Characterizing the Behavior of a Multi-Agent Search by Using it to Solve a Tight, Real-World Resource Allocation Problem

Hui Zou and Berthe Y. Choueiry

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
{hzou,choueiry}@cse.unl.edu

Abstract. In this paper, we extend the empirical study of a multi-agent search method (ERA) for solving a Constraint Satisfaction Problem (CSP) [15]. We compare the performance of this method to that of a local search (LS) and a systematic (BT) search for solving the assignment of Graduate Teaching Assistants (GTA) to academic tasks. This real-world application, tight and often over-constrained, allows us to discover strengths and shortcomings of this multi-agent search, which would not have been possible otherwise. We show that for solvable, tight CSPs, ERA clearly outperforms both LS and BT, as it finds a solution when the other two techniques fail. However, for over-constrained problems, the multi-agent search method degenerates in terms of stability and the quality of the solutions reached. We identify the source of this shortcoming and characterize it as a deadlock phenomenon. Further, we discuss possible approaches for handling and solving deadlocks.

1 Introduction

Constraint Satisfaction has emerged as a powerful paradigm for modeling and solving large combinatorial problems. Scheduling and resource allocation are some of the earliest application areas of this technology [6, 3]. In this document, we discuss a specific application of constraint processing to a real-world problem. This is the assignment of Graduate Teaching Assistants (GTA) to courses in the Department of Computer Science and Engineering of the University of Nebraska-Lincoln. The idea for this particular application was found on the web page of Rina Dechter at the University of California, Irvine. This application is in fact a critical and arduous responsibility that the department's administration has to handle every semester.

In this paper we conduct a comparative study of the behavior of various search algorithms for solving this problem. Search algorithms for solving CSPs are usually classified into two main categories: local repair and systematic algorithms. Recently, Liu et al. [15] proposed a competitive multi-agent based search where agents communicate *indirectly* through a common environment. On the surface, this technique appears to be a variation of local search. However, they differ slightly as we argue in Section 2.1. In order to understand and characterize the behavior of this multi-agent search, we compare it to two other approaches we implemented to solve this problem: a systematic backtrack search and a hill-climbing local search. We stress that our investigations are motivated by, and focused on, the GTA problem of which we have collected a few real data-samples. Working on this practical application has allowed us to identify the advantages and shortcomings of the multi-agent search, which would not have been possible if we were solving toy problems, and surely difficult if we were using randomly-generated problems. The behaviors we identify and describe here should be directly applicable to general CSPs (i.e., beyond our particular application), and we are currently validating our conclusions on randomly generated non-binary CSPs.

The results of our study can be summarized as follows. The multi-agent approach exhibits an amazing ability to avoid local optima. We trace this ability to its fine-grain and de-centralized control mechanism in which agents try to *selfishly* realize their individual goals while communicating indirectly through the environment. As a result, the multi-agent approach can solve tight CSPs when the other two approaches fail. However, with unsolvable problems, its behavior becomes erratic and unreliable and results in a deadlock state. We trace this shortcoming to the decentralized control (the very same feature that constitutes the strength of this approach) and also to the lack of direct inter-agent communications. We argue that deadlock is actually a precious feature that allows us to identify conflicts, which is **NP**-hard in general. We show that when we try to avoid the deadlock phenomenon by adding global control to the multi-agent schema, we destroy its immunity to local optima. We argue that conflict resolution requires negotiation mechanisms among the agents, and thus is heuristic and problem-dependent.

This paper is structured as follows. Section 2 introduces multi-agent based search, the GTA problem, and the ERA model. Section 3 describes five experiments, summarizes our observations, and discusses the relative performance of the methods tested. Section 4 discusses possible approaches to avoid the deadlock phenomenon and presents two extensions to ERA. Finally, Section 5 provides directions for future research.

2 Background

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 $\mathcal{C}$ a set of constraints that restricts the acceptable combination 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.

In this section, we summarize the principle of the multi-agent based search, describe the features of our particular application and its modeling as a CSP, and finally explain how we apply this new search mechanism to solve our application.

2.1 A multi-agent-based method

A multi-agent system is a computational system in which several agents interact and work together in order to achieve a set of goals. Inspired by swarm intelligence, Liu et al. [15] proposed the ERA algorithm (Environment, Reactive rules, and Agents), which a search method for solving CSPs. In ERA, every variable is represented by a single, independent agent. A two-dimensional grid-like environment inhabited by the agents corresponds to the domains of variables. The final positions of the agents in this environment constitute the solution to a CSP. Each agent moves to the position that is most desirable given the constraints and the positions of the other agents, *regardless of whether this move improves or deteriorates the quality of the global solution*. The search stops when all agents are in positions that satisfy all applicable constraints.

Whether it is pertinent to rename the variables as agents and view the CSP as multi-agent system is not the topic of this paper. We simply remain faithful to the terminology proposed by the original authors [15]. In fact, ERA can be viewed as a generalization of local search. However, it differs from local search in some subtle ways as we try to explain below. In local search, moving from one state to another typically involves changing the assignment of one (or two) variables, thus the name *local search* [5]. In ERA, any number of variables can change positions at each step, each agent choosing its own most convenient position. Local search uses an evaluation function to assess the quality of a given state, where a state is a global but possibly inconsistent solution to the problem. This evaluation function is a *global* account of the quality of the state, typically computed as the *total* number of broken constraints for the whole assignment. In ERA, every agent applies the evaluation function *individually*, typically computing the number of the broken constraints that apply to the particular agent. The individual values of the evaluation function for the agents are *not* combined to give a global account of the quality of the state. Thus, ERA appears to de-centralize the control for selecting the new positions of the individual agents. Local search transitions from one state to the next in an attempt to achieve a *global goal*. It is thus directly applicable to optimization problems. In ERA every agent strives to achieve its own *local goal*. The search succeeds and stops when every agent is in a legal position. It is thus most suited to model satisfaction problems. The original paper on this technique encompasses an extensive comparison with other known distributed search techniques.

2.2 Graduate Teaching Assistants (GTA) problem

Below we give a quick description of this application. A detailed description of the problem, the constraints, and the solution obtained by backtrack search can be found in [8]. In this application, we are 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. In this problem, we model the courses as variables and the GTAs as values. Typically, each semester a pool of 25 to 40 GTAs must be assigned as graders or instructors to the majority of courses offered during that semester. In the past, this task has been performed by hand by several members of the staff and faculty. Tentative schedules were iteratively refined and updated based on feedback from other faculty members and the GTAs themselves, in a tedious and error-prone process dragging over 3 weeks. It was quite common that the final hand-made assignments still contained a number of conflicts and inconsistencies, which negatively affected the quality of our academic program. For example, when a course is assigned a GTA who has little knowledge of the subject matter, the course's instructor has to take over a large portion of the GTA's job and the GTA has to invest considerable effort in adjusting to the situation. Moreover, students in the course may receive diminished feedback and unfair grading. Our efforts in modeling and solving this problem using constraint processing techniques have resulted in a prototype system under field testing in our department since August 2001 [8]. This system has effectively reduced the number of obvious conflicts, thus yielding a commensurate increase in course quality. It has also decreased the amount of time and effort spent on making assignments and has gained the acceptance and approval of our faculty and students.

The CSP model includes a number of unary, binary and non-binary constraints that govern the assignments. These constraints model the competence of the GTAs (a set unary constraints), their availability (usually unary and binary constraints), and their employment capacity (typically, global constraints). These constraints must meet the schedule and specifications of the classes, and the assignment of a GTA must not exceed the total load of the courses to which he or she is assigned during a semester. In addition to these hard constraints, candidate GTAs express preferences for specific courses on a scale from 0 to 5 that we try to maximize. In our experience, instances of this application are always tight and often over-constrained. However, *this is not known a priori*. The goal of the GTA problem is to ensure GTA support to as many courses as possible. Consequently, the problem consists in finding the *maximal consistent partial-assignment*, where a solution needs to meet all constraints while maximizing the number of covered courses, as a primary optimization criterion, and the quality of the solution in terms of GTA preferences as a secondary one. Here we need to stress that GTA is not a MAX-CSP [7]. In MAX-CSP, all constraints are soft and the goal is to maximize the number of satisfied constraints. Thus, the solution of a MAX-CSP problem may be inconsistent. Neither our application nor our modeling of it is a MAX-CSP because all the constraints considered in the assignments are hard and are not allowed to be broken.

We have collected four real-data instances of the GTA problem from different academic semesters. These are annotated 'O' in Table 1, denoting that the data are original. These instances may or may not be solvable. Since all the problems were difficult to solve (and two of them are indeed unsolvable), we boosted the available resources to transform these problems into solvable ones. To accomplish this, we added extra resources—*dummy* GTAs—into the data set. These problems are annotated with the mark 'B' in Table 1, denoting that the corresponding problem is boosted, one dummy GTA at a time, until we can solve it.

Data Set	Mark	Domain Size	# variables	Problem Size
Spring2001b	B	35	69	3.5×10^{106}
	O	26	69	4.3×10^{97}
Fall2001b	B	35	65	2.3×10^{100}
	O	34	65	3.5×10^{99}
Fall2002	B	33	31	1.2×10^{47}
	O	28	31	7.3×10^{44}
Spring2003	B	36	54	1.1×10^{84}
	O	34	54	5.0×10^{82}

Table 1. *Problem instances tested.*

(The large domain size of the corresponding CSPs causes tremendous thrashing in systematic backtrack search.)

2.3 ERA model

An ERA system has three components: an Environment (E), a set of Reactive rules (R), and a set of Agents (A). Each variable is an agent. The position of an agent corresponds to the value assigned to this variable. The environment records the number of constraint violations of each agent's position. An agent moves according to its reactive rules. Two assumptions are made: (1) all agents have the same reactive rules, and (2) an agent can only move to positions in its own domain. In our implementation, agents move in sequence, but the technique can equally be in an asynchronous fashion.

The environment E is represented as a two-dimensional array that has n rows corresponding to the number of courses, and has $|D_{\max}|$ columns where $D_{\max}$ is the size of the largest domain. Fig. 1 illustrates the environment E of the GTA problem. An entry $e(i, j)$ in E refers to a position at row i (representing

course-1	(GTA1,3)	(GTA2,12)	(GTA4,12)	(GTA5, 15)	(GTA7,52)
course-2	:	:	:		
course-3	:	:	:	:	
	:	:	:	:	
	:	:	:	:	
course-n	(GTA2,9)	(GTA5,8)	(GTA16, 80)	(GTA21,18)	

Fig. 1. *The data structure of environment E.*

Agent i) and column j (representing the index j in the domain of the variable). The entry $e(i, j)$ stores a list of two values, namely *domain value* and *violation value*. Domain value, $e(i, j).value$, points to the data structure (i.e., object) of a GTA of position index j. Violation value, $e(i, j).violation$, is the number of broken constraints of the agent in the current assignment. A `zero position` is a position for the agent that does not break any of the constraints that apply to it. The current assignment of the agent is consistent with the other agents' assignments. Obviously, if agents are all in `zero position`, then we have a full, consistent solution. The information in E is updated when an agent changes position. The goal is to have each agent find its `zero position`. Liu et al. [15] define three reactive rules:

1. *Least-move*: The agent chooses the position with the minimal violation value and moves to it, breaking ties randomly.
2. *Better-move*: The agent chooses a position at random. If the chosen position has a smaller violation value than the current one, then the agent moves to it. Otherwise the agent keeps its current position.
3. *Random-move*: With a probability p, the agent randomly chooses a position to move to. This rule helps out the agent from getting stuck in local optima.

This architecture is reminiscent of that of the GENET algorithm [4]. However, the two approaches differ in the fact that GENET uses a computationally intensive weight function to assess assignments and a learning mechanism to avoid local optima.

ERA algorithm. The ERA algorithm works as follows. It builds the environment E, generates a random position for each agent, and moves the agents to these positions. Then ERA considers each agent in sequence. For a given agent, it computes the violation value of each possible position for the agent under consideration. If the agent is already in a `zero position`, no change is made. Otherwise, the agent applies the reactive rules to choose a new position and moves to it. Then, ERA moves to the next agent. The agents will keep moving according to the reactive rules until they all reach a `zero position` 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). We noticed that, in practice, the agents' ordering and their concurrency or synchronism do not affect the performance of the technique because of the agents' high reactivity. Since the violation value of each position of an agent under examination is updated at each run, the agent cannot stay in its current position unless this position remains a `zero position`, that is, the position is unchallenged by the remaining agents.

Each iteration of the ERA algorithm, one move per agent, has a time complexity of $\mathcal{O}(n^2 \times D_{\max})$. The space complexity is $\mathcal{O}(n \times D_{\max})$. Liu et al. [15] demonstrated ERA with two benchmark CSPs: the n-queen and coloring problems. Both problems have only binary constraints and the instances tested were solvable. In this paper, we examine the performance of the ERA in solving the more difficult, non-binary, over-constrained GTA problem. Before we describe our experiments, we summarize some possible behaviors of an agent and the rules that govern the behavior. We also review some observations presented by Liu et al. [15].

Possible behaviors and some observations Liu et al. [15] experimented with the following behaviors:

- `LR` is the combination of the *least-move* and *random-move* rules. The agent typically applies *least-move* and uses *random-move* with a probability p to get out of a local optimum.
- `BR` is the combination of the *better-move* and *random-move* rules. It is similar to `LR` except that it replaces *least-move* with *better-move*.
- `BLR` is the combination of the *better-move*, *least-move*, and *random-move* rules. The agent first applies *better-move* to find its next position. If it fails, it applies `LR`.
- `rBLR`: First, the agent applies r times the rule *better-move*. If it fails to find one, it applies the `LR` rule.
- `FrBLR`: The agent applies `rBLR` for the first r iterations and then it applies `LR`, typically $r = 2$.

Liu et al. [15] further reported the following observations. (1) The cost of *better-move* in CPU time is much smaller than that of *least-move*, which requires evaluating all positions. (2) The chance of successfully finding a position to move to with *better-move* is quite high. (3) *Better-move* allows most agents to find better positions at the first step. (4) `FrBLR` outperforms `rBLR`, which in turn outperforms `LR` in terms of runtime.

2.4 Our local search and systematic search

Local search: Our local search (LS) is a hill-climbing search using the min-conflict heuristic for value selection [16]. It begins with a complete assignment (not necessarily consistent) and tries to improve it by changing inconsistent assignments in order to reduce the number of constraint violations. We propagate the effect of consistent assignments over the domains of the variables with inconsistent assignments. This design decision allows us to handle effectively non-binary constraints. We implement local search in a greedy fashion in the sense that we do not backtrack over consistent assignments. Moreover, we apply a random-walk strategy to escape from local optima [2]. With a probability $(1 - p)$, we choose the value of a variable using the min-conflict heuristic, and with probability p we choose this value randomly. Following the indications of [2], we choose $p = 0.02$. Further study on the choice of the value of p for the GTA problem will be reported in [19]. Furthermore, we use random restarts to break out of local optima.

Systematic search: We utilize systematic search techniques based on depth-first backtrack search to solve the GTA problem. In our implementation (BT), forward checking [17] and branch-and-bound mechanisms are integrated into the search strategy. A full look-ahead strategy would drastically increase the number of constraint checks while effectively yielding little filtering since the application has many mutex and global constraints (it is a resource allocation problem). As depth-first search expands nodes in a search path, we check if the expansion of the search path can improve on the current best solution. Once the current best solution cannot be improved, backtrack occurs. In addition, the dynamic variable and value ordering heuristics are applied in BT. Given that problems may be over-constrained we slightly modified search not to backtrack when a domain wipe-out occurs, but only when the current path cannot improve upon the current incumbent. Our implementation is described in detail in [8].

3 Empirical evaluation of ERA

We tested our implementation on known toy problems and the real-world examples shown in Table 1. First, we solved the 100-queen[1] problem and the Zebra puzzle with different agent behaviors. Then, using `FrBLR` as the default behavior of agents, we solved the eight instances of the GTA problem, which include solvable and over-constrained cases. We conducted four main experiments: we tested the behavior of ERA (Section 3.1), compared it to other search techniques (Section 3.2), observed the behavior of individual agents (Section 3.3), and identified a shortcoming of ERA which we call the deadlock phenomenon (Section 3.4). Observations follow each experiment.

3.1 Testing the behavior of ERA

In the following experiment, we recorded the number of agents reaching `zero position` at every iteration as shown in Fig. 2.

Experiment 1 Solve the GTA problem for the data-set Fall2001b using `LR`, `BLR` and `FrBLR`.

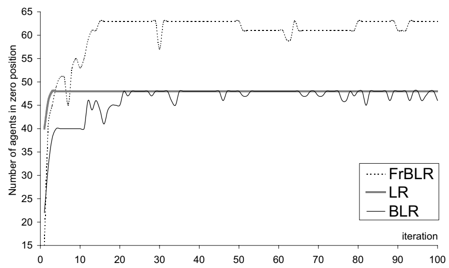

Fig. 2. *Agents in zero position for Fall2001b.*

We report the following observations:

[1] The n-queen problem is not particularly well-suited for testing the performance of search. However, we use it only to be on common grounds with Liu et al. [15]. Indeed, they conducted their tests on the n-queen and the coloring problems, and drew their conclusions from the n-queen problem.

– The number of agents in `zero position` does not grow strictly monotonically with the number of iterations, but may instead exhibit a 'vibration' behavior. This is contrasted with the 'monotonic' behavior of hill-climbing techniques and illustrates how ERA allows agents to move to non-`zero positions` to avoid local optima.
– Fig. 2 shows that the curves for `BLR` and `FrBLR` 'vibrate,' highlighting an unstable number of agents in `zero position` across iterations, while `LR` quickly reaches a stable value. `FrBLR`, which combines `LR` and `BLR`, achieves the largest number of agents in `zero position`.
– After the first few iterations, a large number of agents seem to reach their `zero position` with `LR` than with `BLR`. However, the problem seems to quickly become 'rigid' and the total number of agents in `zero position` becomes constant.

In the GTA problem, as with the toy problems, `FrBLR` seems to yield the best results. We adopt it as the default behavior for all agents.

3.2 Performance Comparison: ERA, LS, and BT

In order to gather an understanding of the characteristics of ERA, we compared its performance with that of two other search strategies we previously implemented. The first strategy is a systematic, backtrack search (BT) with dynamic variable ordering fully described in [8]. The second strategy is a hill-climbing, local search (LS) to be documented in [19]. LS is a combination of constraint propagation to handle non-binary constraints and the min-conflict random walk algorithm as presented in [2].

As stated in Section 2.2, the GTA problem is over-constrained. We try to find the assignment that covers the most tasks and, for an equal solution length, one that maximizes the arithmetic or geometric average of the preference values of the assignments. In all three searches (i.e., ERA, BT, and LS), we store the best solution found so far and update it only when the quality of a new solution exceeds that of the one currently stored. As a result, the search behaves as an anytime algorithm.

Experiment 2 Solve the GTA problem for the real-world data of Fall2001b, Fall2002 and Spring2003 using ERA, BT search [8], and a hill-climbing, local search technique [19]. Since all the problems were difficult to solve (and two of them are indeed unsolvable), we boosted the available resources to transform these problems into solvable ones. To accomplish that, we added extra resources—*dummy GTAs*—into the data set. The results are shown in Tab. 2 and Figs. 3 and 4.

We adopted the following working conditions:

– The quality of the solution reached by BT search did not improve after the first 20 seconds, even when we let the search ran for hours or days. A careful observation of the backtracking showed that the shallowest tree-level

reached was as deep as 70% of the number of variables (i.e., the maximum depth of the tree). This situation did not improve much over time. This can be traced to the large domain size of the variables in this application, which systematically prevents a large portion of the search space from being explored at all. This problem could not be avoided even by using randomized variable ordering.
– The maximum iteration number for LS and ERA is 200. This corresponds to a few minutes of run time for LS and a couple of minutes of ERA.
– We increased the number of dummy GTAs, one at a time, until one of the search techniques can find a solution. The solvable instance thus obtained may have more GTAs than are actually needed.
– The ratio of `total capacity` and `total load` (shown in column 7 in Tab. 2) is an indicator of the tightness of the problem. When the ratio is less than 1, the instance is over-constrained and guaranteed not solvable. Otherwise, it may or may not have a full solution.

We compared the search techniques according to five criteria:

1. *Unassigned courses*: the number of courses that are not assigned a GTA (col. 8, 13, and 18 in Tab. 2). The primary goal is to minimize this value.
2. *Solution quality*: the geometric average of the preferences, with values $\in [1, 5]$ (col. 9, 14, and 19 in Tab. 2). A larger value indicates a better solution.
3. *Unused GTAs*: the number of GTAs not assigned to any task (col. 10, 15, and 20 in Tab. 2). This value is useful to analyze why certain resources are not used by the search mechanism. This constitutes useful feedback in the hiring process.
4. *Available resources*: the cumulative value of the remaining capacity of all GTAs after assignment (col. 11, 16, and 21 in Tab. 2). This provides an estimate of whether a search strategy is wasteful of resources.
5. *CC*: the number of constraint checks, counted using the convention of Bacchus and Van Beek [1] (col. 12, 17, and 22 in Tab. 2).

We report the following observations:

– Only ERA is able to find a full solution to all solvable problems (column 18 of Tab. 2). Both BT and LS fail for all these instances. In this respect, ERA clearly outperforms the other two strategies and avoids getting stuck in useless portions of the search space.
– When the ratio of total capacity to total load is greater than 1 (the problem may or may not be solvable), ERA clearly outperforms BT and LS. Conversely, when the ratio is less than 1 (problem is necessarily over-constrained), ERA's performance is the worst, as shown in Fig. 3. Indeed, we make the conjecture that ERA is not a reliable technique for solving over-constrained problems.
– On average (see Fig. 4), LS performs much fewer constraint checks than ERA, which performs fewer constraint checks than BT.
This feature of LS is useful when checking constraints (e.g., non-binary constraints), but is a costly operation.

Data Set								BT					LS					ERA					
	Original/Boosted	Solvable?	#GTAs	#Courses	Total capacity	Total load	Ratio = $\frac{\text{TotalCapacity}}{\text{TotalLoad}}$	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	CC ($\times 10^8$)	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	CC($\times 10^8$)	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	CC($\times 10^8$)	Row reference
Spring2001b	B	√	35	69	35	29.6	1.18	6	4.05	2	6.5	3.77	5	3.69	0	6.4	0.87	0	3.20	0	5.3	0.18	**A**
	O	×	26	69	26	29.6	0.88	16	3.79	0	2.5	4.09	13	3.54	0	0.9	0.39	24	2.55	8	8.3	7.39	**B**
Fall2001b	B	√	35	65	31	29.3	1.06	2	3.12	0	2.5	1.71	4	3.01	0	3.8	0.33	0	3.18	1	1.9	2.68	**C**
	O	√	34	65	30	29.3	1.02	2	3.12	0	1.5	2.46	4	3.04	1	3.7	0.10	0	3.27	0	0.8	1.15	**D**
Fall2002	B	√	33	31	16.5	13	1.27	1	3.93	0	3.5	2.39	2	3.40	0	5.0	0.85	0	3.62	2	3.0	0.02	**E**
	O	×	28	31	11.5	13	0.88	4	3.58	0	1.8	2.56	4	3.61	0	2.0	0.16	8	3.22	1	2.0	0.51	**F**
Spring2003	B	√	36	54	29.5	27.4	1.08	3	4.49	2	4.2	1.17	3	3.62	0	3.9	0.32	0	3.03	1	2.8	0.49	**G**
	O	√	34	54	27.5	27.4	1.00	3	4.45	0	2.2	1.53	4	3.63	0	3.3	1.42	0	3.26	0	0.8	0.14	**H**
Reference	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	

Table 2. *Comparison between BT, LS, and ERA.* (O/B indicates whether the instance is original or boosted. CC is # of constraint checks.)

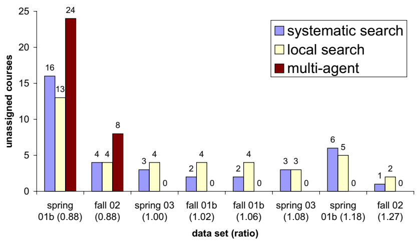

Fig. 3. *Unassigned courses*

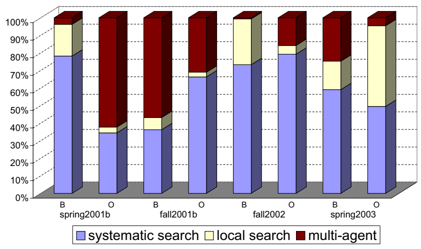

Fig. 4. *Constraint checks*

– ERA leaves more GTAs unassigned than BT or LS (col. 10, 15, and 20 in Tab. 2), which raises concerns about its ability to effectively exploit the available resources.

  In particular, for Spring2001b (O), 8 GTAs remain unused. This alarming situation prompted us to closely examine the solutions generated, which yielded our identification of the *deadlock phenomenon* discussed in Section 3.4. Finally, we compared the behavior of ERA on solvable and unsolvable problems in terms of the number of agents in `zero position` per iteration. The solvable problems are: Spring2001b (B), Fall2001b (B/O), Fall2002 (B), and Spring2003 (B/O) (Fig. 5). The unsolvable ones include Spring2001b (O) and Fall2002 (O) (Fig. 6).

– Neither the basic LS nor ERA (i.e., without restart strategies) includes a mechanism for improving the quality of the solution in terms of GTA preferences, which is the secondary optimization criterion. Indeed, the quality of the solutions found by BT is almost consistently higher. However, this length

of the solutions found by ERA on solvable instances, which is the primary optimization criterion, is significantly larger to both BT and LS.
- Figs. 5 and 6 show that the performance of ERA is more stable when solving solvable problems than when solving unsolvable problems.

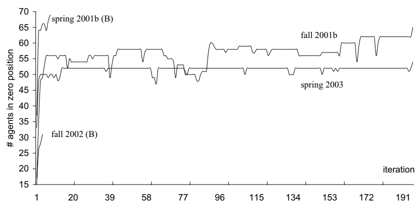

Fig. 5. *ERA performance on solvable problems*

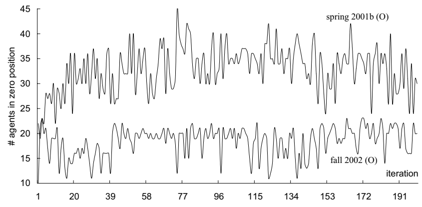

Fig. 6. *ERA performance on unsolvable problems*

3.3 Observing behavior of individual agents

Tracking the positions of individual agents at various iterations, we observed the three types of agent movement shown in Fig. 7. In this figure, we used the index of the agent's position to indicate its assigned value. The three types of movements are the following:

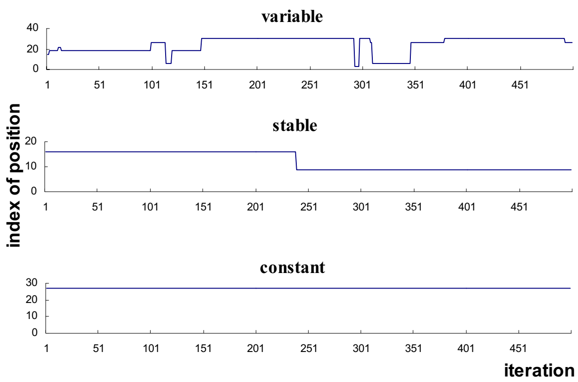

Fig. 7. *Types of agent movement*

- Variable: the agent changes its position relatively frequently and fails to find its `zero position`.
- Stable: the agent rarely changes its position (once or twice).

– Constant: the agent finds a `zero position` at the beginning of the search, and never changes it.

Experiment 3 We set the maximum number of iterations to 500 and tracked the positions of agents over the entire data set, grouped into solvable and unsolvable instances.

We observed the following:

– In solvable instances, most agents are stable, a few are constant, and none of the agents is variable.
– In unsolvable instances, most agents are variable, a few are stable, and none of the agents is constant.

3.4 The deadlock phenomenon

On our two unsolvable instances of Tab. 2 (i.e., Spring2001b (O) and Fall2002 (O)), ERA left some tasks unassigned (col. 18) and some resources unused (col. 20), although, in principle, better solutions could be reached. By carefully analyzing these situations, we uncovered the *deadlock phenomenon*, which is a major shortcoming of ERA and may hinder its usefulness in practice. It is not our claim that the deadlock phenomenon is unique to ERA. It may also show up in other search algorithms. However, the fact that ERA exhibits this shortcoming was not noticed earlier.

Experiment 4 With the Spring2001b (O) data, we examined the positions of each agent in the state corresponding to the best approximate solution found, and analyzed the allocation of resources to tasks.

The best approximate solution for this problem was found at iteration 197, with 24 courses unassigned and 8 GTAs unused. The total number of courses in this problem is 65. We observed that the unsatisfied courses can actually be serviced by the available, unused GTAs. ERA was not able to do the assignment for the following reason. There were several unsatisfied agents (i.e., courses) that chose to move to a position in their respective rows corresponding to the same available GTA, while this GTA could only be assigned to as many agents as its capacity would allow. This situation resulted in constraints being broken and none of the agents reaching a `zero position`. As a consequence, although agents moved to that position, none could be assigned that position, and the corresponding GTA remained unassigned. We illustrate this situation in Fig. 8. Each circle corresponds to a given GTA. Note that there is exactly one circle per GTA. Each square represents an agent. The position of an square on the circle is irrelevant and only useful for visualization purposes. There may be zero or more squares on a given circle. Blank squares indicate that the position is a `zero position` for the agent; these will yield effective assignments. The filled squares indicate that although the position is the best one for the agent, it results in some broken constraints. Thus it is not a `zero position`, and the actual assignment of the position to the agent cannot be made. The circles populated by several filled squares are GTAs that remain unused.

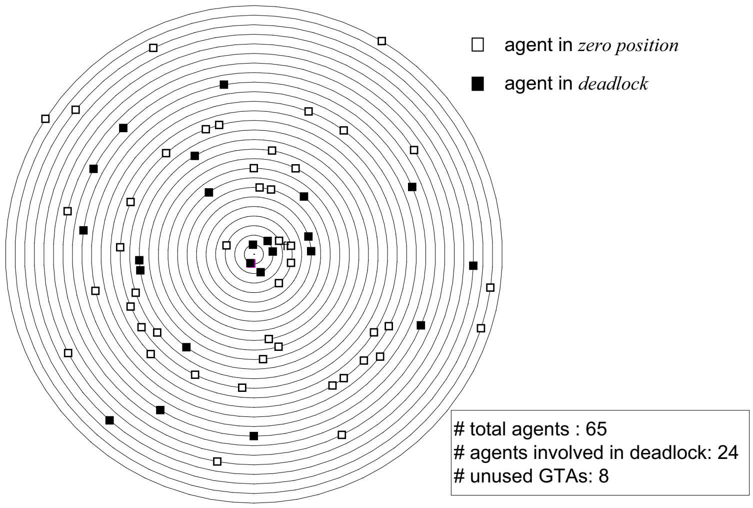

Fig. 8. *Deadlock state*

Definition 1. *Deadlock state:* When the only positions acceptable to a subset for agents are mutually exclusive, a deadlock occurs that prevents any of the agents in the subset from being allowed to move to the requested position.

None of the variables in a deadlock is instantiated, although some could be. Further, a deadlock causes the behavior of ERA to degrade. When some agents are in a deadlock state, one would hope that the independence of the agents would allow them to get out of the deadlock (or remain in it) without affecting the status of agents in `zero position`. Our observations show that this is not the case. Indeed, ERA is not able to avoid deadlocks and yields a degradation of the solution in the sense that it does not maximize the number of courses satisfied. Indeed, subsequent iterations of ERA, instead of moving agents out of deadlock situations, moved agents already in `zero position` out their positions and attempt to find other `zero positions` for them. The current best solution is totally destroyed, and the behavior of the system degrades.

This problem was not reported in previous implementations of ERA, likely because they were not tested on over-constrained cases. Further, it seriously hinders the applicability of this technique to unsolvable problems. It is important that ERA be modified and enhanced with a conflict resolution mechanism that allows it to identify and solve deadlocks. We discuss this issue in Section 4.

3.5 Discussion

From observing the behavior of ERA on the GTA problem, we conclude the following:

– *Better-move vs. least-move:* A key point in iterative-improvement strategies is to identify a good neighboring state. In ERA, this is achieved by the reactive rules. In Minton et al. [16], this is the min-conflict heuristic. We noticed that *better-move* provides more opportunities to explore the search space than *least-move* does, and avoids getting stuck in local optima. With *least-move*, an agent moves to its best position where it stays. This increases the difficulties of other agents and the complexity of the problem, which quickly becomes harder to solve.
– *Reactive behaviors:* Different behaviors significantly affect the performance of ERA. We found that `FrBLR` results in the best behavior in terms of runtime and solution quality. At the beginning of the search, *better-move* can quickly guide more agents toward their `zero position`. Then *least-move* prevents drastic changes in the current state while allowing agents to improve their positions. Finally *random-move* deals effectively with plateau situations and local optima.
– *Stable vs. unstable evolution:* As highlighted in Fig. 5 and 6, the evolution of ERA across iterations, although not necessarily monotonic, is stable on solvable problems and gradually moves toward a full solution. On unsolvable problems, its evolution is unpredictable and appears to oscillate significantly. This complements the results of Liu et al. [15] by characterizing the behavior of ERA on over-constrained problems, which they had not studied.

From comparing systematic, local and multi-agent search on the GTA problem, we identify three parameters that seem to determine the behavior of search, namely (1) the control schema, (2) the freedom to undo assignments during search, and (3) the way conflicts are solved and deadlocks broken. Below, we discuss the behavior of the three strategies we tested in light of these parameters. We expect this analysis to be generalizable beyond our limited considerations. Our analysis is summarized in Tab. 3.

	Goal	**Actions**	
	Control schema	**Undoing assignments**	**Conflict resolution**
ERA	*Local* + Immune to local optima – May yield instability	*Yes, anytime* + Flexible + Solves tight CSPs	*Non-committal* – Deadlock – Shorter solutions
LS	*Global* + Stable behavior – Liable to local optima	*No, greedy approach* + Quickly stabilizes – Fails to solve tight CSPs even with randomness & restart strategies	*Heuristic* + Longer solutions
BT	*Systematic* + Stable behavior – Thrashes	*Only when backtracking* + Quickly stabilizes – Fails to solve tight CSPs even with backtracking & restart strategies	*Heuristic* + Longer solutions

Table 3. *Comparing the behaviors of search strategies in our implementation.*

Control schema: Global vs. local. In ERA, each agent is concerned with, and focuses on, achieving its own local goal—moving to a minimal violation-value position. This increases the 'freedom' of an agent to explore its search space, which allows search to avoid local optima. As a result, the ERA has an inherent immunity to local optima. The global goal of minimizing conflicts of a state is implicitly controlled by the environment E, through which the agents 'communicate' among each other. This communication medium and local control schema of ERA are effective when the problem is solvable, but they fail when problems are over-constrained. Indeed, on unsolvable instances, ERA is unstable and causes oscillations.

This decentralized control should be contrasted with the centralized control of local search where the neighboring states are evaluated globally by a centralized function. Global control used in LS leads to a stable performance: the movement to a successor state is, in general, allowed only when the neighboring state reduces the global cost, such as the total number of broken constraints in the state. However, this kind of control overly restricts the movement of agents and the the search easily gets trapped in local optima, which is unlikely to be overcome even with random restarts [12].

In backtrack search, alternative solutions are examined in a systematic way. Generally speaking, we either expand a partial solution or we chronologically consider immediate alternatives to the last decision. Usually, we record the best solution found so far as an incumbent and update it only when a better solution is found. As a result, the quality of solutions improves with time and the search is typically stable. However, thrashing is the price to pay for the stability and completeness of search. We tested both heuristic and stochastic backtrack search [10] and found that backtracking never goes beyond the third of the depth of the tree on our problems. Random restart strategies and credit-based search are ways to avoid this thrashing, but they sacrifice completeness.

Freedom to undo assignments. Among the three strategies we tested so far (we are testing others), only ERA was able to solve our hard, solvable instances. This ability can be traced to its ability to undo assignments.

In ERA, an agent can undo its assignment as needed, even if it is a consistent one. In fact, no agent may remain in a given position unless this position is acceptable to all other agents, that is, it remains a `zero position` across iterations. This feature seems to be the major reason why ERA is able to solve successfully large, tight problems that resisted the other techniques we tested (i.e., the solvable instances of Tab. 2 were only solved by ERA).

In contrast, in both systematic and hill-climbing search, a value is assigned to the variable that claims it first, on a first-come, first-served basis. Our implementation of local search (a hill-climbing strategy with a combination of constraint propagation and a min-conflict heuristic for value selection) does not undo consistent assignments. However, more generally, in backtrack search and LS, assignments can be undone using backtracking and random-restart strategies, respectively. In our experiments, both backtracking and random-restarts failed to solve tight instances, due to the sheer size of the search space.

Conflict resolution and deadlock prevention. We identify two main approaches for search to deal with conflicts: (1) heuristic, based on some priority such as a 'first-come, first-served,' least commitment, fail-first principle, or using user-defined preferences, and (2) non-committal, where conflict sets are merely identified and handed either to the user or to a conflict resolution procedure [3].

When it is not able to solve a conflict (e.g., a resource contention in the case of a resource allocation problem), ERA *adopts a cautious approach and leaves the variables unassigned.* This yields the deadlock phenomenon encountered in over-constrained cases, introduced in Section 3.4 and discussed in Section 4. We believe that the non-committal strategy is more appropriate in practical settings because it *clearly delimits the sources of conflict* and makes them the responsibility of a subsequent conflict resolution process. We consider this feature of ERA to be particularly attractive. Indeed, conflict identification is a difficult task (perhaps **NP**-hard) and ERA may constitute the first effective and general strategy to approach this problem.

Both backtrack search and LS operate in a more resolute way: they heuristically assign values to as many variables as possible. As a result, when maximizing the solution length, as in the GTA problem, they end up finding solutions that are more competitive (i.e., longer) than ERA.

4 Dealing with the deadlock

While ERA is *not* a complete procedure, we were puzzled by its ability to quickly solve tight problems. However, in over-constrained problems, some agents may be always prevented by other agents from reaching a `zero position`. One can think of the deadlock phenomenon as a powerful feature of ERA since it allows us to identify and isolate conflicts. Conversely, one could think of it as a shortcoming of ERA since, in over-constrained cases, it yields shorter solutions than LS or BT. We identify four possible avenues for dealing with deadlocks.

1. *Direct communication and negotiation mechanism:* In ERA, agents exchange information indirectly, through the environment E, and have no explicit communication mechanism. The information that is passed is a summary of the state of the environment. Agents are not able to recognize each other's individual needs and thus are unable to establish coalition. One could investigate how to establish more effective, informative communications among agents, as in a truly multi-agent approach.
2. *Hybridization algorithms:* When a deadlock occurs in ERA, we could use the solution found as a seed for another search technique such as LS or BT. One could even imagine a portfolio of algorithms where various solvers, with various features and weaknesses, cooperate to solve a given difficult problem. We are working in this direction.

3. *Global control:* The decentralized control of ERA enables an agent to pursue the satisfaction of its own, local goal. However it also undermines the ability of the system to achieve cooperatively a common global goal (when such a goal exists but is not Pareto optimal). In Section 4.1 we investigate how to enhance ERA with global control and examine the advantages and shortcomings of our proposed strategy.
4. *Conflict resolution:* An over-constrained problem by definition, has no solution. Conflict resolution is thus necessarily heuristic and problem-dependent [13]. There are two main approaches to conflict resolution:
   - *Interactive:* In an interactive setting, the identified conflicts are `no-goods` that can be presented to the users and allow them to integrate their own judgment for conflict resolution. This is the case in our application: most conflict resolution is currently done interactively, which allows the integration of 'unquantifiable' constraints into the solutions.
   - *Automatic:* Soft constraints, preferences, and rules to relax constraints could be included in the model in order to solve conflicts automatically. Once a given conflict is identified and solved, a new problem is generated based on the modification of the initial one, and the problem solver is run on this new problem. This process repeats until all conflicts are solved.

Below we investigate two directions to address deadlock by adding global control to ERA (Section 4.1) and by heuristic conflict resolution (Section 4.2).

4.1 Enhancing ERA with global control

Inspired by local search, we propose to enhance ERA with global control in order to avoid deadlocks. To this end, we add the following reactive rule to the ERA system. After selecting a position to move to that improves its own local goal (this is done according to the reactive rules of Section 2.3), the agent also checks the effect of this move on the global goal, measured as the total number of violations of the entire state. Indeed, the agent's new position may increase the number of violations for one or more other agents. Only when it does not deteriorate the global goal does the agent effectively execute the considered move.

Experiment 5 Solve the GTA problem for the data set of Spring2001 (O), an over-constrained instance, and that of Fall2001 (O), a solvable instance, using the original and then the modified ERA algorithm. We choose `FrBLR` as the default behavior and observe the number of agents in `zero position`.

On the over-constrained data set, the new rule we added for global control was able to reduce the deadlock but not eliminate it completely. Indeed, we observed that the modified ERA was able to reach better solutions than the original ERA; however, the solution was still not as good as the one reached by local search. For the solvable data set, the modified ERA was quickly trapped in a local optimum, similar to local search. Thus, our attempt to add global control to ERA fails. On one hand, we are not able to reach as good solution as local search, and on the other hand, we inherit the shortcoming of local search. It is important to investigate hybrid strategies that combine the benefits of various search techniques without inheriting their shortcomings.

4.2 Conflict resolution

Here we discuss two automatic conflict resolution procedures for ERA. The first one is based on introducing dummy values in the CSP, and the second on using the violation values obtained by ERA as a priority criterion.

1. Add *dummy* resources one by one, and attempt to solve the problem again. (The agents given the *dummy* resources remain unassigned in reality.) We implemented this strategy for the data sets Spring2001b (O) and Fall2002 (O) (see Rows (b) and (f) in Tab. 2), yielding the data sets shown in Rows (a) and (e) in Tab. 2. We noticed that ERA was able to solve the problem while the two other search strategies failed. Indeed, there are not enough GTAs to solve these data sets. Boosting them by adding 'dummy' values, one at a time, allows ERA to solve these problems. After a complete solution is obtained, we remove the dummy values. We noticed that this technique allows us to generate partial solutions significantly better than those obtained by LS and BT, which failed to solve even the boosted instances.
2. Given a deadlock (described here as a set of conflicting agents and the unique index of their corresponding positions), two configurations are possible: either all agents have the same value for the position or they have different, non-zero values. The situation is illustrated in Fig. 9. A circle represents the index of the conflicting positions, a square represents an agent, and the value within the square is the violation value for the position. The agents located on a circle cause a deadlock. In the first case, we are in a situation in which the constraints and preferences in the problem yielded positions that have *exactly* the same values for the variables in a deadlock. In other words, ERA does not have enough information to discriminate among the variables involved in the deadlock. Thus, an arbitrary, greedy assignment of the position to any of the agents in the deadlock is the only mechanical way to solve the deadlock. We propose to solve Case 2 as follows. We sort the conflicting

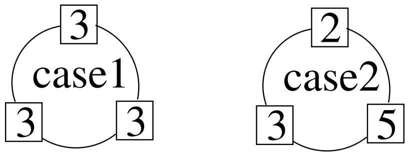

Fig. 9. *Two cases of a deadlock.*

agents in an increasing order according to their violation value. We examine the available position for assignment to the first agent in the priority queue, breaking ties randomly, and check whether this does not yield any inconsistency in the entire problem. If any inconsistency is encountered, we remove the agent from the queue. If not, then we assign the position to the agent, and we update the violation values for the remaining agents. The process is repeated until all agents in the deadlock have been examined. We did not test this technique because it seemed more complex than the previous one.

5 Future research directions

We carried out our experiments on the toy problems (i.e., the 100-queen problem and the Zebra puzzle), and on real-world instances of the GTA problem. While our observations seem generalizable beyond these problems, we still need to validate them on randomly generated and other real-world CSPs, which we are currently doing. In particular, working on the GTA problem, which is a resource allocation problem, allowed us to explain the deadlock as resource contention. It is not clear what a deadlock would mean in CSPs that model other practical applications or even in (over-constrained) random CSPs. We are currently carrying out tests in this direction. The general methodology of [11, 12] for characterizing the behavior of randomly-generated problems will be useful to this end.

In this paper, we showed that ERA is particularly effective at handling tight, solvable problems that resist other search techniques. However its shortcomings with over-constrained problems (i.e., its instability and the degradation of the approximate solutions it finds) significantly undermine its usefulness in practice. We plan to address this problem from the following perspectives:

1. Enhance ERA with a mechanism to handle optimization problems, where two or more consistent assignments can be further differentiated by a preference criterion.
2. Develop and test strategies to address deadlocks, perhaps through endowing agents with non-uniform reactive rules. Note that the ability of ERA to isolate the deadlock is a significant advantage.
3. Experiment with search hybridization techniques with LS, which can reach and maintain a good quality approximate solution within the first few iterations.
4. Conduct similar experiments on other search techniques such as randomized systematic search [10], the squeaky wheel method [14], and market-based techniques [18], in a setting similar to the 'algorithm portfolios' of [9].

Acknowledgments. This research is supported by NSF grant #EPS-0091900. We are grateful to Deb Derrick for editorial help.

References

1. Fahiem Bacchus and Peter van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proc. of AAAI-98*, pages 310–319, Madison, Wisconsin, 1998.

2. Roman Barták. On-Line Guide to Constraint Programming. kti.ms.mff.cuni.cz/~bartak/constraints, 1998.
3. Berthe Y. Choueiry and Boi Faltings. A Decomposition Heuristic for Resource Allocation. In *Proc. of the 11 th ECAI*, pages 585–589, Amsterdam, The Netherlands, 1994.
4. Andrew Davenport, Edward Tsang, Chang J. Wang, and Kangmin Zhu. Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proc. of AAAI-94*, pages 325–330, Seattle, WA, 1994.
5. Rina Dechter. *Constraint Programming*. Morgan Kaufmann, 2003.
6. Mark Fox. *Constraint Directed Search: A Case Study of Job-Shop Scheduling*. Morgan and Kaufmann, Los Altos, CA, 1987.
7. Eugene C. Freuder and Richard J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
8. Robert Glaubius and Berthe Y. Choueiry. Constraint Modeling and Reformulation in the Context of Academic Task Assignment. In *Working Notes of the Workshop on Modelling and Solving Problems with Constraints, ECAI 2002*, Lyon, France, 2002.
9. Carla P. Gomes and Bart Selman. Algorithm Portfolios. *Artificial Intelligence*, 126 (1-2):43—62, 2001.
10. Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proc. of AAAI-98*, pages 431–437, Madison, Wisconsin, 1998.
11. Holger H. Hoos. *Stochastic Local Search—Methods, Models, Applications.* PhD thesis, Technische Universität Darmstadt, Germany, 1998.
12. Holger H. Hoos and Thomas Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, 112 (1-2):213—232, 1999.
13. Michael Jampel, Eugene C. Freuder, and Michael J. Maher, editors. *Over-Constrained Systems*, volume 1106 of *Lecture Notes in Computer Science*. Springer, 1996.
14. David E. Joslin and David P. Clements. Squeaky Wheel Optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.
15. Jiming Liu, Han Jing, and Y.Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136:101–144, 2002.
16. 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.
17. Patrick Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9 (3):268–299, 1993.
18. Tuomas Sandholm. Algorithms for Combinatorial Auctions and Exchanges. Tutorial MA3, AAAI-02, Alberta, Edmonton, Canada, July 2002.
19. Hui Zou. Iterative Improvement Techniques for Solving Over-Constrained Problems. Master's thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, 2003. Forthcoming.