An Empirical Study 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 compare its performance to other search mechanisms in the context of solving 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. Walsh proposed the strategy Randomization and Geometric Restart (RGR), which 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. We empirically evaluate the performance of RDGR by comparing it against a number of heuristic and stochastic search techniques, including RGR, using the cumulative distribution of the solutions. We compare the performance of RGR and RDGR over different run-time durations, different values of the cutoff, and for different 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 in our real-world case-study 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 apply Constraint Processing techniques to model and solve the assignment of Graduate Teaching Assistants (GTA) to courses in our department. The idea of this application was first found on the web-page of Rina Dechter. This 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 believe that the identified behaviors apply beyond our application and are currently working on testing this hypothesis. 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 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 [3–6]. 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. 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 [7]. We provide a constraint model of this problem by representing 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, 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). All strategies implement the above two optimization criteria, except ERA, which models the GTA assignment problem as a satisfaction problem. We tested these search techniques on the real-world data-sets shown in Table 1. Each course 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*.

Below, we review the search techniques to which we compare our new dynamic restart strategy. These search techniques were implemented separately by students, competing to produce the best results.

Data set	Spring2001b	Fall2001b	Fall2002	Fall2002-NP	Spring2003	Spring2003-NP
Reference	1	2	3	4	5	6
Solvable?	$\times$	$\surd$	$\times$	$\times$	$\surd$	$\surd$
#Courses (#variables)	69	65	31	59	54	64
#GTAs (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

Table 1. *Characteristics of the data sets.*

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). Our implementation is described in detail in [4]. Note that adding dummy values to deal with over-constrained instances is a bad choice in our context as it increases the branching factor (which is already too large) and consequently the worsens the thrashing behavior.

We have also implemented several variable and value ordering heuristics to improve the performance of search. For variable ordering we implemented two heuristics for choosing the most constrained variable first: least domain and ratio domain size to degree. We applied these heuristics both statically (i.e., sequence of variables is determined before search and not modified thereafter) and dynamically (i.e., the next variable is chosen after each instantiation). For value ordering, we tested 3 different heuristics: random ordering, and sorted by preference and by occurrence frequency in the domains. The combination of these heuristics yielded 12 ordering strategies. 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.

Furthermore, all these strategies exhibited a serious vulnerability to thrashing, 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 thrashing for a problem with 69 variables and 26 values. Here, the percentage is $\frac{\text{number of variables} - \text{shallowest level}}{\text{number of variables}}$. 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.

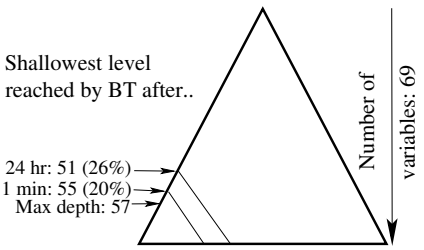

Fig. 1. *BT search thrashing in large search spaces.*

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.*

2.2 Local search

Zou and Choueiry designed and implemented a greedy, local search (LS) technique to solve the GTA Assignment Problem [9–11]. It is a hill-climbing search using the min-conflict heuristic for value selection [12]. 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. The effects of consistent assignments are propagated over the domains of the variables with inconsistent assignments. This design decision effectively handles non-binary constraints. Also, the 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 [13]. 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 [13], $p = 0.02$ is used. Finally, random restarts are used to break out of local optima.

2.3 ERA model

Zou and Choueiry also implemented a multi-agent-based search for solving the GTA Assignment Problem [9–11]. Liu et al. [14] proposed the ERA algorithm (Environment, Reactive rules, and Agents), a multi-agent-based search for solving CSPs. Each agent represents a variable. The positions of an agent in the environment E correspond to the values in the domain of the variable. 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. An 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 (R). 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 in [9–11]. 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 deadlock phenomenon undermines its stability resulting in particularly short solutions. However, it has been shown that 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 has the nice property of being complete and sound. However, the performance of heuristic BT proved to be unpredictable in practice and seriously undermined by thrashing (i.e., searching unpromising parts of the search space). Thrashing can be explained by incorrect heuristic choices made early in the search process, and forces BT search to explore large 'barren' parts of the search tree. As the problem size increases, the effects of thrashing become more important, and the performance of search dramatically decreases. Another major problem is the high degree of unpredictability in the run-time of BT over a set of problem instances, even within the same problem type. Gomes et al. [1] noticed that this run-time can be often modeled by a heavy-tailed distribution. They proposed to use randomization and restart strategies 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. [1] 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. 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 [15], 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 [15] and RRR [1]) employ an optimal cutoff value that is fixed for all the restarts of a particular problem instance.

However, 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 setting 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 [15] and Walsh's RGR [2]). They utilize the idea of an increasing cutoff value in order to ensure the completeness of the restart strategy. However, if these restart strategies do not find a solution in the initial few restarts, then the increasing cutoff value leads to fewer restarts, which may yield thrashing and diminishes the benefits of the restart strategy. We propose a restart strategy that dynamically adapts the cutoff value for each restart based on the performance of previous restarts. We do this at the expense of completeness. We also implemented RGR and empirically compared it with our dynamic restart strategy.

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 this paper. 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. 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 follow the same cutoff schedules for search paths that improve solutions. When this is not the case, RGR cutoff values keep on increasing, thus making RGR more of a randomized BT search than a randomized BT search with restarts. In contrast, RDGR keeps cutoff at smaller values. This explains the dynamic nature of RDGR. For problems that are not tight, solutions are found within a few restarts. In such cases, 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 [16], these are optimization Las Vegas algorithms, RGR is probabilistically approximately complete (PAC), and LS, ERA, and RDGR are essentially incomplete. We tested these search techniques on the 6 real-world data-sets of the GTA Assignment Problem shown in Table 1. Three of the data sets (1, 3, and 4) are over-constrained, and the remaining ones (2, 5, and 6) are tight but solvable. Table 2 shows the performance of BT on data set 1 for various run 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	84961180	316839736	1813726855	3584407690	21569911975	86996547613

Table 2. *Performance of BT for various CPU run-times.*

We repeated our experiments 500 times for all stochastic search procedures. 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 our data, as shown in Figure 3 for data set 1, which justifies our decision. We experimented with different run-times for each run of each algorithm. We also experimented with different ratios used to increase the cutoff value in RGR and RDGR. We compare the performance of the algorithms using the following criteria:

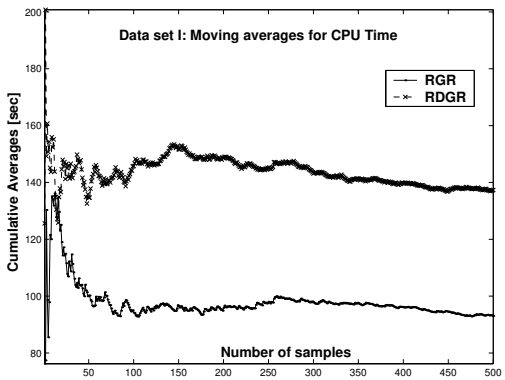

Fig. 3. *Moving average for CPU run-times for data set 1.*

1. *Solution quality distributions* (SQD) taking as reference the longest known solution for each data set, as recommended by Hoos and Stützle in [16]. 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 the longest solution and, the point 20% denotes a solution that is 20% shorter that the longest solution.
2. *Descriptive statistics* of all the solutions found, for all search techniques.
3. *95% confidence interval* of the mean improvement using 25 mean sample points, each sub-sample being of size 20. The confidence interval was computed using a t-distribution. The improvements of RDGR with respect to an algorithm A are computed as:

$$\text{Improvement}(X) = \frac{X(A) - X(\text{RDGR})}{X(A)} \tag{2}$$

where X is deviation from the best known solution in percentage. Table 3 reports the improvements of RDGR over RGR and ERA.

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. Results are shown in Figures 4, 5, 6, and 7, and Table 4.
- Data set 5 as a representative of a tight but solvable problem. Results are shown in Figures 8, 9, 10, and 11, and Table 5.

We also evaluated all the search techniques on randomly generated problems, generated with the model B type generator of of [17]. We generated three types of randomly generated problems, each containing 100 instances and each instance run for 3 minutes:

- The first type of randomly generated problems (R1) are *under-constrained* binary CSPs with 40 variables, uniform domain size of 20 values, 0.5 constraint proba-

Data set	Improvements over RGR			Improvements over ERA		
	LL	Average	UL	LL	Average	UL
1	1.83	2.23	2.63	45.47	46.26	47.06
2	1.19	1.48	1.78	-5.64	-5.17	-4.69
3	2.61	2.94	3.27	30.05	32.37	43.69
4	1.03	1.35	1.66	24.71	26.70	28.70
5	0.61	0.84	1.08	-3.54	-3.38	-3.23
6	0.86	1.15	1.45	-2.47	-1.91	-1.36

LL: lower limit of the confidence interval.
UL: upper limit of the confidence interval.

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

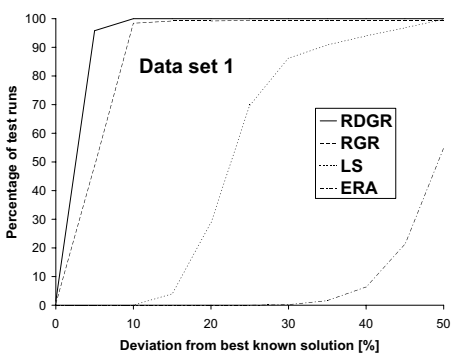

Fig. 4. *SQD for data set 1 (500 runs, 10 min each).*

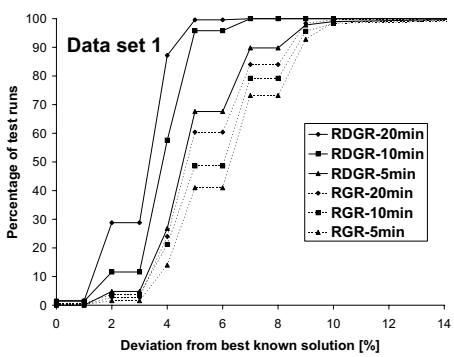

Fig. 5. *RGR and RDGR over different run-times for data set 1 (500 runs).*

bility, and 0.2 constraint tightness. We give their results in Figures 12, 13 and 14.

– The second type of randomly generated problems (R2) are *over-constrained* binary CSPs with 40 variables, uniform domain size of 20 values, 0.5 constraint probability, and 0.5 constraint tightness. We give their results in Figures 15, 16, 17, and 18.

– 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 constraint probability, and 0.36 constraint tightness. They were split into two sets, each of 100 instances. The first set (R3) are solvable, while the second set (R4) are not solvable. We give their results in Figures 19, 20, 21 (RDGR), 22 (RDGR), 23 (RGR), and 24 (RGR).

Below we report our observations:

Improvement of RDGR over RGR: Figures 4, 5, 8, 9, 12, 15, 19, and 20 show that RDGR clearly improves upon RGR. In Figures 4 and 8, RDGR has greater probability of finding solutions up to 10% relative solution size. After that value, RDGR

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
RDGR	59.66	60	60	0.77	58	62
RGR	58.27	58	58	2.83	23	62

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

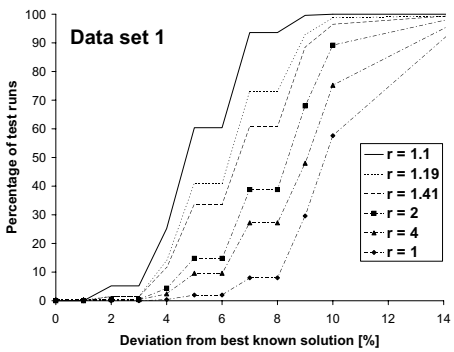

Fig. 6. *SQDs of RGR with different ratios for cut-off value (500 runs, 5 min each).*

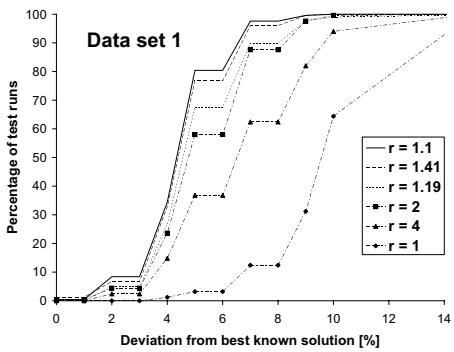

Fig. 7. *SQDs of RDGR with different ratios for cutoff value (500 runs, 5 min each).*

and RGR have similar performances. In Figures 5 and 9, RDGR consistently performs better than RGR over different run-times. The positive improvements in Table 3 show that RDGR performs better than RGR over all the GTA data sets.

Best results on the ratio used to increase the cutoff: In accordance with [2], Figures 6, 10, 13, 17, 23, and 24 show that a value of r=1.1 is the best among the values tested for RGR. While, for RGR, this optimal ratio does not change with the problem type (i.e., GTA vs. random problem), it does for RDGR. For the GTA problem, it is r=1.1 (Figures 7 and 11). For randomly generated problems, it is r=2 (Figures 14, 18, 21, and 22).

Improvement of RDGR over BT: Tables 4 and 5 show that the maximum value of the solution sizes produced by RDGR is clearly greater than that of the solution sizes produced by BT. However, due to its stochastic nature, RDGR suffers from high instability in its solution quality. On randomly generated problems also, RDGR outperforms BT (Figures 12, 15, 19, and 20).

Superiority of RDGR over LS: The performance of RDGR is clearly superior to that of LS (see Tables 4 and 5, and Figures 4, 8, 12, 16, 19, and 20). Although this 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.

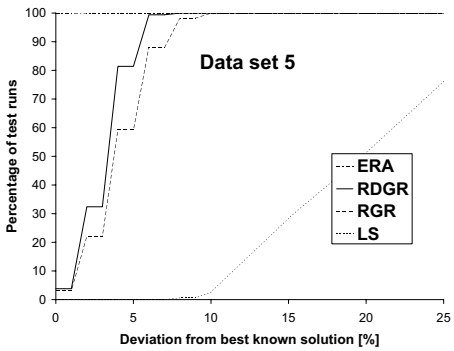

Fig. 8. *SQD for data set 5 (500 runs, 10 min each).*

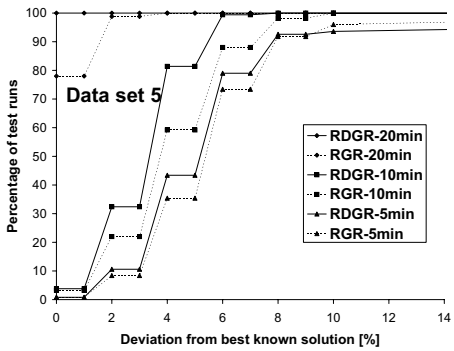

Fig. 9. *RGR and RDGR over different run-times for data set 5 (500 runs).*

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
RDGR	52.17	52	52	0.78	50	54
RGR	51.70	52	52	1.04	49	54

Table 5. *Statistics of solution size (500 runs, 10 min each).*

Superiority of RDGR over ERA on over-constrained problems: On over-constrained problems (Figures 4, 15, and 16 and Table 3), the deadlock phenomenon prevents ERA from finding solutions of quality comparable to those found by the other techniques [9–11]. 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 8 and 12), 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 (Figures 4 and 16) RDGR, RGR, BT and LS are superior to ERA due to the deadlock phenomenon. At the phase transition (Figures 19 and 20), 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: Due to their stochastic nature, RDGR and RGR techniques show a high instability in their solution quality. However, the standard deviation column of Tables 4 and 5 show that RDGR is relatively more stable than RGR.

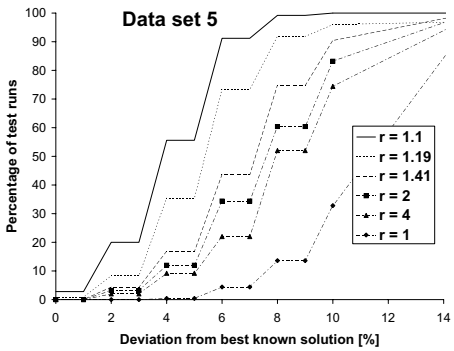

Fig. 10. *SQDs of RGR with different ratios for cutoff value (500 runs, 5 min each).*

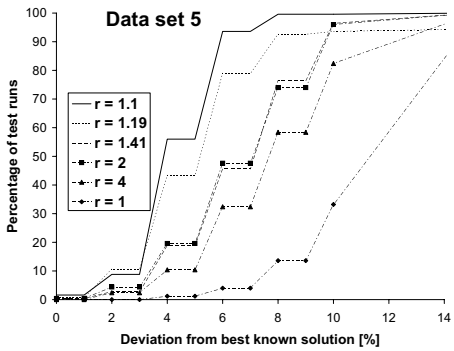

Fig. 11. *SQDs of RDGR with different ratios for cutoff value (500 runs, 5 min each).*

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

Larger number of restarts in RDGR: On data set 1, the average number of restarts is 74.5 for RDGR and 16.7 of RGR. On data set 5, the average number of restarts is 56.9 for RDGR and 22.4 for RGR. This confirms our expectations stated in Section 3.2 that RDGR performs more restarts than RGR.

The following three statements, where $\succ$ denotes an algorithm dominance over another, summarize the behavior of the 5 search strategies, also shown in Table 6:

- On solvable instances: ERA $\succ$ RDGR $\succ$ RGR $\succ$ BT $\succ$ LS
- On over-constrained instances: RDGR $\succ$ RGR $\succ$ BT $\succ$ LS $\succ$ ERA
- At the phase transition: 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. While BT is stable, it suffers from thrashing. LS is vulnerable to local optima. ERA shows difference in performance with different problem types. ERA has an amazing ability to solve under-constrained problems. However, ERA's performance degrades on over-constrained problems due to the deadlock phenomenon. This same deadlock phenomenon may be affecting ERA at the phase transition. Restart strategies effectively prevent thrashing, but their solution quality is highly variable. RGR operates by increasing cutoff values at every restart, which makes is more increasingly vulnerable 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. In the future, we plan to study the following directions:

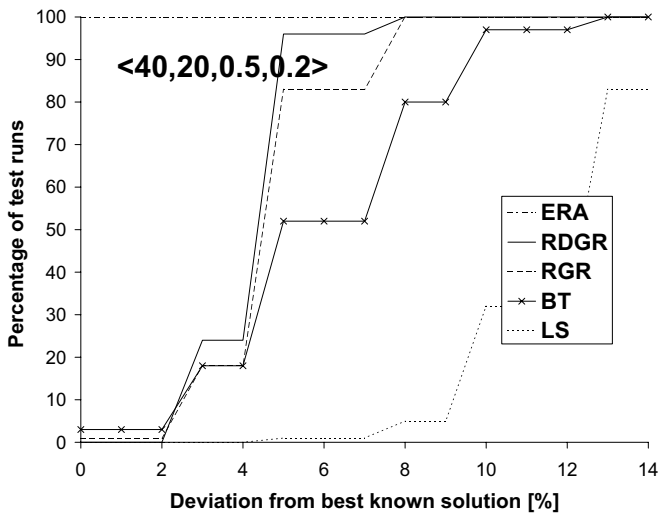

Fig. 12. *SQDs of data set R1 (100 instances, 3 min each).*

1. Validate our findings on other real-world case-studies. And,
2. 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), Madison, Wisconsin (1998) 431–437
2. Walsh, T.: Search in a small world. In: Proc. of the 16 th IJCAI. (1999) 1172–1177
3. 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)
4. 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)
5. Glaubius, R., Choueiry, B.Y.: Constraint Modeling and Reformulation in the Context of Academic Task Assignment. Poster presentation at the Fifth International Symposium on Abstraction, Reformulation and Approximation (SARA 2002) (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

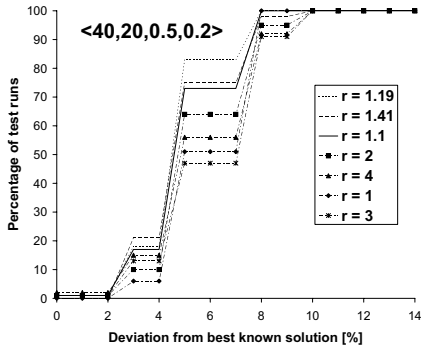

Fig. 13. *RGR with different ratios for data set R1 (100 instances, 3 min each).*

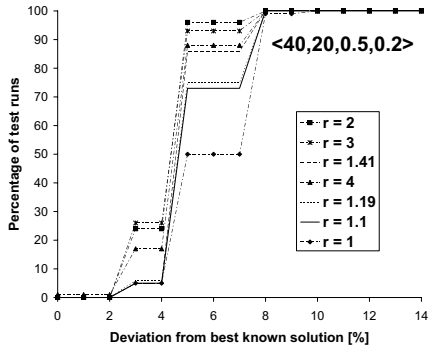

Fig. 14. *RDGR with different ratios for data set R1 (100 instances, 3 min each).*

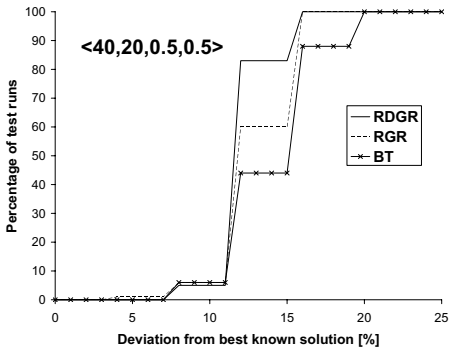

Fig. 15. *SQDs of RDGR, RGR, and BT for data set R2 (100 instances, 3 min each).*

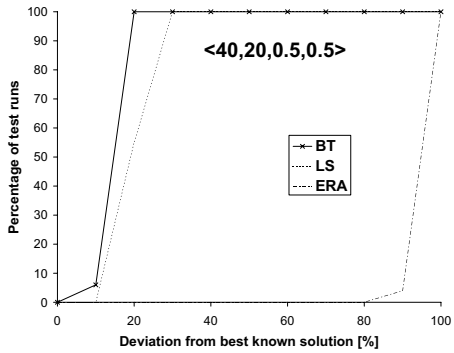

Fig. 16. *SQDs of BT, LS, ERA for data set R2 (100 instances, 3 min each).*

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. 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
10. Zou, H.: Iterative Improvement Techniques for Solving Tight Constraint Satisfaction Problems. Master's thesis, Department of Computer Science & Engineering, University of Nebraska-Lincoln (2003)
11. 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
12. 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

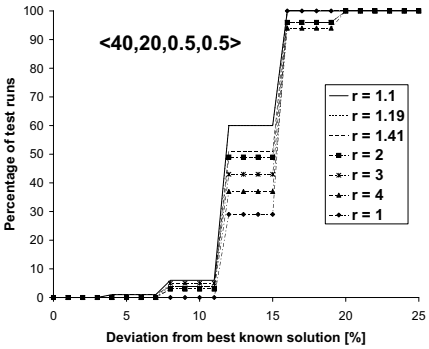

Fig. 17. *RGR with different ratios for data set R2 (100 instances, 3 min each).*

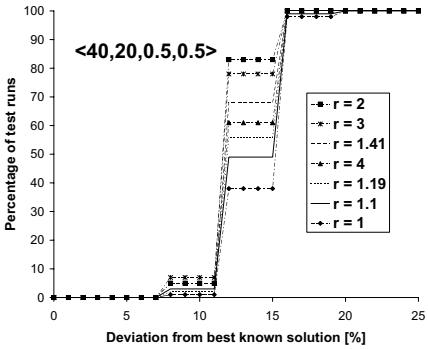

Fig. 18. *RDGR with different ratios for data set R2 (100 instances, 3 min each).*

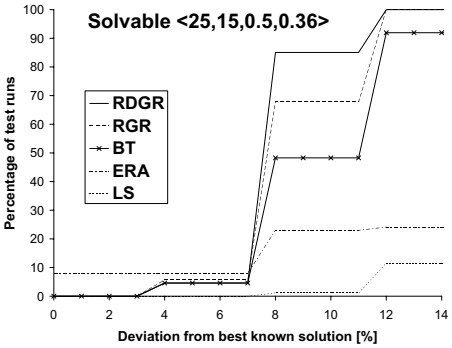

Fig. 19. *SQD for the data set R3 (100 instances, 3 min each).*

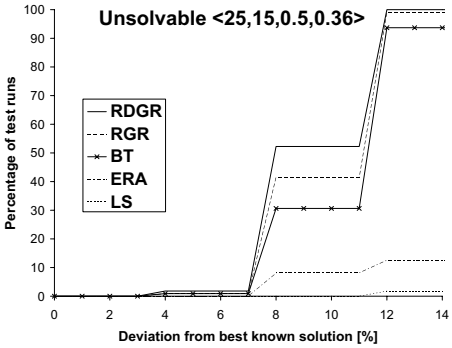

Fig. 20. *SQD for the data set R4 (100 instances, 3 min each).*

13. Barták, R.: On-Line Guide to Constraint Programming. kti.ms.mff.cuni.cz/˜bartak/constraints (1998)
14. Liu, J., Jing, H., Tang, Y.: Multi-agent oriented constraint satisfaction. Artificial Intelligence **136** (2002) 101–144
15. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of las vegas algorithms. In: Israel Symposium on Theory of Computing Systems. (1993) 128–133
16. Hoos, H., Stützle, T.: Stochastic Local Search Foundations and Applications. Morgan Kaufmann (2004) Forthcoming.
17. van Hemert, J.I.: RandomCSP: generating constraint satisfaction problems randomly. homepages.cwi.nl/ jvhemert/randomcsp.html (2004)

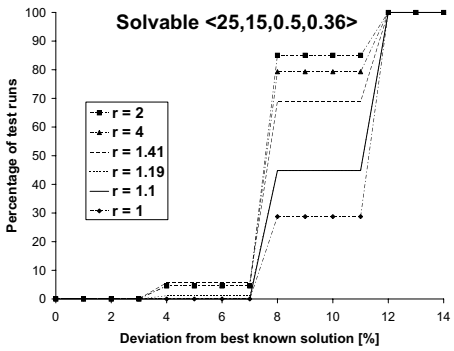

Fig. 21. *SQDs of RDGR with different ratios for cutoff value (100 instances, 3 min each).*

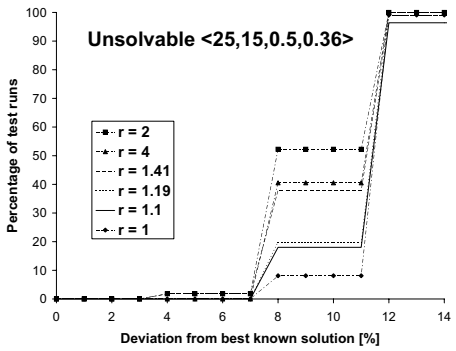

Fig. 22. *SQDs of RDGR with different ratios for cutoff value (100 instances, 3 min each).*

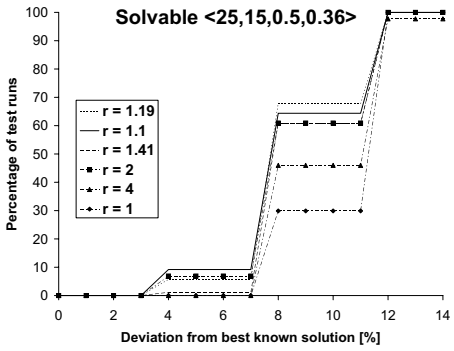

Fig. 23. *SQDs of RGR with different ratios for cutoff value (100 instances, 3 min each).*

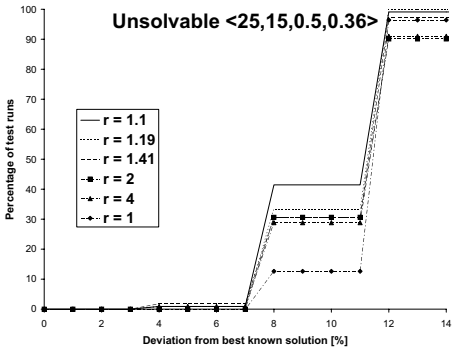

Fig. 24. *SQDs of RGR with different ratios for cutoff value (100 instances, 3 min each).*

	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

Table 6. *Comparing the behaviors of search strategies in our context.*