# Swarming the Kingdom: A New Multiagent Systems Approach to N-Queens

**Alex Kutsenok**[1], **Victor Kutsenok**[2]

Department of Computer Science and Engineering[1], Michigan State University, East Lansing, MI 48825
Email: kutsenok@verizon.net

Department of Mathematics[2], University of St. Francis, Fort Wayne, IN 46808
Email: vkutsenok@sf.edu

Keywords: swarm intelligence, multiagent systems, and constraint satisfaction

**Abstract**

This paper introduces Swarm Queens, a Swarm Intelligence approach to the N-Queens problem. ERA is the current best Multiagent Systems solution to this classic Constraint Satisfaction Problem. Our algorithm improves upon ERA in terms of both time and space complexity. In addition, Swarm Queens has a simpler design and delivers much more consistent time performance from run to run. We discuss the similarities and differences between these two algorithms, showing how the design decisions we made led to better performance. Our experiment demonstrates that consistent global results can be achieved with non-deterministic local behavior in Multiagent Systems. Furthermore, Swarm Queens exhibits features that may be useful in designing a better general MAS algorithm for solving Constraint Satisfaction Problems.

## 1 Introduction

Swarm Intelligence (SI) is a biologically inspired Multiagent Systems (MAS) technique that involves simple agents working on small parts of a large problem. Work in SI began when scientists noticed how intellectually primitive organisms like ants and bees work together in colonies to accomplish very complex tasks. Swarm Intelligence algorithms attempt to solve problems in the world of computer science by creating teams of simple agents that are guided by nature-influenced rules. Swarm agents are only given a local perspective of their environment and are allowed to work for some time. Ultimately, a global solution is reached through the interaction of these agents (Bonabeau, Dorigo, & Theraulaz 1999). Many other MAS approaches involve complex agents, direct communication, simultaneous actions, and even centralized interference with agents as they work. These features increase the complexity of a MAS algorithm and are not always necessary to obtain the best performance. On the other hand, SI attempts to solve problems by using simple agent designs that are consistent with the behavior of primitive biological organisms such as ants.

We applied SI to the classic Constraint Satisfaction Problem (CSP), N-Queens, to see if a simpler MAS approach could compete with existing MAS solutions. After comparative testing, we found that our SI approach is better than the leading MAS solution, ERA, in terms of time complexity, space complexity, and consistency. Swarm Queens finds a solution to N-Queens in quadratic time, having just linear space complexity. Furthermore, Swarm Queens has minimal time deviation from run to run for problems of the same size.

In this paper, Section 2 discusses SI and previous solutions to N-Queens, including ERA. In Section 3, we explain the rationale behind our MAS synchronization method and agent design. In Section 4, we describe the Swarm Queens algorithm in detail. Section 5 discusses comparative data we collected, the time complexity of Swarm Queens, and other results. Finally, Section 6 provides a summary and discusses implications for future work.

## 2 Background

Swarm Intelligence algorithms have been applied to various problems since their introduction by Dorigo, Maniezzo, and Colorni (1991). These applications include the Traveling Salesman Problem, Quadratic Assignment Problem, tweaking neural networks, and scheduling. Two of the most popular SI approaches are the Ant System (Maniezzo, Gambardella, & De Luigi 2004) and Particle Swarm Optimization (Kennedy, Eberhart, & Shi 2001). Other SI approaches exist, though much of SI work involves applying variants of these two algorithms to different problems.

N-Queens is a Constraint Satisfaction Problem that is solved by finding values for each variable such that every constraint is satisfied. Specifically, a solution is found by placing N queens on an N-by-N board such that they do not threaten each other. Although a fast algorithm has been found that solves N-Queens in linear time (Sosic & Gu 1994), this problem is still commonly used as a benchmark for CSP algorithms (Liu, Han, & Tang 2002).

Han, Liu, and Qingsheng (1999) designed the first MAS solution to N-Queens, introducing the idea of viewing queens as agents. These agents are bound to a single row on the N-by-N board and can move only along that row. Their approach is quite complex, involving heterogeneous agents with different behaviors, energy levels, agent death, and evolution of agent behaviors. This algorithm was improved upon by another MAS approach called ERA (Environment, Reactive rules, and Agents). This approach has homogeneous agents, no agent death or energy levels, and no evolutionary component. Besides being much simpler, the ERA algorithm has better performance than its predecessor (Liu, Han, & Tang 2002).

ERA works by having queen-agents act simultaneously to position themselves in their rows. Thus, during a single ERA round, each of the N queens moves to a square in its row to reduce the number of threats it receives from the other queens. The number of threats from all queens at each square on the N-by-N board is stored in a matrix that is updated after each move. There are three different behaviors from which a queen-agent picks randomly when making an action: least-move, better-move, and random-move (Liu, Han, & Tang 2002).

ERA has $O(n^2)$ space complexity to store the N-by-N matrix of threats. A single round takes $O(n^2)$ time and there can be any number of such rounds as N grows larger. Memory requirements limited tests to problems of up to size N= 7000 (Liu, Han, & Tang 2002). ERA is an inconsistent algorithm, taking very different amounts of time to run for problems of the same size when initialized with different random seeds (Basharu, Ahriz, & Arana 2003).

The ERA framework is general, in that its usefulness is not restricted to N-Queens. This MAS technique has been successfully applied to other CSPs like graph coloring and the Propositional Satisfiability Problem with minimal changes to the basic algorithm (Liu, Han, & Tang 2002; Liu, Jin, & Han 2002).

## 3 Design Decisions and Motivations

### 3.1 Agent Synchronization: Sequential Movement

Recent MAS literature argues for simultaneous actions by agents, where agents work in parallel without a turn-based ordering of when each agent can act (Weyns & Holvoet 2003). The rationale behind this argument is that it is always best to have as little central control as possible. However, we suspect that this reasoning does not apply to all domains where Multiagent Systems are used. In fact, we believe that many problems, such as N-Queens, can be more easily solved by a MAS if its agents move sequentially instead of simultaneously.

For example, every ERA queen-agent looks at the current state of the board and picks a place to move to without knowing how the other agents will move. This leads to unnecessary conflict when two or more queens move to squares in the same column because they each saw that column as unthreatened. These agents ignored better moves elsewhere because they did not know how the others would act. Thus, agents working in parallel are at a disadvantage because they have to make decisions based on incomplete information.

Another problem with the parallel actions of ERA occurs when all but a few queen-agents have found squares where they are not threatened. In this situation, every queen (including the ones that have already found an unthreatened square) is still forced to make a move, wasting computational time. Finally, MAS algorithms with sequential moves are simpler and easier to understand because one can trace the agent actions one at a time as the problem is being solved.

Consequently, we chose to implement a simple sequential ordering for our approach. Each queen-agent in Swarm Queens is assigned to a row on the N-by-N board, as in ERA. The queen in row 0 goes first, followed by the queen in row 1, and so on. When all N Queens have made a move, the queen in row 0 moves again and the cycle continues until a solution is found.

### 3.2 Agent Design: Random-Weight Move

An SI agent should be as simple as possible, so we decided to have the same single movement behavior for all agents. For this behavior, we selected the mechanism used by Ant System agents in the classic Swarm Intelligence approach to the TSP (Dorigo, Maniezzo, & Colorni 1991). We call this behavior a Random-Weight move because it involves making a random decision from among several choices with different probabilities of being selected.

In Swarm Queens, a Random-Weight move consists of a queen choosing a square to move to in its row, where all N squares have a chance of being selected. Squares with fewer threats from other queens have a higher probability of being chosen than those with more threats. This behavior encourages agents to frequently move to least threatened squares. However, agents also sometimes make locally sub-optimal exploration moves to more threatened squares, which help the MAS escape from local optima. The specific formula for calculating the probability of moving to each square we borrowed from Dorigo, Maniezzo, and Colorni (1991), and it is described in Section 4.2.

Because our agents have only one behavior, we did not have to tweak the probabilities determining how often each behavior should be selected by an agent, as had to be done by Liu, Han, and Tang (2002) and Liu, Jin, and Han (2002). In addition, we thought it was important to incorporate random local behavior in our approach to test Basharu, Ahriz, and Arana (2003)'s hypothesis that randomness in local behavior results in inconsistent global performance from run to run. Thus, our approach makes use of randomness, just like ERA, to show that it is possible to obtain consistent results with a locally non-deterministic MAS algorithm.

## 4 The Swarm Queens Algorithm

### 4.1 The Main Idea is Similar to ERA

As in ERA, we assign an agent to each of the N queens and place one such queen-agent in every row of the N-by-N board. Initially, the queens are placed at random squares in their rows. The queen-agents have a local perspective, in that they can only look at the squares in their own row when deciding about where to move. Agents can communicate indirectly with others by notifying the squares that they threaten. Thus, each square holds the number of threats from all the queens that threaten it. When it is an agent's turn to move, it looks at all the squares in its row, and consequently, has some idea about the positions of other agents. As the agents make moves, they position themselves such that there are fewer and fewer queens threatening each other until there are no more threats (Liu, Han, & Tang 2002).

### 4.2 The Random-Weight Move

As described in Section 3.1, the queens take turns acting in a pre-defined sequential order. When it is an agent's turn, it can make a Random-Weight move that consists of:

1) reading the threat values of all N squares in its row
2) making a random-weighted decision about where to go
3) going to the selected square

To decide where to go, agents calculate the probability of moving to every square in their row. The square an agent currently occupies is treated exactly like the other N–1 choices, and a queen can decide to go to this square (i.e. stay where it is). Just as Ant System agents are more likely to move to the least distant nodes (Dorigo, Maniezzo, & Colorni 1991), queen-agents are more likely to move to squares with the fewest threats. During its turn, an agent calculates the probability of moving to square j (Pj) for $0 \le j \le N-1$. The formula for calculating Pj is the same one used by ant-agents, except that queen-agents look at threats instead of distances and pheromones (the latter is a second factor Ant System agents have to consider):

$$Pj = \frac{\left(\dfrac{1}{Threats(j)}\right)^C}{\sum_{i=0}^{n-1}\left(\dfrac{1}{Threats(i)}\right)^C}$$

The numerator is the "Safety" value that represents how safe from threats a queen will be at square j. The number of threats at square j is inverted, since a square having more threats should result in a lower Safety value. The denominator is the sum of the Safety values of all the squares in the agent's row. Therefore, Pj is a ratio comparing the Safety of the current square with the combined Safeties of all squares in that row. The sum of all Pj for $0 \le j \le N-1$ is always equal to 1. Note that a queen contributes 1 threat to each square in its row, so the number of threats at any square is always greater than zero.

The constant C is the exploration vs. exploitation constant. Higher values of C correspond to more exploitation, as agents are more likely to choose squares that have fewer threats and are Safer. On the other hand, lowering the value for C results in a higher likelihood of exploration moves. Exploration moves occur when agents move to squares that have more threats than others, and thus, are less Safe. Such moves are necessary to move the MAS away from local optima. Thus, the formula for finding Pj ensures that agents make locally sub-optimal moves to help the algorithm find the global optimal solution. Empirical testing of different C values for a wide range of N showed that C between 12 and 20 yield better results than other values. We set C equal to 16 when obtaining the data provided in this paper.

### 4.3 Efficient Storage of Threat Values

It is not necessary to store an entire N-by-N array of threat numbers to remember the number of threats at each square. A queen threatens squares in 8 directions that correspond to 4 "threat lines": row, column, upper-left diagonal, and upper-right diagonal. It is wasteful for queens making a threat to write on every square along those threat lines, which takes linear time. An improvement can be made by allowing a queen to inform an entire line that it is being threatened, not the individual squares that make up that line. Then, a queen-agent only has to inform the 4 threat lines passing through the square it is in about being threatened.

For example, queen Q1 in square (2, 1) threatens the 2nd column, as shown in Figure 1. It must add 1 to the number of threats along that column. This can be accomplished in constant time by incrementing a single element of an array of columns. Incrementing the number of threats along a single column has the same effect as incrementing the number of threats in all the squares of that column. Q1 also threatens the 1st row and two diagonals; the number of threats along those threat lines can be incremented similarly.

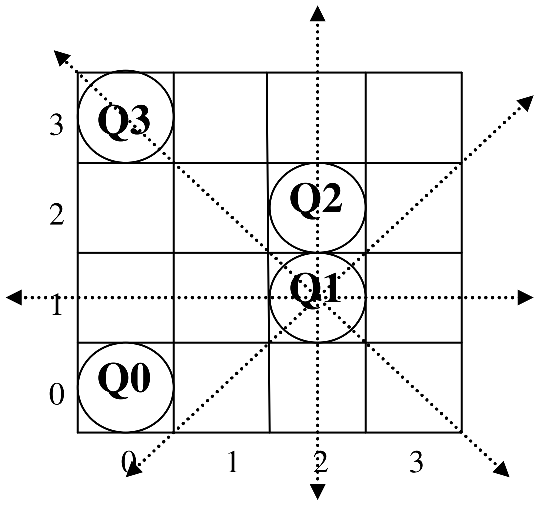

Figure 1: Threat Lines from Queen Q1 at (2,1) for a problem of size N=4. Q1 contributes 1 threat to all the squares through which its threat lines pass.

Thus, we store only 4 arrays corresponding to the 4 types of lines passing through the board. This requires linear storage space, which is an improvement over the O($n^2$) space complexity of ERA (Liu, Han, & Tang 2002). Reading and writing threats to lines instead of squares does not affect how decision-making is done by agents. They still look at the number of threats at each square in their row, regardless of how that number is stored.

## 4.4 Queen Movement

When a queen-agent decides to leave one square to move to another, all the squares it previously threatened have to be informed that they are no longer threatened. Additionally, the squares the queen will threaten from its new square have to be informed that there is an additional threat. In Swarm Queens, only the 4 threat lines passing through the square a queen is leaving have to be informed, which takes constant time. Similarly, the lines it threatens from the new square can be informed in constant time. Therefore, incrementing or decrementing threat values in all the squares a queen threatens (or used to threaten) takes constant time. Consequently, moving a queen takes constant time. This is an improvement over the linear time ERA takes to move a single queen (Liu, Han, & Tang 2002).

## 4.5 The Happy Lazy Principle

ERA is a good approximating algorithm (Basharu, Ahriz, & Arana 2003; Liu, Han, & Tang 2002), which means it converges quickly to a state where all but a few of the queens are not threatened by others. However, these "happy" queens keep making moves, most of the time resulting in their staying in the same square. These unnecessary moves take up valuable processing cycles for the rest of the time the algorithm runs.

We experimented with the idea of having happy queens act less frequently since they are satisfied with their position. Surprisingly, testing showed that a global solution can be found if happy queens are "lazy" and do not move at all. We call this the Happy Lazy Principle, which dictates that happy queens skip their turn. Therefore, only unhappy queens may move. This principle makes sense because it is unnatural for agents that are completely satisfied to want to move somewhere else as long as they are happy. This does not cause the MAS to become stuck at local optima because unhappy agents can still indirectly communicate with happy ones. Specifically, unhappy queens may threaten the squares of happy queens to force them to move.

As a result, we must make a distinction between two types of turns in Swarm Queens: skipped and actual turns. A skipped turn occurs when an agent only checks that the number of threats in its current square is 1 and is done (the only threat comes from itself). An actual turn occurs when a queen makes a Random-Weight move after realizing that it is threatened by one or more other queens. A skipped turn takes constant time, which is insignificant compared to the linear time an actual turn takes. For a problem of size N=1000, Swarm Queens took an average of 14,429 total turns. However, only an average 696 of those turns were actual turns that took up significant processing time. This means that the Happy Lazy Principle decreased the time to find a solution by $\frac{14{,}429}{696} \approx 20$ times for this problem size. Testing for problems of other sizes yielded similar gains in performance (see Section 5 for more data).

Besides making our algorithm faster, the Happy Lazy Principle is also responsible for making Swarm Queens significantly more consistent. The standard deviation of the total number of turns Swarm Queens takes is as high as 60% of the average for some N. This means that the turn number varies wildly. However, the standard deviation of the actual number of turns is only 3.30% of the average for N=1000. This shows that there is little difference in the number of actual turns from run to run. As N grows larger, the number of actual turns becomes even more consistent, with the standard deviation equal to 0.25% of the average for N=70,000. The time our algorithm takes is directly related to the number of actual turns, so the running time is consistent as a result (see Section 5 for data on time consistency).

## 4.6 Finding a Solution with Swarm Queens

Swarm Queens does not take O(n) time to check for a solution like ERA (Liu, Han, & Tang 2002). Instead, a single variable (Num_Skipped) is kept to record how many queens have skipped their turn in a row. When

Num_Skipped = N, it means that all N queens are happy, so a solution has been found.

A high-level view of the Swarm Queens algorithm is given in Figure 2. Initially, it assigns a queen to each of the N rows, giving it a random x-coordinate. Then, the algorithm iterates through all the queens sequentially, allowing happy queens to skip their turn according to the Happy Lazy Principle. If a queen is not happy, it always makes a Random Weight move and resets the Num_Skipped variable to 0. Checking for a solution takes constant time, done by comparing Num_Skipped to N.

```
Random_Init_Queen_X_Coords()
Num_Skipped= 0, current_queen= 0
while(Num_Skipped<N)
{
  if(Threatened(current_queen) = = true)
  {
    Random_Weight_move(current_queen)
    Num_Skipped=0
   }
  else
    Num_Skipped++

   current_queen= (current_queen+1) % N
}
```

Figure 2: The Swarm Queens Algorithm

## 5 Results

### 5.1 Comparative Testing

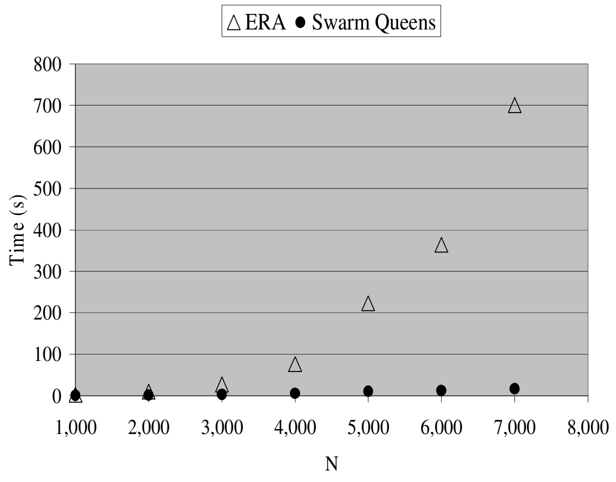

Figure 3: Graph of Time vs. N for ERA and Swarm Queens

To compare the performance of ERA and Swarm Queens, we ran both algorithms on the same machine. All of our tests were conducted on an Intel Pentium III 695Mhz computer with 128 MB of RAM, running Windows XP. Figure 3 shows the average times of both algorithms from 100 runs on each problem of size up to N= 7,000. As N grows, the time ERA takes to run increases much more rapidly than Swarm Queens' time. Thus, our approach is a faster algorithm than ERA.

As in earlier tests (Basharu, Ahriz, & Arana 2003), the Standard Deviation (SD) of the ERA runs in our experiments was quite high. For example, ERA has SD of 709.72s with an average time of 700.72s for N=7000. By comparison, Swarm Queens had a SD of only 1.30s with an average time of 16.9s for the same problem size. Swarm Queens' SD is a much smaller percentage of its average, and therefore, it is a more consistent algorithm from run to run.

### 5.2 Time Complexity of Swarm Queens

When making a Random-Weight move during an actual turn, an agent reads the threat values of all N squares in its row, makes a random-weighted decision about where to go, and moves to the selected square. Reading the threat value at a single square takes constant time, so doing N such reads takes O(n) time. Our implemented algorithm for randomly selecting one square from N choices with different probabilities also takes O(n) time. Finally, moving a single queen takes constant time, as discussed in Section 4.4 Therefore, an actual turn takes 2O(n) + c = O(n) time.

For large N, the time to initialize the queen positions and the time skipped turns take can be ignored. Therefore, Time ≈ (# actual turns)×O(n). The number of actual turns cannot be determined analytically since it is a product of the emergent behavior of the MAS. Therefore, we must look at empirical evidence of Swarm Queens' performance for large N to determine its time complexity.

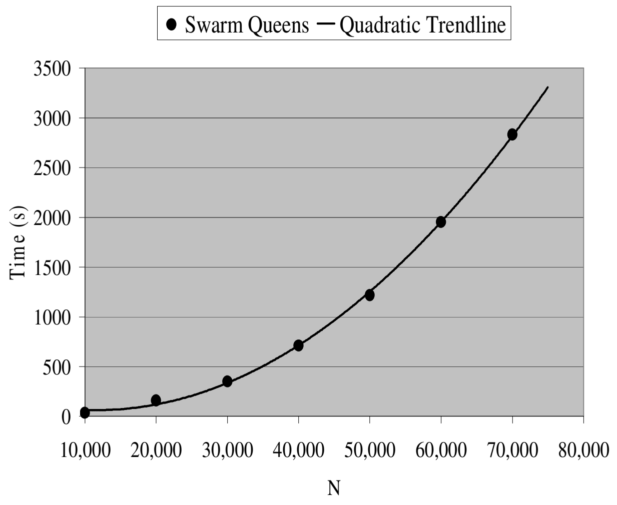

Figure 4: Graph of Time vs. N for Swarm Queens (large N)

Figure 4 is a graph of time vs. N for data points collected by running Swarm Queens on problems from

N=10,000 to 70,000. We did not also collect ERA data for these larger problems since ERA cannot handle them due to its time and memory limits. Each time is an average of 10 runs. Performing regression analysis on the data points yielded the quadratic function $f(x)= 8\times10^{-7}\times x^2 - 0.0188x + 170.73$ that fits the data with $R^2$ value 0.9994. Thus, empirical evidence shows that Swarm Queens works in approximately $O(n^2)$ time. On the other hand, the ERA data from Figure 3 suggests a time complexity of $O(n^3)$ for that algorithm.

## 5.3 Swarm Queens Consistency and Other Results

| N | Time (s) | Time SD (s) | Actual Turns | AT SD | Total Turns |
|---|---|---|---|---|---|
| 10,000 | 35.3 | 0.67 | 6,362 | 43 | 126,002 |
| 20,000 | 158.1 | 0.74 | 13,205 | 57 | 364,324 |
| 30,000 | 349.0 | 1.49 | 18,983 | 70 | 539,196 |
| 40,000 | 711.4 | 2.01 | 27,759 | 70 | 793,557 |
| 50,000 | 1218.4 | 6.38 | 37,816 | 104 | 780,715 |
| 60,000 | 1953.9 | 7.68 | 49,994 | 117 | 965,031 |
| 70,000 | 2831.8 | 13.19 | 62,530 | 157 | 1,126,727 |

Figure 5: Swarm Queens data for large N

Figure 5 lists the times and other data collected from the experiments discussed in Section 5.2. The time and turn values are averages over the 10 runs. Note the low standard deviations of both time and actual number of turns, in comparison to their respective average values. On the other hand, the average standard deviation for the ERA algorithm was equal to 88% of the mean time, which implies the times varied wildly from run to run. Compare that to the average standard deviation of less than 1% of the mean time for the Swarm Queens algorithm. This demonstrates the consistency of our algorithm from run to run for large values of N.

The number of actual turns is much lower than the number of total turns, the algorithm taking that much less time to run because of the Happy Lazy Principle. Swarm Queens was able to find a solution for all problems it was given, from N= 4 to N= 70,000. It returns solutions in less than 8 milliseconds for $N < 100$ and takes less than 1 second to run for problems with $N < 1000$.

## 6 Conclusion

We have demonstrated that our Swarm Intelligence approach to N-Queens produced the fastest, most space efficient, and most consistent MAS solution to this problem to date. In addition, we have introduced the Happy Lazy Principle and utilized Random-Weight moves, a concept borrowed from the Ant System. We believe our approach is simpler than ERA because it uses just one behavior for all agents, and they take turns moving in a fixed sequence without making simultaneous actions. In fact, we only had to tweak one constant while designing our algorithm, which was the exploration vs. exploitation value.

We have shown that sequential agent movement in Multiagent Systems can be more effective than simultaneous movement. Furthermore, Swarm Queens is consistent from run to run without requiring a complex centralized global feedback mechanism for this purpose, as suggested by Basharu, Ahriz, and Arana (2003). Thus, our experiments prove that non-determinism at the local level does not necessarily cause inconsistent global behavior. While making a MAS more centralized can improve and stabilize its performance (Basharu, Ahriz, & Arana 2003), Swarm Queens demonstrates that this can be accomplished with changes at the local level only.

In the future, it may be possible to solve N-Queens with a MAS approach in less than quadratic time. Both moving a queen and checking for success take constant time in Swarm Queens. Therefore, the performance bottleneck is the linear time agents take to decide where to go. This time can be reduced by altering Random-Weight move so that agents look at only a few of the squares in their row to reach a decision. If this change does not cause agents to take significantly more actual turns, the resulting algorithm will solve N-Queens in sub-quadratic time. To reach this goal, future work should focus on designing an effective pruning algorithm for picking what squares an agent should consider during a Random-Weight move.

Ultimately, ERA is a general MAS algorithm that is capable of solving not just N-Queens but also other CSPs. While our work shows that Swarm Queens is better than ERA for the N-Queens problem, more research must be done to determine whether our approach can match ERA in terms of generality in the constraint satisfaction domain. Therefore, we intend to apply the combination of sequential movement, Random-Weight moves, and the Happy Lazy Principle to other CSP problems. These experiments should provide additional information about the usefulness of the ideas presented in this paper.

## References

Basharu, M.; Ahriz, H.; and Arana, I. 2003. Escaping Local Optima in Multi-Agent Oriented Constraint Satisfaction. *Proceedings of the 23rd SGAI Intern. Conf. on Innovative Techniques and Apps. of AI.*

Bonabeau, E.; Dorigo, M.; and Theraulaz, G. 1999. *Swarm Intelligence: From Natural to Artificial Systems.* New York: Oxford University Press.

Dorigo, M.; Maniezzo, V.; and Colorni, A. 1991. Positive feedback as a search strategy, Technical Report, 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Han J.; Liu J.; and Qingsheng C. 1999. From ALIFE agents to a kingdom of n-queens. In Liu, J., Zhong, N., eds., *Intelligent Agent Technology: Systems,* *Methodologies, and Tools.* The World Scientific Publishing Co. Pte, Ltd. 110-120.

Kennedy, J.; Eberhart, R. C.; and Shi, Y. 2001. *Swarm Intelligence*. San Francisco: Morgan Kaufmann Publishers.

Liu, J.; Han, J.; and Tang, Y. 2002. Multi-agent oriented constraint satisfaction. *Artificial Intelligence* 136:101-144.

Liu, J.; Jin, X.; and Han, J. 2002. Distributed Problem Solving Without Communication. *Intern. Journal of Pattern Recognition and AI* 16(8): 1041-1064.

Maniezzo, V.; Gambardella, L. M.; and De Luigi, F. 2004. Ant Colony Optimization. In Onwubolu, G. C., Babu, B.V., eds., *New Optimization Techniques in Engineering*. Springer-Verlag. 101-117.

Sosic, R., and Gu, J. 1994. Efficient local search with conflict minimization: A case study of the N-queen problem. *IEEE Transactions on Knowledge and Data Engineering* 6(5):661-668.

Weyns, D., and Holvoet, T. 2003. Model for Situated Multi-Agent Systems with Regional Synchronization. *10th Intern. Conf. on Concurrent Engineering, Agents and Multi-Agent Systems CE 2003*, Balkema Publishers.