable, 500 runs).

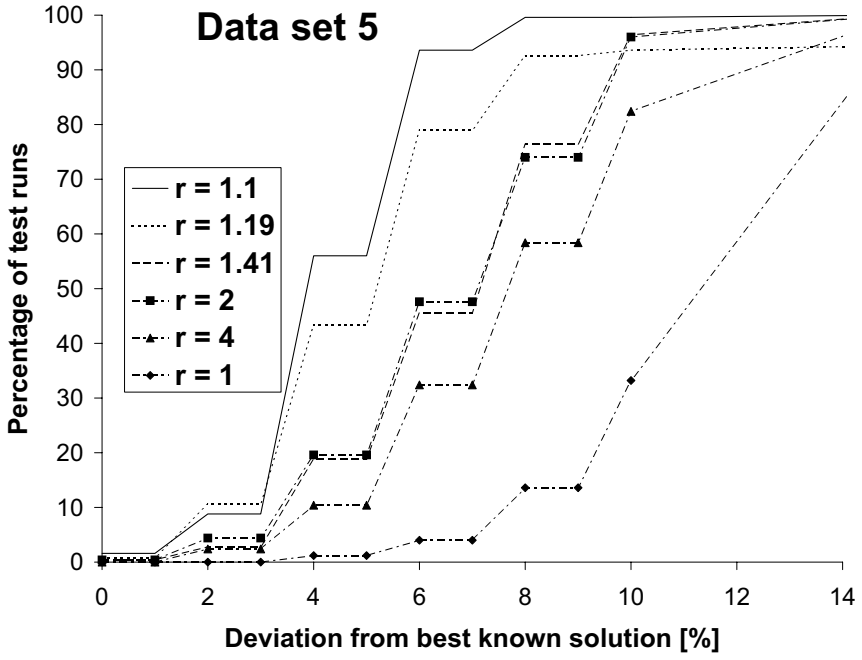

Figure A.26: Effect of r: RDGR on GTAAP, data set 5 (solvable, 500 runs).

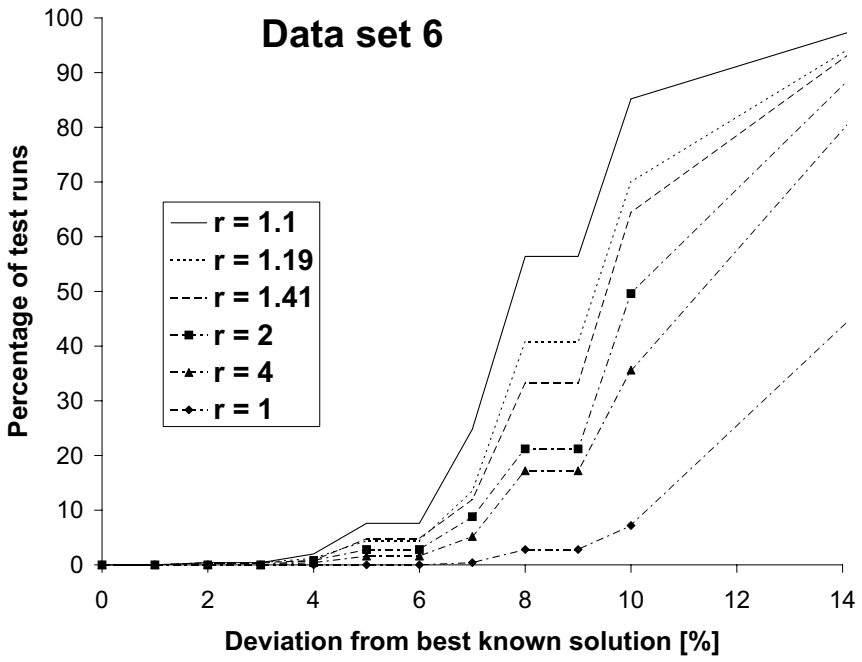

Figure A.27: Effect of r: RGR on GTAAP, data set 6 (solvable, 500 runs).

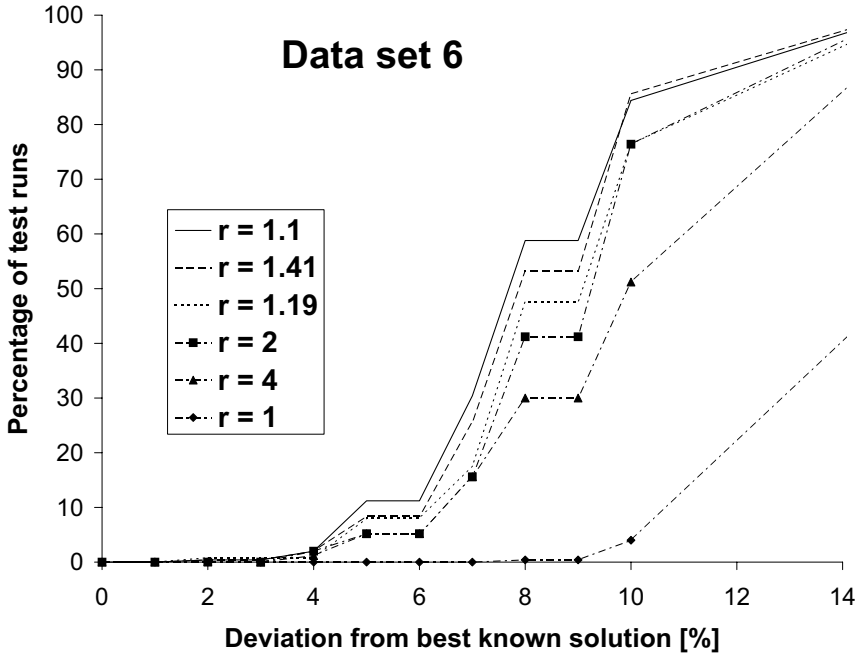

Figure A.28: Effect of r: RDGR on GTAAP, data set 6 (solvable, 500 runs).

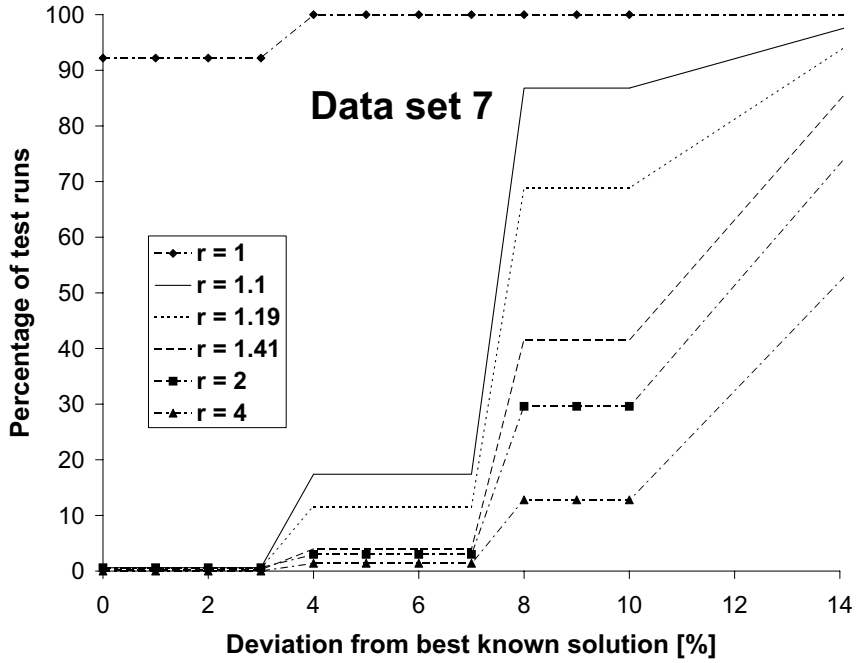

Figure A.29: Effect of r: RGR on GTAAP, data set 7 (solvable, 500 runs).

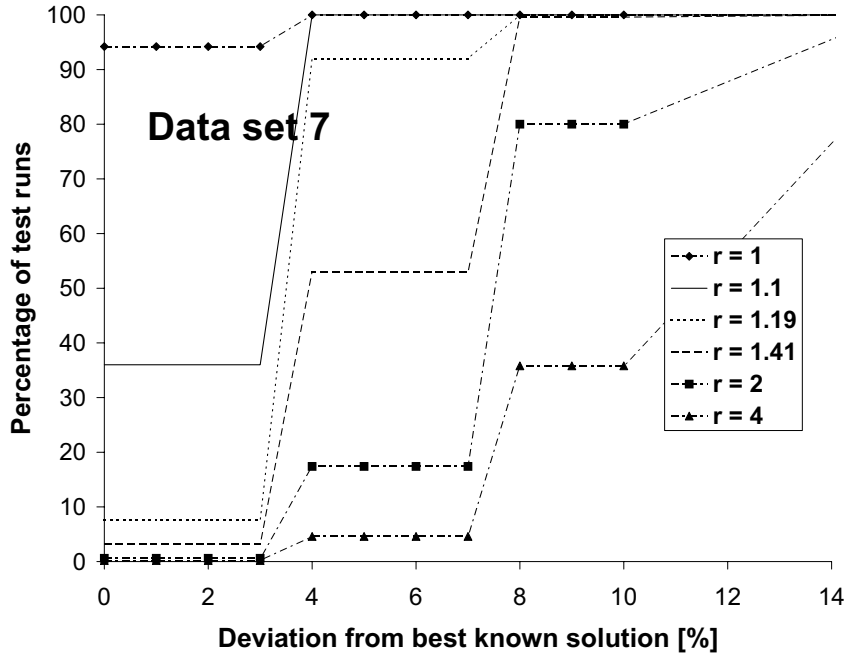

Figure A.30: Effect of r: RDGR on GTAAP, data set 7 (solvable, 500 runs).

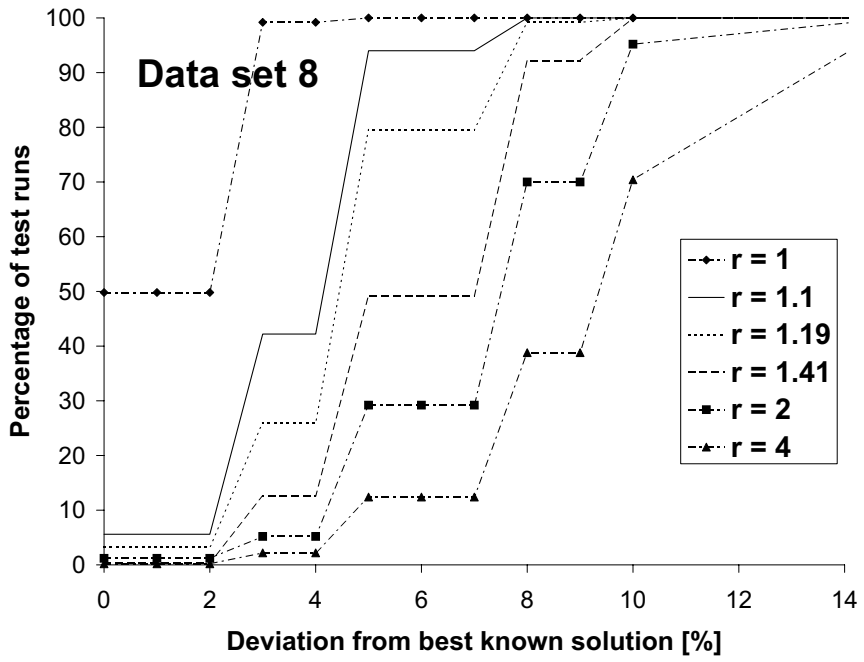

Figure A.31: Effect of r: RGR on GTAAP, data set 8 (solvable, 500 runs).

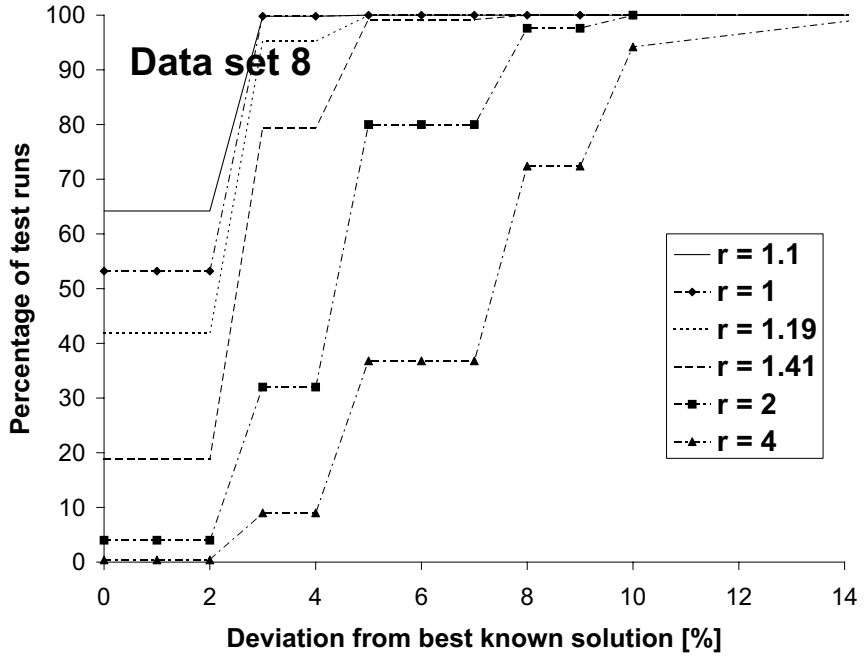

Figure A.32: Effect of r: RDGR on GTAAP, data set 8 (solvable, 500 runs).

Bibliography

[Barták, 1998] R. Barták. On-Line Guide to Constraint Programming. kti.ms.mff.cuni.cz/˜bartak/constraints, 1998.

[Bistarelli *et al.*, 1995] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proc. of the 14 th IJCAI*, pages 624–630, 1995.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.

[Freuder and Wallace, 1992] E.C. Freuder and R.J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[Freuder, 1989] E.C. Freuder. Partial Constraint Satisfaction. In *Proc. of the 11 th IJCAI*, pages 278–283, Detroit, MI, 1989.

[Glaubius and Choueiry, 2002a] R. Glaubius and B.Y. Choueiry. Constraint Constraint Modeling and Reformulation in the Context of Academic Task Assignment. In *Working Notes of the Workshop Modelling and Solving Problems with Constraints, ECAI 2002*, Lyon, France, 2002.

[Glaubius and Choueiry, 2002b] R. Glaubius and B.Y. Choueiry. Constraint Modeling in the Context of Academic Task Assignment. In Pascal Van Hentenryck, editor, 8^{th} *International Conference on Principle and Practice of Constraint Programming (CP 02)*, volume 2470 of *LNCS*, page 789. Springer, 2002.

[Glaubius, 2001] R. Glaubius. A Constraint Processing Approach to Assigning Graduate Teaching Assistants to Courses. Undergraduate Honors Thesis. Department of Computer Science & Engineering, University of Nebraska-Lincoln, 2001.

[Gomes *et al.*, 1998] C.P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 98)*, pages 431–437, Madison, Wisconsin, 1998.

[Haralick and Elliott, 1980] R.M. Haralick and G.L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[Hoos and Stützle, 2004] H.H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann, 2004.

[Hoos, 1998] H.H. Hoos. *Stochastic Local Search—Methods, Models, Applications*. PhD thesis, Technische Universität Darmstadt, Germany, 1998.

[Junker, 2002] U. Junker. Preference-Based Search and Multi-Criteria Optimization. In Rina Dechter, Michael Kearns, and Richard S Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 02)*, pages 34–40, Menlo Park, California, 2002. American Association for Artificial Intelligence, AAAI Press.

[Kautz *et al.*, 2002] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic Restart Policies. In *AAAI*, 2002.

[Liu *et al.*, 2002] J. Liu, H. Jing, and Y.Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136:101–144, 2002.

[Luby *et al.*, 1993] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. In *Israel Symposium on Theory of Computing Systems*, pages 128–133, 1993.

[Minton *et al.*, 1992] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58:161–205, 1992.

[Revesz, 2002] P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, New York, 2002.

[van Hemert, 2004] J.I. van Hemert. RandomCSP: generating constraint satisfaction problems randomly. homepages.cwi.nl/ jvhemert/randomcsp.html, 2004.

[Wallace and Freuder, 1995] R. Wallace and E. Freuder. Heuristic methods for over-constrained constraint satisfaction problems. In M. Jampel, E. Freuder, and M. Maher, editors, *OCS 95: Workshop on Over-Constrained Systems at CP 95*, Cassis, Marseilles, 1995.

[Wallace, 1996] R.J. Wallace. Analysis of heuristic methods for partial constraint satisfaction problems. In *Principles and Practice of Constraint Programming*, pages 482–496, 1996.

[Walsh, 1999] T. Walsh. Search in a small world. In *Proc. of the 16 th IJCAI*, pages 1172–1177, 1999.

[Zou and Choueiry, 2003a] H. Zou and B.Y. Choueiry. Characterizing the Behavior of a Multi-Agent Search by Using it to Solve a Tight, Real-World Resource Allocation Problem. In *Workshop on Applications of Constraint Programming*, pages 81–101, Kinsale, County Cork, Ireland, 2003.

[Zou and Choueiry, 2003b] H. Zou and B.Y. Choueiry. Multi-agent Based Search versus Local Search and Backtrack Search for Solving Tight CSPs: A Practical Case Study. In *Working Notes of the Workshop on Stochastic Search Algorithms (IJCAI 03)*, pages 17–24, Acapulco, Mexico, 2003.

[Zou, 2003] H. Zou. Iterative Improvement Techniques for Solving Tight Constraint Satisfaction Problems. Master's thesis, Department of Computer Science & Engineering, University of Nebraska-Lincoln, December 2003.