Multi-agent Based Search versus Local Search and Backtrack Search for Solving Tight CSPs: A Practical Case Study

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 for solving a Constraint Satisfaction Problem (CSP) [Liu *et al.*, 2002]. We compare this method's performance with that of a local search (LS) and a systematic (BT) search, in the context of a real-world application that is over-constrained—the assignment of Graduate Teaching Assistants (GTA) to academic tasks. We report our observations and summarize our analysis of the main features and limitations of this multi-agent search. We show that for solvable, tight CSPs, multi-agent search 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.

1 Introduction

A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ the set of their respective domains, and $\mathcal{C}$ is a set of constraints that restricts the acceptable 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. CSPs are used to model a wide range of decision problems, thus they are important in practical settings.

Search algorithms for solving CSPs are usually classified into two main categories: local search and systematic algorithms. Recently, Liu et al. [2002] proposed a competitive multi-agent based technique. In this paper, we analyze the behavior of a multi-agent based approach, ERA of Liu et al. [2002], in solving an over-constrained practical problem: the assignment of graduate teaching assistants (GTA problem) to academic tasks[1]. We compare the behavior of the multi-agent approach to two other approaches we have implemented to solve this problem: a systematic, backtrack search and a hill-climbing, local search. We stress that our investigations are motivated by, and focus on, the GTA problem of which we have collected a few, but real, data samples. Consequently, our experiments are exploratory in nature. More thorough experiments, in a methodology similar to that of [Hoos, 1998], still need to be carried out to validate our conclusions.

The results of our study can be summarized as follows. The multi-agent approach exhibits the best ability to recover from local optima due to its goal-directed behavior and communication capabilities. 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. We were able to trace this shortcoming to the same feature that constitutes the strength of this approach, i.e. the inter-agent communication mechanism, which results in a deadlock state in over-constrained situations.

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 and summarizes our observations. Section 4 discusses the relative performance of the methods tested. Finally, Section 5 provides direction for future research.

2 Background

Here we summarize the principle of the multi-agent based search and describe the main features of our particular application. We then explain how the application relates to the principle.

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. [2002] proposed a search method for solving CSPs based on a multi-agent approach in which 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. Thus, the positions of the agents in such an environment constitute the solution to a CSP.

Liu et al. [2002] presented an algorithm, called ERA (i.e., Environment, Reactive rules, and Agents), that is an alternative, multi-agent formulation for solving a general CSP. Although ERA can be viewed as an extension to local search, it

*115 Ferguson Hall, Computer Science and Engineering, UNL, Lincoln NE, 68588-0115, fax +1(402)472-7767.

[1]Note that we do not address the more general Employee Timetabling Problem (ETP) defined in [Schaerf and Meisels, 2000].

differs from local search in some subtle ways. Moving from one state to another in local search typically involves changing the assignment of one (or two) variables, thus the name local search. In multi-agent search, any number of variables can change positions at each move; each agent chooses its most convenient position (e.g., value). The evaluation function that assesses the quality of a given state in local search is a global account of the quality of the state (typically the total number of broken constraints). In ERA, the value of the state is a combination of the value of the individual agents (typically the number of broken constraints of an agent). ERA appears to decentralize the global control of the selection of the next state to the individual agents.

2.2 Graduate Teaching Assistants (GTA) problem

As a real-world CSP, the GTA problem is defined as follows. In a semester, given a set of graduate teaching assistants, a set of courses, and a set of constraints that specify allowable assignments, find a consistent and satisfactory assignment of GTAs to courses [Glaubius and Choueiry, 2002]. In practice, this problem is over-constrained. Typically there are not enough GTAs to cover all tasks, and some courses may have no GTA assigned to them. The goal of the GTA problem is to ensure GTA support to as many courses as possible. We measure the quality of a solution primarily by the number of courses assigned to a GTA. A secondary criterion is to maximize the arithmetical or geometric average of the assignments since each GTA expresses a preference value (between 0 and 5) for each course.

In the GTA problem, the courses are modeled as variables and the GTAs are the values. There are a number of unary, binary and non-binary constraints that model the rules governing the assignments. In particular, each course has a load. The *total load* of a semester is the maximum of the cumulative load of the individual courses (in our setting, the semester has two parts that do not always have equal loads). Further, each GTA has a capacity factor, which is constant throughout the semester and indicates the maximum course weight he or she can be assigned at any point in time during the semester. The sum of the capacities of all GTAs' represents the *resource capacity*. A detailed description of the problem, the constraints, and the solution derived by using a backtrack search can be found in [Glaubius and Choueiry, 2002].

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). The environment records the number of constraint violations of the current state for each value in the domains of all variables. Each variable is an agent, and the position of the agent corresponds to the value assigned to this variable. The agent moves according to its reactive rules. Three assumptions are made: all agents have the same reactive rules; an agent can only move to positions in its own domain; and agents move in sequence and not at the same time.

Environment

An environment E is 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. The entry $e(i,j).violation$ in the environment E refers to a position at row i (representing Agent i) and column j (representing the value of index j in the domain of Agent i). It stores a list of two values for Agent i in position j, namely *domain value* and *violation value*. Domain value, $e(i,j).value$, is the current value assigned to an agent. Violation value, $e(i,j).violation$, is the number of constraints broken by the current assignment of an agent. Fig. 1 illustrates the environment E of in the GTA problem. Each course is an agent, and each cell records two values of the agent: the domain value (i.e., a GTA) and the violation value. This violation value

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)	

Figure 1: *The data structure of environment E.*

of an agent is also called its `position value`. A `zero position` is a position for the agent that does not break any of the constraints that apply to it. That means the current assignment of the agent is consistent with the other agents' assignments. Obviously, if agents are all in zero positions, 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`.

Reactive rules

Liu et al. [2002] define three reactive rules, R, that govern the interaction between an agent and the environment:

1. *Least-move*: The agent chooses the position with the minimal 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 value than the current position value, 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 avoids the possibility of the agent getting stuck in *a local optimum*.

Agent

A variable is represented by an agent. At each state, the agents choose a position to move to following the reactive rules. The agents will keep moving until all have reached `zero position` or a certain time period has elapsed.

ERA algorithm

The ERA algorithm has the following five main functions:

1. `Initialization` builds the environment E, generates a random position for each agent, and moves the agent to this position.
2. `Evaluation` calculates the violation value of each possible position for each agent.
3. `Agent-Move` checks whether an agent is in `zero position`. If it is not, it tries to find a new position

for the agent and calls `Evaluation` to update the current state. Otherwise, it does nothing.

4. `Get-Position` uses the applicable reactive rule to find a new position for an agent.

`Initialization`
Input: a problem
Output: a random state
1: Build environment E and initialize its entries
2: **for** each agent **do**
3: move to a random position
4: **end for**

Algorithm 1: Initialization

`Evaluation`
Input: a state
Output: update position values
1: **for** each agent i **do**
2: **for** each position in the current domain of agent i **do**
3: Calculate the position value based on the current assignments of other agents'
4: Store this value
5: **end for**
6: **end for**

Algorithm 2: evaluation

`Agent-Move`
Input: a state
Output: a new state
1: **for** each agent i **do**
2: **if** ($e(i,j).violation$=0) **then**
3: do nothing
4: **else**
5: $j \leftarrow$ *Get-Position*
6: call *Evaluation*(`state`)
7: **end if**
8: **end for**

Algorithm 3: Agent-Move

5. ERA loops over the agents and keeps moving them until they are in `zero position` or a specified number of iterations `MAX_MOVE` is reached. When all agents reach a `zero position`, the problem is solved and the solution is returned. Otherwise, the best approximate solution encountered to date is returned.

Liu et al. [2002] established that the time complexity of the ERA algorithm $\mathcal{O}(n^2 \times D_{\max})$ and its space complexity is $\mathcal{O}(n \times D_{\max})$. Further, they 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 experiment, 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. [2002].

`Get-Position`
Input: an agent
Output: a new position
1: calculate a probability p
2: **if** ($p = p_1$) **then**
3: $position \leftarrow$ *least-move*
4: **else if** ($p = p_2$) **then**
5: $position \leftarrow$ *better-move*
6: **else**
7: $position \leftarrow$ *random-move*
8: **end if**
9: return $position$

Algorithm 4: Get-Position

`ERA`
Input: a problem
Output: a solution
1: $step \leftarrow 0$
2: *Initialization*
3: *Evaluation*
4: **while** not all agents are in `zero position` or $step \leq$ `MAX_MOVE` **do**
5: *Move-Agent*
6: $step \leftarrow step + 1$
7: compare and store solution
8: **end while**
9: output solution

Algorithm 5: ERA

Possible behaviors and some observations

Liu et al. [2002] 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* 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*. In case 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. [2002] 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. And (4) `FrBLR` outperforms `rBLR`, which in turn outperforms `LR` in terms of runtime.

3 Empirical evaluation of ERA

We tested our implementation on known problem instances. First, we solved the 100-queen problem with different agent

behaviors [2]. Then, using FrBLR as the default behavior of agents, we solved 8 instances of the GTA problem, including solvable and over-constrained cases. We conducted four main experiments: we tested the behavior of ERA (Section 3.1), compared it to that 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 and are numbered accordingly.

3.1 Testing the behavior of ERA

In the following two experiments we recorded the number of agents reaching zero position at every iteration.

Experiment 1. Solve 100-queen problem with LR, BLR and FrBLR.

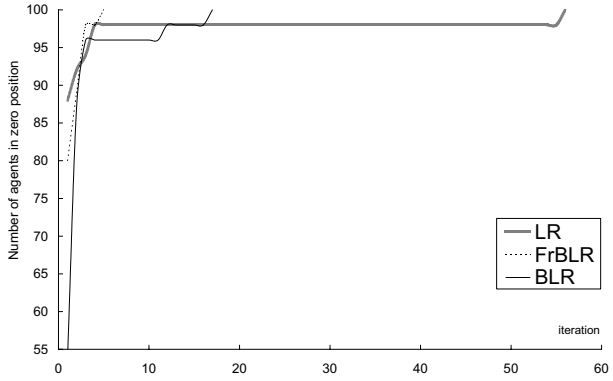

Figure 2: *Agents in zero position in the 100-queen problem.*

Observation 1.1. FrBLR exhibits the best performance in terms of the number of iterations necessary to reach a solution, see Fig. 2. Indeed:

- After the first iteration, there are 54 agents in zero position with LR, 88 with BLR, and 80 with FrBLR. For BLR and FrBLR, over 80% of the agents are already in zero position.
- FrBLR reaches a full, consistent solution after only 5 iterations. BLR requires 17 iterations, and LR 56.

This observation is in agreement with the results of Liu et al. [2002]. We report the following additional observations.

Experiment 2. Solve the GTA problem for the data-set Fall2001b using LR, BLR and FrBLR.

Observation 2.1. 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.

[2]The n-queen problem is not particularly well-suited for testing the performance of search. However, we use it only to be on a common level with Liu et al. [2002]. Indeed, they conducted their tests on the n-queen and the coloring problems, and drew their conclusions from the n-queen problem.

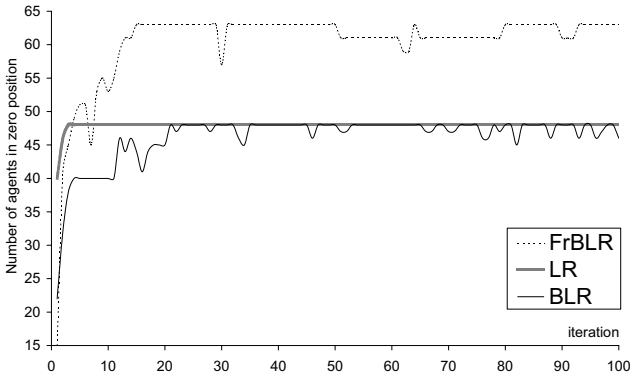

Figure 3: *Agents in zero position for Fall2001b.*

Observation 2.2. Fig. 3 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.

Observation 2.3. 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.

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 [Glaubius and Choueiry, 2002]. The second strategy is a hill-climbing, local search (LS) to be documented in [Zou, 2003]. LS is a combination of constraint propagation to handle non-binary constraints and the min-conflict random walk algorithm as presented in [Barták, 1998].

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, so that the search behaves as an anytime algorithm.

Experiment 3. Solve the GTA problem for the real-world data of Fall2001b, Fall2002 and Spring2003 using ERA, BT search [Glaubius and Choueiry, 2002], and a hill-climbing, local search technique [Zou, 2003]. 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. 1 and Figs 4 and 5.

We adopted the following working conditions:

- As soon as BT search becomes unable to find a better solution after 20 seconds, it is interrupted.
- The maximum iteration number for LS and ERA is 200.

- We increased the number of dummy GTAs, one at time, until one of the search techniques could find a solution. The solvable instance thus obtained may have more GTAs that it is actually needed.
- The ratio of `total capacity` and `total load` (shown in column 7 in Tab. 1) is an indicator of the tightness of the problem. When it 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. 1). 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. 1). 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. 1). This value is useful to analyze why certain resources are not used by the search mechanism, 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. 1). 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 [Bacchus and van Beek, 1998] (col. 12, 17, and 22 in Tab. 1).

Observation 3.1. Only ERA was able to find a full solution to all solvable problems (column 18 of Tab. 1). Both BT and LS failed 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.

Observation 3.2. When the ratio of total capacity to total load is greater than 1 (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. 4. Indeed, we make the conjecture that ERA is not a reliable technique for solving over-constrained problems.

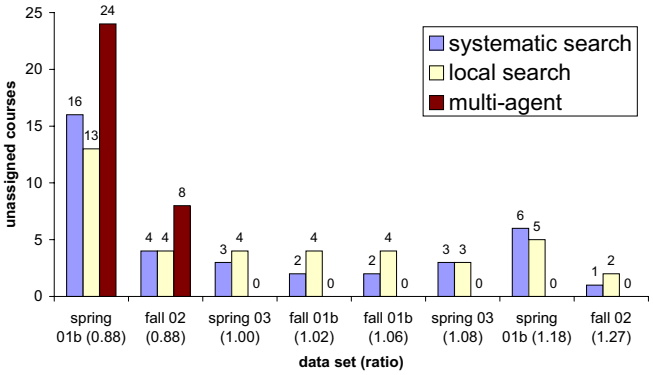

Figure 4: *Unassigned courses*

Data Set								Systematic Search (BT)				Local Search (LS)				Multi-Agent Search (ERA)			
	Original/Boosted	Solvable?	#GTAs	#Courses	Total capacity	Total load	Ratio = $\frac{\text{TotalCapacity}}{\text{TotalLoad}}$	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource	Unassigned Courses	Solution Quality	Unused GTAs	Available Resource
											CC ($\times 10^8$)				CC($\times 10^8$)				CC($\times 10^8$)
Spring2001b	B	√	35	69	35	29.6	1.18	6	4.05	2	6.5	5	3.69	0	6.4	0	3.20	0	5.3
	O	×	26	69	26	29.6	0.88	16	3.79	0	2.5	13	3.54	0	0.9	24	2.55	8	8.3
Fall2001b	B	√	35	65	31	29.3	1.06	2	3.12	0	2.5	4	3.01	0	3.8	0	3.18	1	1.9
	O	√	34	65	30	29.3	1.02	2	3.12	0	1.5	4	3.04	1	3.7	0	3.27	0	0.8
Fall2002	B	√	33	31	16.5	13	1.27	1	3.93	0	3.5	2	3.40	0	5.0	0	3.62	2	3.0
	O	×	28	31	11.5	13	0.88	4	3.58	0	1.8	4	3.61	0	2.0	8	3.22	1	2.0
Spring2003	B	√	36	54	29.5	27.4	1.08	3	4.49	2	4.2	3	3.62	0	3.9	0	3.03	1	2.8
	O	√	34	54	27.5	27.4	1.00	3	4.45	0	2.2	4	3.63	0	3.3	0	3.26	0	0.8
Column reference		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**

Table 1: *Comparison between BT, LS, and ERA.* strategies O/B indicates whether the instance is original or boosted. `CC` is the number of constraint checks.

Observation 3.3. On average (see Fig. 5), 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) is a costly operation.

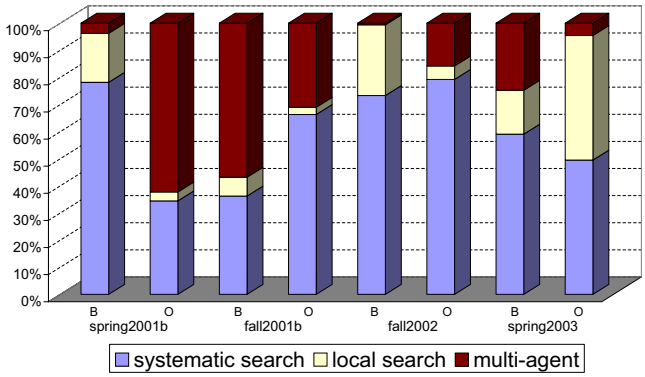

Figure 5: *Constraint checks*

Observation 3.4. ERA leaves more GTAs unassigned than BT or LS (col. 10, 15, and 20 in Tab. 1), 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. 6). The unsolvable ones include Spring2001b (O) and Fall2002 (O) (Fig. 7).

Observation 3.5. Fig. 6 and Fig. 7 show that the performance of ERA is more stable when solving solvable problems than when solving unsolvable problems.

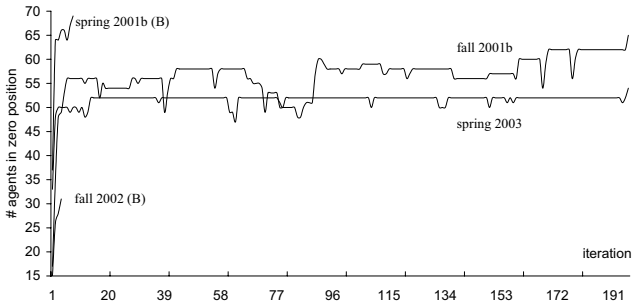

Figure 6: *ERA performance on solvable problems*

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. 8. In this figure, we used the index of the agent's position to indicate its assigned value.

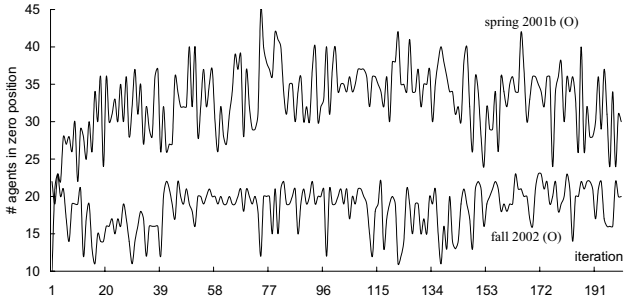

Figure 7: *ERA performance on unsolvable problems*

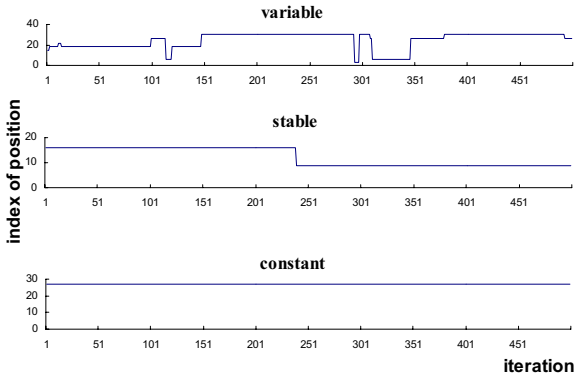

Figure 8: *Types of agent movement*

- Variable: the agent changes its position relatively frequently and fails to find its `zero position`.
- Stable: the agent rarely changes its position.
- Constant: the agent finds a `zero position` at the beginning of the search, and never changes it.

Experiment 4. 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.

Observation 4.1. In solvable instances, most agents are stable, and a few are constant. None of the agents is variable.

Observation 4.2. In unsolvable instances, most agents are variable, and a few are stable. None of the agents is constant.

3.4 The deadlock phenomenon

On our two unsolvable instances of Tab. 1 (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.

Experiment 5. 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.

Observation 5.1. 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 the position values never becoming `zero position`s for any of the agents. 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. 9. Each circle corresponds to a given GTA, note that there is exactly one circle per GTA. Each square represents an agent. 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.

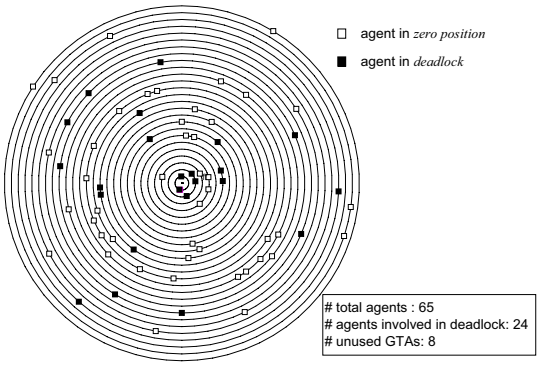

Figure 9: *Deadlock state*

Definition 1. *Deadlock state:*When a subset of agents competes for positions that are mutually exclusive but the only positions acceptable for them, a deadlock occurs that prevents all agents from being assigned this 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`.

Observation 5.2. ERA is not able to avoid deadlocks and yields a degradation of the solution. 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 position`s 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.

4 Discussion

We further discuss the performance of ERA.

4.1 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. [1992], 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.

4.2 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 towards 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.

4.3 Global vs. local repair

Our implementation of local search, a hill-climbing strategy with a combination of a min-conflict heuristic and propagation of capacity constraints, does not undo consistent assignments. No variable is repaired more than once. Once a variable has a consistent assignment, this assignment is never undone. Repairs are done locally, to one variable at a time, which monotonically increases the number of instantiated variables. In practice, such an approach seems to get quickly stuck in local optima, which is unlikely to be overcome even with random restarts [Hoos and Stützle, 1999].

In ERA, however, there is no such restriction. An agent can undo its assignment as needed, even if it is a consistent one. This feature of ERA seems to be the major reason why it is able to solve successfully large, tight problems that resisted the other techniques we tested (i.e., solvable instances of Tab. 1 were only solved by ERA).

In ERA, each agent has its own local goal—to move to a minimal violation-value position. During search, each agent focuses on achieving its local goal. The global goal of minimizing conflicts of a state is implicitly controlled by the environment *E*, through which the agents 'communicate.' This communication medium is effective when the problem is solvable, because agents are always able to find a 'better position' to move to. It also increases the 'freedom' of an agent to explore its search space, which allows search to avoid local

optima. This decentralized control is contrasted with the centralized control used in local search where the neighboring states are evaluated globally by a centralized function.

4.4 Soft vs. hard assignment

We identify two main strategies to handle deadlocks:

Definition 2. *Hard assignment:* none of the variables is assigned the value.

Definition 3. *Soft assignment:* using some conflict resolution strategy, some of the variables are assigned the value in a manner that respects its capacity constraint.

The former strategy is implemented in ERA and results in fewer variables being instantiated. The latter is adopted in BT and LS and results in more variables being instantiated. We believe that the former 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. Indeed, conflict identification is a difficult task (perhaps NP-hard) and ERA may well be an effective strategy to approach it.

4.5 Stable vs. unstable evolution

As highlighted in Fig. 6 and 7, the evolution of ERA across iterations, although not necessarily monotonic, is stable on solvable problems and gradually moves towards a full solution. On unsolvable problems, its evolutions is unpredictable and appears to oscillate significantly. This contradicts the conclusion of Liu et al. [Liu *et al.*, 2002], which claims that ERA always evolves toward better states. This clearly does not hold true for over-constrained problems.

4.6 Solvable vs. unsolvable problems

Among the three strategies we tested (we are testing others), only ERA was able to solve our hard, solvable instances. This ability can be traced to its decentralized control, which allows it to move out of local optima. However, this feature is also the cause of the deadlock phenomenon encountered in over-constrained cases. Indeed, ERA's performance is unstable and leaves many solvable tasks unsolved. This yields solutions less competitive than those obtained by BT and LS. We therefore need to develop a more sophisticated inter-agent communication mechanism to overcome this obstacle, which is overlooked in the original ERA algorithm [Liu *et al.*, 2002].

5 Future research directions

Our experiments were carried out on the 100-queen problem and real-world instances of the GTA problem. While our observations seem generalizable beyond these two problems, we still need to characterize the behavior of multi-agent search on random and other real-world CSPs.

We showed that ERA is particularly effective at handling tight, solvable problems that resisted other search techniques. However its shortcomings on 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. Development of conflict resolution strategies to overcome deadlocks. Note that the ability of ERA to isolate the deadlock is a significant advantage in this task.
2. Experiment with search hybridization techniques with LS, which can reach and maintain a good quality approximate solution within the first few iterations.
3. Further, we plan to expand our study to include techniques such as randomized systematic search [Gomes *et al.*, 1998], the squeaky wheel method [Joslin and Clements, 1999], and market-based techniques [Sandholm, 2002], in a setting similar to the 'algorithm portfolios' of [Gomes and Selman, 2001].
4. Finally, we plan on conducting a more thorough empirical evaluation of the behavior of the various algorithms on randomly generated problems following the methodology of [Hoos, 1998; Hoos and Stützle, 1999].

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

References

[Bacchus and van Beek, 1998] F. Bacchus and P. van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proc. of AAAI-98*, pages 310–319, 1998.

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

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

[Gomes and Selman, 2001] C.P. Gomes and B. Selman. Algorithm Portfolios. *Artificial Intelligence*, 126 (1-2):43—62, 2001.

[Gomes *et al.*, 1998] C.P. Gomes, B. Selman, and H. Kautz. Boosting Combinatorial Search Through Randomization. In *Proc. of AAAI-98*, pages 431–437, 1998.

[Hoos and Stützle, 1999] H.H. Hoos and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, 112:213—232, 1999.

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

[Joslin and Clements, 1999] D.E. Joslin and D.P. Clements. Squeaky Wheel Optimization. *JAIR*, 10:353–373, 1999.

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

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

[Sandholm, 2002] T. Sandholm. Algorithms for Combinatorial Auctions and Exchanges. Tutorial MA3, AAAI-02, July 2002.

[Schaerf and Meisels, 2000] A. Schaerf and A. Meisels. Solving Employee Timetabling Problems by Generalized Local Search. In *AI*IA 99: Italian AI*, LNCS Vol. 1792, pages 380–389, 2000.

[Zou, 2003] H. Zou. Iterative Improvement Techniques for Solving Over-Constrained Problems. Master's thesis, CSE-UNL, Lincoln, NE, August 2003. Forthcoming.