Characterization of a New Restart Strategy for Randomized Backtrack Search

Venkata Praveen Guddeti and Berthe Y. Choueiry

Constraint Systems Laboratory
Computer Science & Engineering
University of Nebraska-Lincoln
Email: {vguddeti, choueiry}@cse.unl.edu

Abstract. We propose an improved restart strategy for randomized backtrack search, and evaluate its performance by comparing to other heuristic and stochastic search techniques for solving random problems and a tight real-world resource allocation problem. The restart strategy proposed by Gomes et al. [1] requires the specification of a cutoff value determined from an overall profile of the cost of search for solving the problem. When no such profile is known, the cutoff value is chosen by trial-and-error. The Randomization and Geometric Restart (RGR) proposed by Walsh does not rely on a cost profile but determines the cutoff value as a function of a constant parameter and the number of variables in the problem [2]. Unlike these strategies, which have fixed restart schedules, our technique (RDGR) dynamically adapts the value of the cutoff parameter to the results of the search process. Our experiments investigate the behavior of these techniques using the cumulative distribution of the solutions, over different run-time durations, values of the cutoff, and problem types. We show that distinguishing between solvable and over-constrained problem instances yields new insights on the relative performance of the search techniques tested. We propose to use this characterization as a basis for building new strategies of cooperative, hybrid search.

1 Introduction

We have developed a system for modeling and solving a resource allocation problem, which is the assignment of Graduate Teaching Assistants (GTA) to courses in our department [3]. We exploit this system as a platform for developing and characterizing new problem-solving strategies. The research we describe in this paper was motivated and enabled by this project. However, our results are here extended beyond this particular application.

The Graduate Teaching Assistants Assignment Problem (GTAAP) is a critical and arduous task that the department's administration has to drudge through every semester. 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 we have implemented to solve this problem. Such an insight is unlikely to be gained from testing toy problems, and surely difficult from testing random problems. We show that the identified behaviors apply beyond our application. The contributions of this paper are as follows: (1) The development of a new dynamic restart strategy for

randomized backtrack search, and (2) an empirical evaluation of the performance of this new strategy and a comparison with other heuristic and stochastic search techniques on a real-world problem and on randomly generated binary CSPs.

This paper is structured as follows. Section 2 describes the GTA assignment problem (GTAAP) and our implementations of a backtrack search, a local search, and a multi-agent search technique for solving it. Section 3 introduces our new proposed dynamic restart strategy for randomized backtrack search and our implementation of Walsh's restart strategy [2]. Section 4 presents our experiments and our observations. Finally, Section 5 concludes the paper and provides directions for future research.

2 GTA Assignment Problem

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 [4–6]. 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 0 to 5) must be maximized. 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. 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 [7]. Our constraint-model represents the courses as variables, the GTAs as domain values, and the assignment rules as a number of unary, binary, and non-binary constraints. We 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.) We implemented a number of search strategies for solving this problem (summarized below). We tested these search techniques on the real-world data-sets shown in Table 1. Each course

Table 1. Characteristics of the data sets.

Data set	Spring2001b	Fall2001b	Fall2002	Fall2002-NP	Spring2003	Spring2003-NP
Reference	1	2	3	4	5	6
Solvable?	×	√	×	×	√	√
#Variables	69	65	31	59	54	64
Max domain size	26	34	28	28	34	34
Total capacity	26	30	11.5	27	27.5	31
Total load	29.6	29.3	13	29.5	27.4	30.2
Ratio = $\frac{\text{TotalCapacity}}{\text{TotalLoad}}$	0.88	1.02	0.88	0.91	1.00	1.02

has a load that indicates the weight of the course. For example, a value of 0.5 means this course needs one-half of a GTA. The *total load* of a semester is the cumulative load of the individual courses. Each GTA has a capacity factor which indicates the maximum

course weight he/she can be assigned during the semester. The sum of the capacities of all GTAs represents the *total capacity*.

We compare our new dynamic restart strategy (RDGR) with 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 restart (RGR). All strategies implement the above two optimization criteria, except ERA, which models the GTAAP as a satisfaction problem. These search techniques were separately implemented on the same model and data structures by students competing to produce the best results. Below, we summarize the design of BT, LS, and ERA.

2.1 Heuristic backtrack search

Our heuristic backtrack (BT) search is a depth-first search with forward checking [8]. Because the problem may be over-constrained, we 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). This solution is equivalent to adding a dummy value in the variables' domains. Our implementation is described in detail in [5]. We have implemented several ordering heuristics to improve the performance of search (see [9]). Our experiments showed that dynamic variable ordering is consistently superior to static ordering, but that the influence of the other factors is not significant in the context of our application.

All these strategies exhibited a serious vulnerability to thrashing (i.e., searching unpromising parts of the search space), which seriously undermined their ability to explore wider areas of the search space. Indeed, although BT is theoretically sound and complete, *the size of the search space makes such guarantees meaningless in practice*. Figure 1 illustrates the gravity of thrashing for a problem with 69 variables and 26 values. We define the 'shallowest level' as the shallowest level in a search tree attained by the backtracking mechanism along any given path. The percentage denotes the ratio

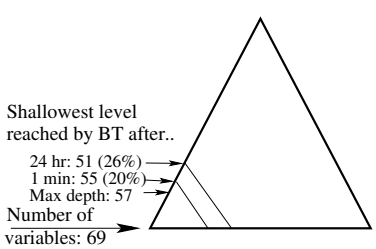

Fig. 1. BT thrashing in large search spaces.

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 %

Fig. 2. BT search thrashing.

$\frac{\text{number of variables} - \text{shallowest level}}{\text{number of variables}}$. 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. Figure 2 shows, for each data set, the number of variables, the longest solution (max depth), and the shallowest BT levels in terms of the level and the percentage of backtracking in the search tree attained after 5 minutes and 6 hours.

As the problem size increases, the effects of thrashing become more important. Table 2 shows the performance of BT on data set 1 for various run times. Even after letting

Table 2. Performance of BT for various running times.

Data set 1 (69 variables, over-constrained)						
Running 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

our best heuristic backtrack search run for over 24 hours, the quality of the solution, in terms of solution length, is not improved. The improvement of the assignment quality, in terms of the geometric mean of the preference values, is insignificant. Finally, we notice that the assignment of the first 51 variables in the ordering was never undone. Consequently, in practice, completeness is a purely theoretical feature.

2.2 Local search

Zou and Choueiry designed and implemented a greedy, local search (LS) technique for the GTAAP system [10–12]. It is a hill-climbing search using the min-conflict heuristic for value selection [13]. It begins with a complete, random assignment (not necessarily consistent), and tries to improve it by changing inconsistent assignments in order to reduce the number of constraint violations. In order to deal effectively with global constraints (e.g., capacity constraints), we identify, one at a time and in random order, a variable that satisfies all remaining constraints as consistent, and propagate the effects of this consistent assignment by filtering the domains of the remaining variables. This design decision effectively handles non-binary constraints. Our local search is greedy in the sense that consistent assignments are not undone. Moreover, a random-walk strategy is applied to escape from local optima [14]. With a probability $(1 - p)$, the value of a variable is chosen using the min-conflict heuristic, and with probability p this value is chosen randomly. Following the indications of [14] and after testing, $p = 0.02$ is used. Finally, random restarts are used to break out of local optima.

2.3 Multi-agent search

Liu et al. proposed the ERA algorithm (Environment, Reactive rules, and Agents) as a multi-agent-based search for solving CSPs [15]. Zou and Choueiry implemented and tested ERA for solving GTAAP [10–12]. In ERA, each agent represents a variable. The positions of an agent in the environment correspond to the values in the domain of the variable. Starting from a random positioning of the agents in the environment, each agent evaluates the quality of its positions given the positions of the remaining agents and decides to move to what seems to be the best position, the choice being determined stochastically by the reactive rules. The agents keep moving until they all reach a satisfying position (i.e., a full, consistent solution) or a certain time period has elapsed. 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*. Zou and Choueiry showed that the extreme mobility of agents in the environment is the reason behind ERA's amazing immunity to local optima [10–12]. They found that ERA is indeed the only search technique to solve GTAAP instances that remain unsolved by all other techniques tested. They uncovered the weakness of ERA on over-constrained problems and characterized it as a livelock phenomenon (where some agents keep forcing each others out of chosen positions thus causing cycles and undermining the stability of search). Finally, they showed how this phenomenon can be advantageously used to isolate, identify, and represent conflicts in a compact manner.

3 Randomized BT search with restarts

Unlike ERA and local search, general backtrack (BT) search is, in principle, complete and sound. However, the performance of heuristic BT is seriously undermined by thrashing. Thrashing can be explained by incorrect heuristic choices made early in the search process. We explore randomization in BT as a way to overcome this shortcoming of systematic search. First we review the main concepts, then we describe the two strategies that we tested.

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 [1]. Thrashing in BT search 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. It is important to restart search 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. 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 [16], the randomization and rapid restart (RRR) of Gomes

et al. [1], and the randomization and geometric restarts (RGR) of Walsh [2]. 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 [16] and RRR [1]) 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 [16] and Walsh's RGR [2]). 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 restart. We propose a restart strategy that dynamically adapts the cutoff value for each restart based on the performance of previous restarts. Our strategy loses the guarantee of completeness, which, anyway, is not achievable on large problems.

3.1 Randomization and Geometric Restarts

Walsh proposed the Randomization and Geometric Restarts (RGR) strategy to automate the choice of the cutoff value [2]. 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 ith 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 studied various values of r and report them in Section 4.2. We combined this restart strategy with the backtrack search of Section 2.1, randomizing the selection of variable-value pairs.

3.2 Randomization and Dynamic Geometric Restarts

We now introduce a simple but effective improvement to RGR. All static restart strategies suffer from the problem of increasing cutoff values after each restart. While this ensures completeness of the search, it 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 does not necessarily increase, completeness is no longer guaranteed. This situation is acceptable in application domains (like ours) with large problem size where completeness is, anyway, infeasible in practice. Smaller cutoff values result in a larger number of restarts taking place in RDGR than RGR, which increases the probability of finding a solution. All other implementation details are similar to RGR.

Let C_i be cutoff value for the ith 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:

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

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. Each time search begins with a different random seed, it traverses different search paths. Some paths may be more fruitful than others. RGR and RDGR increase the cutoff values in the same way on search paths that improve solutions. When the solution is not being improved, RGR keeps increasing the cutoff values, thus making RGR more of a randomized BT search than a randomized BT search with restarts. In contrast, RDGR maintains the cutoff value. Figure 3 shows that RGR increases the cutoff values across iterations significantly more rapidly than RDGR does, for r=1.1 and 2 on random binary CSPs. This

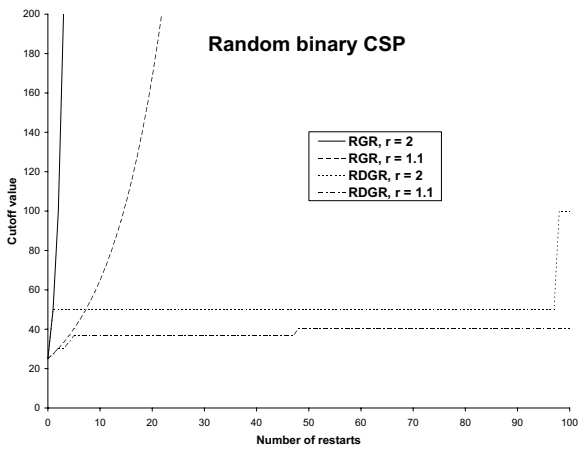

Fig. 3. Increase of the cutoff value (3 minutes).

explains the dynamic nature of RDGR. For problems that are not tight, solutions are found within a few restarts, and RGR and RDGR exhibit similar behaviors. For tight and over-constrained problems, RDGR seems to dominate RGR as we show in our experiments (Section 4).

4 Experiments and results

We tested and compared the above listed 5 search strategies, namely: BT (Section 2.1), LS (Section 2.2), ERA (Section 2.3), RGR (Section 3.1), and RDGR (Section 3.2). BT is deterministic and the other 4 search techniques (i.e., LS, ERA, RGR, and RDGR) are stochastic. In the terminology introduced by Hoos and Stützle in [17], these are optimization Las Vegas algorithms, RGR is probabilistically approximately complete

(PAC), and LS, ERA, and RDGR are essentially incomplete. 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.

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 [17]. SQD's are cumulative distributions of the solution quality, similar to the cumulative distributions of run-time in run-time distributions. The horizontal axis 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 the longest solution and, the point 20% denotes a solution that is 20% shorter that the longest solution. The vertical axis represents the percentage of test runs.
2. *Descriptive statistics* of all the solutions found, for all search techniques. This includes the measures: mean, median, mode, standard deviation, minimum, and maximum of the solution.
3. *95% confidence interval* of the mean improvement. The confidence interval was computed using the Mann-Whitney test. Table 3 reports the improvements of RDGR over RGR and ERA.

Table 3. Improvements of RDGR with 95% confidence level.

Data set	Improvements over RGR		Improvements over ERA	
	Lower limit	Upper limit	Lower limit	Upper limit
1	1.16	1.61	45.16	46.77
2	1.53	1.61	-6.15	-6.15
3	3.44	3.44	27.58	31.03
4	1.85	1.85	24.07	27.77
5	0	1.85	-3.7	-3.7
6	1.56	1.56	-6.25	-6.25

We tested these search techniques on the 6 real-world data-sets of the GTAAP of Table 1 and 4 sets of randomly generated problems. 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. 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 for data set 1, which justifies our decision. We report the results for the following data sets (the same qualitative observations hold across all data sets):

- Data set 1 as a representative of an over-constrained problem.
- Data set 5 as a representative of a tight but solvable problem.

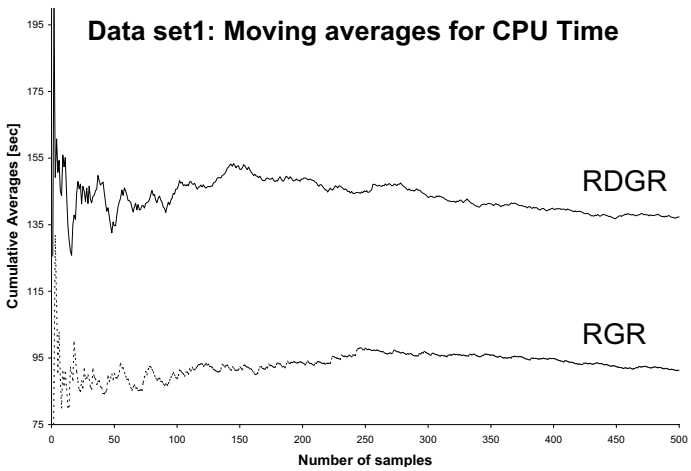

Fig. 4. Moving average for CPU run-times for data set 1.

For randomly generated problems, we used the model-B-type generator of Hemert [18]. We generated three types of randomly generated problems, each containing 100 instances and each instance run for 3 minutes:

- *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.
- *Over-constrained instances*. The second type of randomly generated problems are *over-constrained* binary CSPs with 40 variables, uniform domain size of 20 values, 0.5 proportion of constraints, and 0.5 constraint tightness.
- *Instances at the phase transition*. The third type of randomly generated problems are from the *phase transition* area. These are binary CSPs with 25 variables, uniform domain size of 15 values, 0.5 proportion of constraints, and 0.36 constraint tightness. We split these instance into two sets, each of 100 instances, separating solvable instances and unsolvable instances.

4.1 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. The results are shown in Figures 5 and 6. In both these figures, RDGR consistently outperforms RGR over different run-times. Further, increasing the running time has no affect on the relative dominance of algorithms.

4.2 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 7, 8, 9, and 10 show the influence of the ratio r used to increase the cutoff value in RGR and RDGR. In accordance

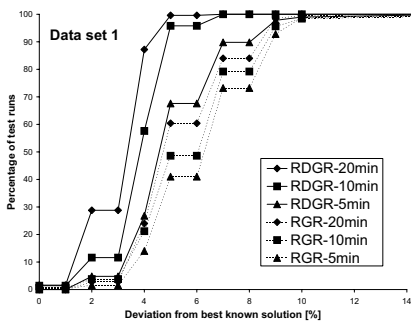

Fig. 5. Varying run time: GTAAP, over-constrained.

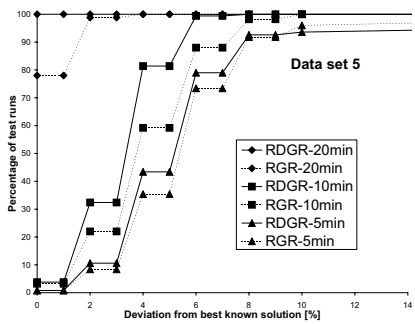

Fig. 6. Varying run time: GTAAP, solvable.

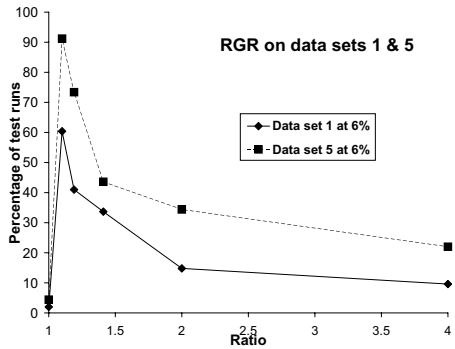

Fig. 7. Effect of r: RGR on GTAAP.

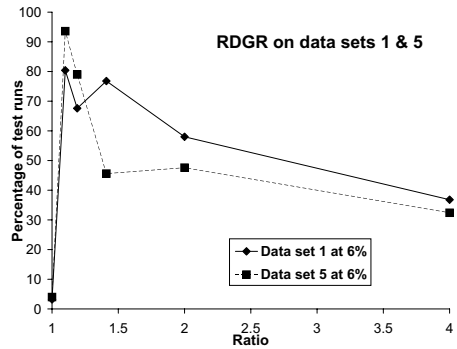

Fig. 8. Effect of r: RDGR on GTAAP.

with [2], Figures 7 and 9 show that a value of r=1.1 is the best among the values tested for RGR. This While, for RGR, this optimal ratio does not change with the problem type (i.e., GTAAP vs. random problem), it does for RDGR. For the GTAAP, it is r=1.1 (Figure 8). For randomly generated problems, it is r=2 (Figure 10). Our experiments indicate that the curves remains flat around these 'optima.'

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

In this section we compare the relative performance of all the search techniques developed for the GTAAP system. Each stochastic algorithm was run 500 times of 10 min each on the GTAAP data-set, and on 100 instances of random CSPs of 3 min each. Figures 11 and 12 show the relative performance of the search techniques on GTAAP data. Figures 13, 14, 15, and 16 show the relative performance for the random problems. We do not show LS and ERA in Figure 14 because they go off the scale.

Improvement of RDGR over BT: Table 4 shows that the maximum value of the solution sizes produced by RDGR is clearly greater than that of the solution sizes produced by

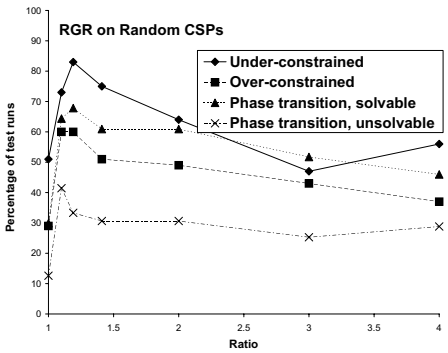

Fig. 9. Effect of r: RGR on random CSPs.

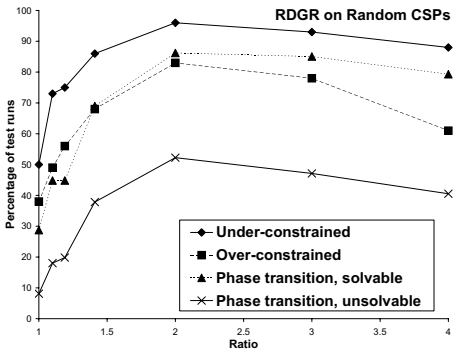

Fig. 10. Effect of r: RDGR on random CSPs.

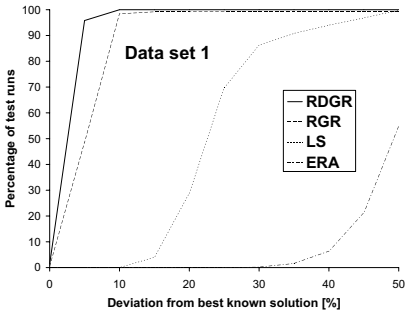

Fig. 11. SQDs: GTAAP, over-constrained.

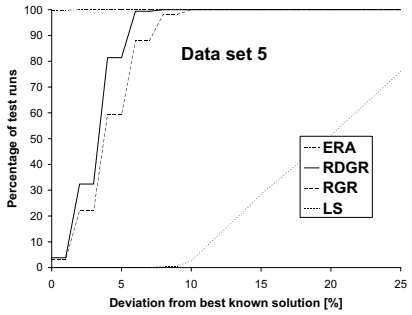

Fig. 12. SQDs: GTAAP, solvable.

BT. However, due to its stochastic nature, RDGR suffers from high instability in its solution quality.

Superiority of RDGR over LS: The performance of RDGR is clearly superior to that of LS (see Table 4 and Figures 11, 12, 13, 15, and 16). Although the solution quality is highly variable for both RDGR and LS, the low mean value of the solution quality of LS ensures that RDGR remains superior to LS.

Superiority of RDGR over ERA on over-constrained problems: On over-constrained problems (Figure 11 and Table 3), the deadlock phenomenon prevents ERA from finding solutions of quality comparable to those found by the other techniques [10–12]. BT, LS, RDGR, and RGR do not exhibit such a dichotomy of behavior between over-constrained cases and solvable instances.

Performance of ERA: On solvable problem instances (Figures 12 and 13), ERA dominates all techniques. It is the only algorithm that finds complete solutions for nearly all the runs. ERA completely dominates LS. However, on over-constrained problem instances (Figure 11), RDGR, RGR, BT and LS are superior to ERA due to the deadlock

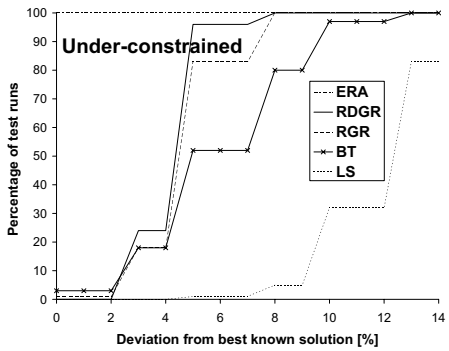

Fig. 13. SQDs: under-constrained, random CSPs.

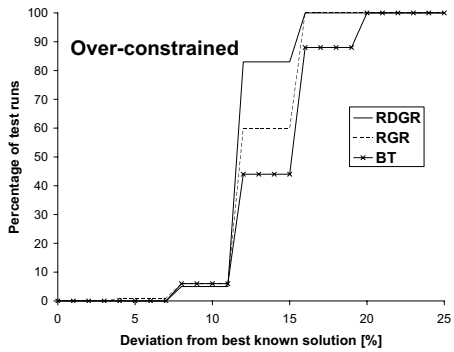

Fig. 14. SQDs: over-constrained, random CSPs.

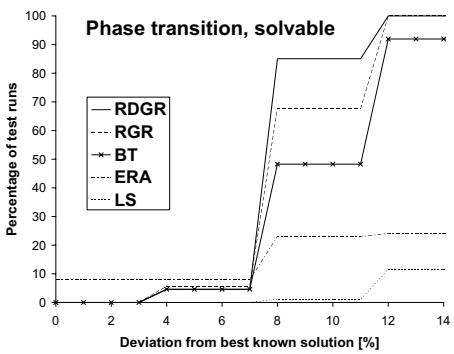

Fig. 15. SQDs: solvable random CSPs, at phase transition.

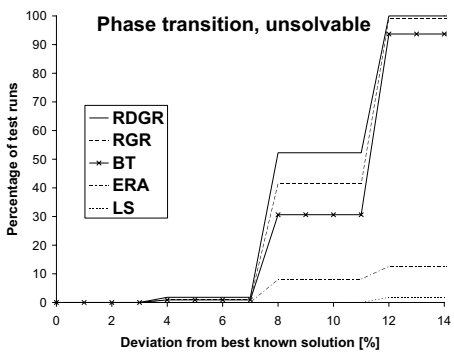

Fig. 16. SQDs: unsolvable random CSPs, at phase transition.

phenomenon. At the phase transition (Figures 15 and 16), the behavior of ERA is independent of the solvability of the problem. ERA performs only better than LS, while RDGR, RGR and BT perform better than ERA. This difference in performance of ERA may have to do with the structure of the randomly generated problems and the GTA problem. More tests are needed to understand this phenomenon.

RDGR is more stable than RGR: Table 5 shows the standard deviation of RGR and RDGR on the GTAAP data sets. Due to 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 5 show that RDGR is relatively more stable than RGR.

Sensitivity of LS to local optima: LS sensitivity to local optima makes it particularly unattractive in our context. Even BT outperforms LS.

Table 4. Statistics of solution size (500 runs, 10 min each).

	Data set 1 (69 variables, over-constrained)						**Data set 5** (54 variables, tight but solvable)					
Search	Mean	Median	Mode	Standard deviat.	Min.	Max.	Mean	Median	Mode	Standard deviat.	Min.	Max.
BT	57	57	57	0	57	57	52	52	52	0	52	52
LS	47.12	48	49	4.44	30	55	42.88	44	46	3.94	29	50
ERA	30.99	31	32	4.37	18	45	53.99	54	54	0.04	53	54
RDGR	59.66	60	60	0.77	58	62	52.17	52	52	0.78	50	54
RGR	58.27	58	58	2.83	23	62	51.70	52	52	1.04	49	54

Table 5. Standard deviation in solution quality on GTAAP data.

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

Table 6. Average number of restarts on GTAAP data.

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

Larger number of restarts in RDGR: Table 6 shows the average number of restarts occurring in RGR and RDGR. This confirms our expectations stated in Section 3.2 that RDGR performs more restarts than RGR.

Summary: The following five statements, where $\succ$ denotes dominance of an algorithm over another, summarize the behavior of the 5 search strategies, also shown in Table 7:

- 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 15). 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.

5 Conclusions and future work

By addressing a real-world application, we are able to identify, characterize, and compare the behavior of various search techniques. BT is stable but suffers from thrashing. LS is vulnerable to local optima. ERA shows difference in performance with different

Table 7. Comparing the behaviors of search strategies.

	Characteristics
ERA	**General:** Stochastic and incomplete
	Tight but solvable problems: Immune to local optima
	Over-constrained problems: Deadlock 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 deadlock, 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

problem types: while it has an amazing ability to solve under-constrained problems, its performance degrades on over-constrained problems due to the livelock phenomenon. Restart strategies effectively prevent thrashing, but their solution quality is highly variable. RGR operates by increasing cutoff values at every restart, which increases its vulnerability to thrashing. RDGR attenuates this effect by making the cutoff value depend upon the result obtained at the previous restart, which increases the number of restarts in comparison to RGR. Consequently, RDGR exhibits a more stable behavior than RGR while yielding at least as good solutions. In the future, we plan to study the following directions:

1. Validate our findings on other real-world case-studies.
2. Design 'progress-aware' restart strategies, that is, strategies that can decide, *during* a given restart, whether to continue or abandon this particular execution.
3. Use our current application as a 'platform' to study and characterize the performance of other deterministic and stochastic search techniques.
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.

Acknowledgments

This work is supported by NSF grants #EPS-0091900 and CAREER #0133568. The experiments were conducted utilizing the Research Computing Facility of the University of Nebraska-Lincoln.

References

1. Gomes, C.P., Selman, B., Kautz, H.: Boosting Combinatorial Search Through Randomization. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98). (1998) 431–437
2. Walsh, T.: Search in a Small World. In: Proc. of the 16 th IJCAI. (1999) 1172–1177
3. Lim, R., Guddeti, V.P., Choueiry, B.Y.: An Interactive System for Hiring and Managing Graduate Teaching Assistants. In: Conference on Prestigious Applications of Intelligent Systems (ECAI 04), Valencia, Spain (2004) 730–734
4. Glaubius, R.: A Constraint Processing Approach to Assigning Graduate Teaching Assistants to Courses. Undergraduate Honors Thesis. Department of Computer Science & Engineering, University of Nebraska-Lincoln (2001)
5. Glaubius, R., Choueiry, B.Y.: 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)
6. Glaubius, R., Choueiry, B.Y.: Constraint Modeling in the Context of Academic Task Assignment. In Hentenryck, P.V., ed.: 8th International Conference on Principle and Practice of Constraint Programming (CP'02). Volume 2470 of LNCS., Springer (2002) 789
7. Freuder, E.C., Wallace, R.J.: Partial Constraint Satisfaction. Artificial Intelligence **58** (1992) 21–70
8. Prosser, P.: Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence **9 (3)** (1993) 268–299
9. Guddeti, V.P.: Empirical Evaluation of Heuristic and Randomized Backtrack Search. Master's thesis, Computer Science & Engineering, University of Nebraska-Lincoln (2004)
10. Zou, H., Choueiry, B.Y.: 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, Kinsale, County Cork, Ireland (2003) 81–101
11. Zou, H.: Iterative Improvement Techniques for Solving Tight Constraint Satisfaction Problems. Master's thesis, Computer Science & Engineering, University of Nebraska-Lincoln (2003)
12. Zou, H., Choueiry, B.Y.: 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), Acapulco, Mexico (2003) 17–24
13. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. Artificial Intelligence **58** (1992) 161–205
14. Barták, R.: On-Line Guide to Constraint Programming. kti.ms.mff.cuni.cz/˜bartak/constraints (1998)
15. Liu, J., Jing, H., Tang, Y.: Multi-Agent Oriented Constraint Satisfaction. Artificial Intelligence **136** (2002) 101–144
16. Luby, M., Sinclair, A., Zuckerman, D.: Optimal Speedup of Las Vegas Algorithms. In: Israel Symposium on Theory of Computing Systems. (1993) 128–133
17. Hoos, H., Stützle, T.: Stochastic Local Search Foundations and Applications. Morgan Kaufmann (2004)
18. van Hemert, J.I.: RandomCSP: generating constraint satisfaction problems randomly. homepages.cwi.nl/ jvhemert/randomcsp.html (2004)