THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF ENGINEERING

DEVELOPING A KAKURO PUZZLE SOLVER USING SWARM INTELLIGENCE

MATTHEW J. SHUSTER
SPRING 2010

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Software Engineering
with honors in Software Engineering

Reviewed and approved by the following:

Xiaocong Fan
Assistant Professor of Electrical and Computer Engineering
Thesis Supervisor

Wen-Li Wang
Associate Professor of Electrical and Computer Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Kakuro is a non-polynomial (NP) complete, highly-coupled numerical puzzle that visually resembles a crossword puzzle, but involves mathematical combinations. The primary motivation for solving it stems from the fact that such a solution can be modified to solve other real-world problems, such as data storage utilization, circuit wiring, and multiprocessor scheduling, which are also considered NP-complete problems. The Kakuro-solver developed is based on the concept of swarm intelligence, an artificial intelligence (AI) built on the communication and learning that occur between numerous problem-solving agents. These interactions are facilitated via shared conflict data. Through learning, these agents are able to find better possible answers based on a set of heuristics, eventually developing the puzzle's correct solution. These heuristics govern the modifications made to a potential solution, and are vital to solving success. Experimental results show that the program created is time-efficient, and is capable of solving several puzzles. Also, a discussion is presented for the advantages and disadvantages of the algorithmic-based and AI-based solving approaches. Future work will focus on observing solving patterns and modifying the AI to increase solving efficiency based on these findings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Kakuro is a non-polynomial (NP) complete, highly-coupled numerical puzzle that visually resembles a crossword puzzle, but involves mathematical combinations.[1] The ultimate goal of this research endeavor was to develop a program that is of some help to society. Because Kakuro is a highly-coupled problem, it is inherently similar to many other problems that affect society, including but not limited to multiprocessor scheduling, data storage utilization, and circuit wiring optimization. However, Kakuro, along with other NP-complete problems such as Sudoku, Heyawake, and the traveling salesman problem are all viable research topics in this respect. In particular, Kakuro was chosen as the primary focus of study because of its easy set of rules, simple concept, and amount of instructional information available on rules and possible solving techniques. An in-depth summary of the concepts and complexity of Kakuro will be presented in Section 2, along with other potential applications for a Kakuro-solving artificial intelligence.

To solve the Kakuro problem, a solving technique had to be chosen that yielded a time-efficient and highly-modifiable solution. For this reason, swarm intelligence was selected as the basis for the Kakuro-solver. Swarm intelligence is based on the collective behavior of several solving agents who work together in order to refine a potential solution to a problem until the correct solution is reached. Such behavior is modeled after real-life swarms, such as ant colonies, where the strength of the whole is dependent upon the interconnection and learning performed between the individuals. For instance, ants lay down scents, or pheromones, to direct each other to various resources within their environment.[2] Using this analogy, swarm intelligence is a "decentralized problem-solving system composed of many relatively simple interacting entities."[2] These entities are solving agents who act just as an ant would within the colony, sharing their results with the other agents via data left from their own operations, allowing their successors to make better-informed judgments about the problem's solution. Swarm intelligence was chosen as the solving method for the Kakuro-solver application due to this unique property, with respect to other possible techniques. Although developing a program based on algorithms or an intelligent agent could theoretically accomplish the same task, swarm intelligence presents a distinct opportunity to develop a system that is comprised of several, simple agents that, when used together, are capable of performing a complicated task. These agents have nothing more than simple heuristics to guide their learning and communication with each other, making the development of a swarm intelligence a unique and rewarding learning opportunity.

To begin, a detailed explanation of the concepts and complexity of Kakuro must be presented.

2 Kakuro

2.1 The Basic Concepts of Kakuro

Each puzzle is made up of three types of cells. Black cells are simply used for formatting purposes, full white cells are used to contain player values, and triangular white cells, or restriction cells, are used to identify numerical restrictions for each sequential set of cells, henceforth referred to as "rooms". Rooms may be horizontally or vertically oriented and are composed of set of cells bounded on the top or left by a restriction cell, and on the bottom or right by a non-white cell. A triangular cell with a value in the lower left-hand corner serves as a restriction placed on the vertically oriented room beneath it, while a triangular cell with a value placed in the upper right-hand corner acts as a restriction on the horizontally oriented room to its right.[1] Puzzles can vary widely in size, but for illustration purposes, a simple 8x8 Kakuro puzzle is shown in Figure 1.

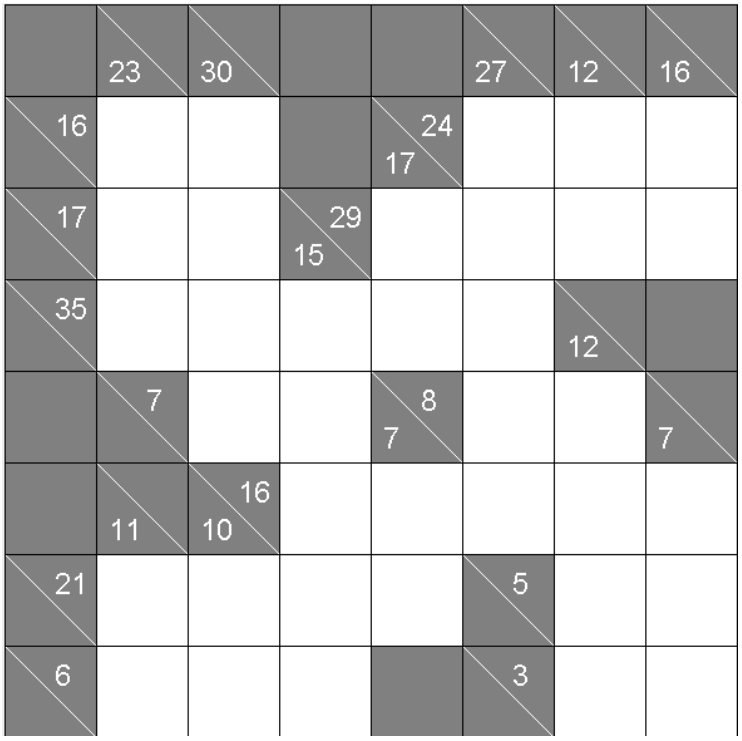

Figure 1: Simple 8x8 Kakuro Puzzle

Each cell in Kakuro may contain one integer from the range 1 to 9, and the sum of the cells within each room must be equal to the restriction value placed on that room.[1] To further illustrate this concept, given Figure 2, the room with 16 as a restriction could have values 9 and 7, corresponding to the two cells upon which the restriction is placed. Similarly, in the vertically oriented room with 23 as the restriction, possible values for cells in the room could be 9, 8, and 6.

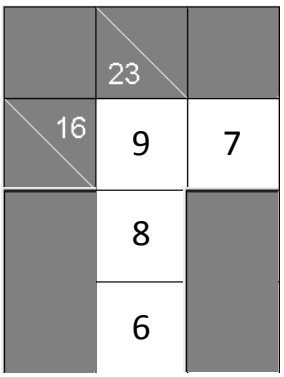

Figure 2: Sample Intersection of Rooms in a Kakuro Puzzle

Another rule for completing a Kakuro puzzle is that each room cannot contain duplicate values.[1] The example values given in Figure 2 result in a correct solution that sums to the correct value and is without duplicates. However, if values of 8 and 8 were given to the horizontal room, the sum of the room would be correct, but a cell value would be duplicated, thus resulting in an incorrect solution.

The above issue is one of two types of "conflict" that may occur within a Kakuro puzzle. The other arises when a room's sum value does not equal the restriction value placed upon it. It is the job of the swarming AI to eliminate these conflicts and develop the correct puzzle solution.

The final rule of Kakuro, which adds further difficulty for the AI states that the values that sum to equal the restriction may be in any order, but each puzzle only has one correct solution.[1] This rule ensures that the swarm intelligence developed must not only determine the values that correctly sum to the restriction, it must also determine the correct order of the values such that there are no conflicts.

2.2 The Complexity of Kakuro

Kakuro, while simple, has a set of rules that make it challenging to solve - it is a game that lacks an algorithm that can consistently solve it in linear time, hence its NP-completeness. NP-complete is "the complexity class of decision problems for which answers can be checked for correctness [...] by an algorithm whose run time is polynomial in the size of the input ... and no other NP problem is more than a polynomial factor harder."[3] In other words, the solution for an NP-complete problem can be verified relatively quickly by an algorithm, and once an algorithm for solving has been developed for an NP-complete problem, it can be translated into a solution for another NP-complete problem.[3] This second criterion, and examples of other applications of a Kakuro-solver will be discussed in Section 2.3.

In general, the complexity of an NP-complete problem stems from the fact that the time to solve increases dramatically if the solution is not already known. The time complexity in such a case is said to be of the exponential order, a representation of which shown below where $T(n)$ is time complexity, n corresponds to the size of the problem, and m is the number of choices available for each element within n.[4]

$$T(n) = m^n \quad (1)$$

Given this equation, it is easy to observe the fact that as *n* increases, so too does the time complexity of the problem, making it even more difficult to solve in an efficient manner.

2.3 Other Applications of a Kakuro-Solver

The applications of a Kakuro-solver in the real-world are nearly endless. Although research was singularly focused on developing a swarm intelligence capable of solving a Kakuro puzzle, the application developed can be used to address many other problems that affect society, such as multiprocessor scheduling and data storage utilization.

Multiprocessor scheduling is an optimization problem that can effectively be explained using an allegory involving the scheduling of airline maintenance periods where there are *n* maintenance jobs to perform and *m* maintenance facilities which can be used to perform each job.[5] The problem lies in how to perform all *n* jobs on the *m* machines in the shortest time possible, with each job n_i having a time requirement l_i. The same issue exists with a multiprocessor CPU, in which jobs must be scheduled to run so that total execution time is the shortest possible. The way in which a swarm intelligence could be applied in this instance is relatively easy to see. The AI, given the jobs, their lengths, and the number of possible processors can begin ordering each job and, given the total execution time of the chosen configuration, change or modify the answer until no better scheduling solution exists. Such an implementation would involve many of the functions already present within the Kakuro-solving agents, however, scents would most likely be applied to the timeslots available on the processors, as well as the jobs that could possibly occupy them, mirroring the solution already developed for Kakuro involving cells and possible values.

Another possible application of the Kakuro-solving AI involves bin packing, a data storage problem. This problem, like Kakuro, is a combinatorial optimization problem that can be solved using a swarm intelligence based on heuristics. In a traditional bin packing problem, there exists a set of items, *S*, with each item s_i within the set carrying a weight of w_i. The items must be packed into bins which have a weight-carrying capacity of *C*.[6] The intent is to organize the storage of the items in the bins such that the minimal number of bins are needed. This roughly corresponds to how a computer stores data on a hard drive - it attempts to organize the data in such a way as to utilize storage capacity. The Kakuro-solving AI could be used in this case, much like in the previous real-world application. Scents could be applied to each item, as well as to each storage bin, allowing the ants to determine which items fit best in which bin. Each solving iteration would seek to refine the solution for the fewest number of bins possible, much like each solving iteration for Kakuro focuses on determining a conflict-free puzzle solution.

Indeed, because a solution developed for an NP-complete problem must be translatable into a solution for any other NP-complete problem, as discussed in Section 2.1, the AI-based Kakuro solver can be modified to address other NP-complete problems, of which there are many

more than the two aforementioned examples. This ensures that the research performed in developing a swarm intelligence for solving the Kakuro puzzle is not limited to just this problem, increasing its potential usefulness to society.

3 Methodology Behind a Kakuro-Solver

Although there are several different types of AI that could be created to solve the Kakuro problem, the intent of this research was focused on the creation of AI based on swarming artificial intelligence.

What makes swarm intelligence both unique, and challenging, is the fact that "swarm-intelligent systems are hard to 'program' because the paths to problem solving are not predefined but emergent […] and result from interactions among individuals and between individuals and their environment as much as from the behaviors of the individuals themselves."[2] Also, because swarm intelligence is based on global behavior, it requires "thorough knowledge not only of what individual behaviors must be implemented but also of what interactions are needed to produce such or such global behavior."[2] To this end, a more detailed discussion of the various components used to implement this methodology is in order. There are essentially three guiding factors that help form the methodology behind creating a swarm intelligence: AI behavior on a local level (the behavior of the problem-solving agents), AI behavior on a global level, and the problem of optimizing a solution to a local, rather than global optimum.

3.1 Solving Agent Behavior

Solving agents, interchangeably referred to as ants, form the basis of swarm intelligence. One of the more interesting aspects of this kind of design is that each agent, as an individual entity, is not "smart" – it cannot solve the puzzle on its own; much like an ant from the natural world would be unlikely to survive on its own without the help from its peers.[2] Therefore, swarm intelligence design focuses on keeping the solving agents simple, and allowing them to learn from one another using highly-connected relationships. The "learning" performed by the agents is key.

Part of the learning that the AI performs involves each ant within the Kakuro-solving AI having a local copy of the Kakuro puzzle to be solved and using two double-precision numerical values in order to track, on one level, how many correct a cell is, and on another level, which of a cell's possible values causes the fewest conflicts throughout the puzzle. Each cell has a scent value applied to it, and each cell is associated with an array of its possible values, each of which also has a scent value. These numerical values roughly correspond to the scent that ants in the natural world use to inform their peers of resources or hazards in their environment. The scent value assigned to individual cells is based on a heuristic, shown below in Equation 2, which takes into account the correctness of the room as a whole, as well as the number of duplicates that may or may not be present within the room or intersecting rooms. *C* represents the cell scent,

R corresponds to the limit, or restriction, imposed on the room in which the cell resides, *S* represents the sum of the values in the room, *V* represents the size of the room that contains the cell, and *D* corresponds to the number of times the value within the cell is present within the room.

$$C = \frac{|R - S|}{V} + (D - 1) \tag{2}$$

The rationale behind this heuristic lies in the fact that it must take the two forms of conflict that may occur during the solving process (i.e. duplicate values and incorrect room sums) and combine them to form one piece of data that can effectively represent a cell's correctness. The first half of Equation 2 takes into account the absolute difference between a room's sum and the room's limit and divides this value by the number of cells in the room. In essence, this ensures that each cell in the room is equally responsible for the room's correctness. Thus, they all have a chance of being modified by an ant. The second half of Equation 2 determines an additional amount of scent based on the number of times the value within a given cell is duplicated throughout the room. This greatly increases the scent, which makes the cell even more likely to be chosen for modification, which will hopefully result in removal of the duplicate.

Because this heuristic is room-dependent, the function above must be run for each room within the puzzle, thus, the scent for a cell is actually the aggregate scent calculated for each of the rooms in which the cell can be found. One would assume this would unfavorably affect cells that occur at an intersection between rooms. However, because the puzzle is highly-coupled, and every cell can be found within two rooms, as exhibited in Figure 1, this culmination of scent applies to all cells, thus, the overall effect of calculating the scent in this fashion is negligible. Overall, the cells will have a scent that takes into account its correctness with respect to both of the rooms in which it resides, with a much greater scent given to those that have a duplicated value.

The possible value scent is initialized using the heuristic shown below, and is modified by the ants throughout program execution in order to determine the best value for the cell that has been chosen for modification. The aim of this equation is to give all possible values within a room the same probability of being chosen upon initializing the solving process. *P* corresponds to the initial possible value scent, while *L* represents the number of possible values the cell may have.

$$P = \frac{1.0}{L} \tag{3}$$

The scent values are used by the ants to determine which cell they should modify, and once the cell is chosen, which of its possible values should be chosen as the new cell value. Scents at the cell level are indicative of the conflicts remaining in the puzzle, and allow the ants

to optimize a solution by modifying a cell with a new value. Much of an ant's behavior is less about hard-coded algorithms or actions mandated by the programmer, and more about behavior modeled over time using scents. This supports the assertion that "swarm-intelligent systems are hard to 'program' because the paths to problem solving are not predefined but emergent."[6] Because scent generally corresponds to how correct a cells value is, the higher the scent, the less-correct a cell is considered to be, and the more probable it is for an ant to choose the cell for modification. Each ant chooses a cell in the puzzle using a heuristic which can best be modeled using natural language. With a 90 percent probability, the ant will choose the cell in the puzzle with the highest scent value, the one with the most conflicts associated with it. With 10 percent probability, the ant will randomly choose a cell, regardless of its scent value. This ensures that most of the time, an ant will choose the worst, most conflicted cell in the puzzle to modify, hoping to change its value such that it is no longer associated with any conflicts. However, in order to avoid optimizing to the local optimum rather than the global optimum, the ants will also choose to modify a random cell, regardless of its scent, 10 percent of the time. The local optimum problem, one of the guiding factors in developing a swarming intelligence methodology, will be discussed in more detail in Section 3.3.

Upon choosing a cell, an ant must then choose a new value for it, preferably one that will result in fewer conflicts. To do this, it will again use scents in order to model its behavior. Each cell has associated with it an array of its possible values along with their scents. These scents are initialized as shown in Equation 3, and work much like those for the cell. In this case, they are used to determine which possible value an ant should pick for the cell. With a 90 percent probability, the ant will choose the possible value with the highest scent value, the one with the most conflicts associated with it, and with 10 percent probability, the ant will choose a value based on the scent of the values, with a higher priority going to those with a higher scent. With this heuristic, there are a few caveats. First, the current value of the cell cannot be chosen as the new value. Second, if there are several values with a scent equal to the worst scent, one from this set will randomly be selected as the replacement value. At first glance, the logic used to model this heuristic may seem incorrect. In theory, it would make sense to have an ant pick the possible value with the lowest scent 90 percent of the time, the value that has fewer conflicts associated with it. However, continually picking the best possible value may lead to the local optimum problem, one of the generally accepted pitfalls of a swarm intelligence. In instances where the best possible value is chosen, the cell scent will reflect this, and the cell itself will be less likely to be chosen for modification by subsequent ants. In summary, while the AI chooses the worst value most of the time, when it does choose the best value, the cell will be more likely remain unmodified. Therefore, optimization to the puzzle solution is ultimately achieved.

After choosing a cell and modifying its value, each ant modifies the scent of the cell and possible value chosen so that subsequent solving agents will be less likely to pick that cell and value. This allows for a breadth of different puzzle solutions to be generated, giving the AI a better chance of solving the puzzle quickly. The cell scent modification performed by the ants is shown below in Equation 4 where C corresponds to the new cell scent, O corresponds to the

current cell scent, A is the number of solving agents in the system, and E corresponds to the number of cells in the puzzle. Equation 5 shows the modification performed to the scent of the possible value chosen for the cell where P is the new possible value scent, and N is the possible value's current scent. A and E are the same values as shown Equation 4.

$$C = .9 \times O + .1 \times \frac{1.0}{A \times E} \quad (4)$$

$$P = .9 \times N + .1 \times \frac{1.0}{A \times E} \quad (5)$$

These heuristics are flexible. The actual modification performed is not important, so long as the result is a new scent value that is smaller than it was previously. These heuristics may change the solving time of the AI, but as long as they decrease the scent so that other cells and values may chosen, the desired effect is achieved. The rationale for the specific heuristics given in Equations 4 and 5 lies in a desire to sufficiently decrease the scent of the cell and possible value without minimizing them to the point that they are never selected for modification. The above equations, over time, will minimize to the point that they equal the value represented by 1 divided by the product of the number of agents and the number of cells in the table. This ensures that there is a base value for scents that is proportional to the size of the puzzle, and that no cell or value is ever inaccessible by the ants. This, in turn, helps avoid the local optimum problem.

After performing all of the aforementioned tasks and updating the global puzzle with the scent modification process described above, the ant, for all intents and purposes, is done executing until being run again by the global level of the AI in order to further optimize the global puzzle solution. In between ant executions, control of the program passes to the global level, which will be described in-detail in Section 3.2.

3.2 Global AI Behavior

Essentially, within a swarming AI, there are two levels of learning that take place: one on the local level and one on the global level. The learning the ants perform as they pick cells and values based upon previous ant choices reflects the former of the two. Global learning to this point has not been discussed, and involves almost all AI functionality outside of the solving agents.

The global functionality of a swarming AI can be compared to the colony that ants in the natural world form. Both maintain and direct the ants for the betterment of the whole. Upon program execution, the global level of the AI starts and initializes the ants, giving each a local copy of the global puzzle. Following this step, and for the remainder of program execution, it runs the swarm of ants continuously until there are no longer any conflicts in the puzzle. We will consider each loop through this functionality a “solving iteration”. During the first solving iteration, ranges of possible values are assigned to each cell. This eliminates the possibility of

assigning a value to a cell where the value cannot exist, given the restriction and number of cells within the room that must sum to equal the restriction. The pseudocode for the range-finding functions is given below. Equation 6 shows the steps taken for finding the minimum possible value of a cell, while equation 7 shows the steps taken for finding the maximum possible value of a cell.

$$
\begin{aligned}
&min \leftarrow cell\ restriction \\
&s \leftarrow 9 \\
&\boldsymbol{for}\ i\ \leftarrow 0\ \boldsymbol{to}\ room\ size - 1 \\
&\quad min\ \leftarrow min - s \\
&\quad s\ \leftarrow s - 1 \\
&\boldsymbol{if}(min < 0) \\
&\quad \boldsymbol{then}\ min = 1
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
&max \leftarrow cell\ restriction \\
&s \leftarrow 1 \\
&\boldsymbol{for}\ i\ \leftarrow 0\ \boldsymbol{to}\ room\ size - 1 \\
&\quad max\ \leftarrow max - s \\
&\quad s\ \leftarrow s + 1 \\
&\boldsymbol{if}(max > 9) \\
&\quad \boldsymbol{then}\ max = 9
\end{aligned}
\tag{7}
$$

Because finding the range of possible values is conducted on a per-room basis, the functions found in Equations 6 and 7 will run for each room the cell is found in. To determine the range of values a cell may have with regard to all of its rooms, the function shown in Equation 8 is run, effectively determining the absolute range of values a cell may have.

$$
\begin{aligned}
&\boldsymbol{if}(min > currentMin) \\
&\quad \boldsymbol{then}\ currentMin = min \\
&\boldsymbol{if}(\max < currentMax) \\
&\quad \boldsymbol{then}\ currentMax = max
\end{aligned}
\tag{8}
$$

Finding the range during the first solving iteration mimics how a human would play Kakuro, where their first step would be to eliminate all invalid cell values from the list of a cell's possible values, making solving much easier to perform.

Also during the first iteration, the AI, on a global level, instructs each ant to randomly choose cell values for all of the cells within the puzzle. Initially, all cells contain null values because they have not yet been assigned an integer. This first iteration allows ants to choose cell values regardless of scent so that the cells may be initialized with, at least, some value. After each ant has randomly assigned a value to each cell, the AI globally calculates the number of conflicts present within each ant's puzzle, thus determining which puzzle solution is the closest to being correct. The puzzle solution with the fewest conflicts serves as the basis for the rest of

the solving procedure, which is performed by the ants using the methods described in Section 3.2.

An important design decision to note at this point is that after the first solving iteration, the ants are restricted to picking one cell to modify, as implied by the heuristics presented in Section 3.1. Although the AI could be designed to allow each ant to modify the whole puzzle, as is done in the first, initializing iteration, creating it in this way keeps the swarm from modifying a cell that could possibly be correct. By having an ant modify the worst cell 90 percent of the time, cells with the most conflicts associated with them are isolated and altered in hopes that it solves those conflicts. Meanwhile, cells that are associated with few or no conflicts are kept from being modified for no reason. Generally speaking, this behavior, repeated over time by the ants, results in an optimization toward the puzzle solution.

In the second and subsequent solving iterations, all agents are run on the most up-to-date global puzzle. To reiterate the series of actions that occur at a local level, the ant will observe the puzzle, choose a cell as described in Section 3.1, choose a value for that cell, and then decrease the scent for both the cell and the possible value using Equations 4 and 5 respectively. At this point, the AI will recalculate the cell scents for the rest of the ant's local puzzle, using the method outlined in Equation 2, which accounts for inaccurate sums in a room as well as duplicate values. The total number of conflicts for the ant's local solution is calculated by evaluating the difference between the room restrictions and the sum of the values in the rooms for each room in the puzzle. If the number of conflicts for this ant's solution is less than that of the global solution (which is the best from the previous iteration), the ant's identifier is stored so that its solution may be referenced after each ant has executed. This process continues until each ant has executed its solving logic.

After each ant has been allowed to run, the identifier stored by AI will be that of the ant with the best solution from the previous solving iteration. Using that solution, in hopes that optimizing the best solution from the most recent solving iteration will result in an optimization toward the puzzle's solution, the AI then modifies the all possible value scents in the global puzzle by using the equation shown below, where P is the new global puzzle possible value scent, S is the current possible value scent in the global puzzle, and V is the possible value's scent in the ant's local puzzle.

$$P = .9 \times S + \frac{.1}{V + 1} \tag{9}$$

As with all the other heuristics developed for the swarming AI, the above function is flexible in its design as long as it results in the desired outcome. In this case, the scents in the global puzzle should be modified, but should not outright change to those the ant has in its local puzzle. This would create a scent that fluctuates with each iteration, making it difficult to optimize the puzzle. Instead, the possible value scent takes into account both the current global value as well as the new local value in order to create a new scent that reflects both.

After modifying the possible value scent for each value that has been used in the current best puzzle solution, the AI globally updates the cell scent for the cell modified by the ant. The logic behind this modification harkens back to the idea that the AI should modify the puzzle solution one cell at a time, in hopes of eliminating conflicts and preserving cells that may contain correct answers. To do this, the AI uses the same heuristic as shown in Equation 9, the difference being the type of scent. In this case, the cell scent would be altered instead of the possible value scent.

After performing both scent modifications on the global puzzle, the AI then tries to find any cell where the scent is 0.0. If this occurs, it may indicate one of two scenarios. Either the cell and value are correct, or the room's restriction and sum are equal and without duplicates, but the cell and its value are not the correct values for the puzzle solution. In order to keep the AI from not choosing the cell if the latter happens to be the case, any cell with a scent of 0.0 is given a minimal value based inversely on the number of times it has been chosen. Thus, if a cell has been chosen many times, it will have a very small base scent assigned to it, while a cell that is only been chosen a few times, yet has a scent of 0.0, will have a larger base scent assigned to it. This ensures that the AI will always be able to pick a cell to modify, and that precedence is given to cells with a high conflict, not those whose conflict has been found, either correctly or incorrectly, to be 0.0. In this case, the logic implemented is also meant to ensure that a cell with a scent of 0.0 that has been chosen very few times is more likely to be chosen over a cell that also has a scent of 0.0, yet has been chosen many times. The justification here is that a cell that has been chosen many times, yet has been found to have a scent of 0.0 is more likely to be correct, as it has undergone several solving iterations. Such a practice also helps the AI from falling into a local optimum when attempting to find the best puzzle solution, a problem that will be discussed in more detail in Section 3.3.

3.3 The Local Optimum Problem

One of the major issues that may occur throughout the solving performed by the swarm intelligence is the optimization of a solution to a local optimum, rather than the global optimum, a problem which is illustrated in Figure 3.

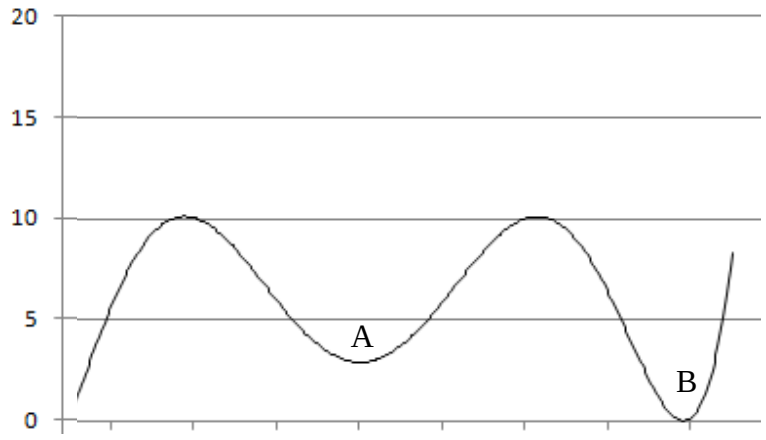

Figure 3: Example Graph of Puzzle Conflicts Remaining

To continue the allegory involving ants and ant colonies in the natural world, imagine a colony of ants in an environment. If the ants have found a natural resource, such as water, the scent along the path leading to the water will be very strong, as there have been many ants that have taken the path, thus leaving behind large amounts of scent. However, there is the possibility of an even more plentiful water resource existing within the environment. If each ant followed only the path with the highest amount of scent, all ants would gather from the first resource and may never find the second. In this case, a local optimum as been reached; the ants have found a natural resource that they can use, but it may not be the best one within the environment. With respect to the graph above, the colony has fallen into the local optimum at letter A. However, the global optimum, or the correct answer, is located at letter B. Thus, to be as efficient as possible at finding the best possible resources in the environment, most ants should follow the strongest scent, while some should attempt to forge their own paths toward new resources. One of these ants in the second group may find the better water resource, thus, the ant colony has exited the local optimum at A, and has fallen into the desired global optimum at B, finding the best possible resource in the environment. Such activities make the colony as efficient as possible at finding the best possible resources from which to gather, which is the ultimate goal.

In terms of the Kakuro-solving AI, if we consider the measure of correctness of the Kakuro-solving AI to be the number of conflicts remaining in the puzzle, then an AI that has found the correct puzzle solution should find that there are 0 conflicts remaining to be fixed. Because ants learn from one another, and each one is programmed to find the absolute best solution, it can be assumed that each ant will take the scent left by the preceding ants and use it to make an informed decision of how to eliminate the most conflicts. In the figure above, A is considered to be a local optimum for the conflicts remaining as the AI solves the puzzle. If the swarming intelligence is programmed such that ants always choose to modify the worst cell value, it may fall into this optimum, instead of going to B, which is the global optimum - the "puzzle solved" scenario. If this is the case, the number of unique solutions developed by the AI is decreased, as are its chances of developing the correct answer. Indeed, there may be an

instance where the AI appears to halt because it has fallen into a local optimum and the ants continue to make the same choices as their peers.

To address this problem, the heuristics shown in Sections 3.1 and 3.2 are implemented so that not only are the decisions made by ants based on calculations, they are also made based on probability in an effort to reduce the impact made by the local optimum problem. A common probability used throughout the heuristics developed for the Kakuro-solving AI involves performing the desired task with 90 percent probability, and another desired, but not optimal task with 10 percent probability. If the AI begins to fall into a local optimum, the functions performed with 10 percent probability will break the cycle of faulty optimization and get the swarm to begin optimizing to the correct solution. However, because ant behavior is modeled based on behavior developed dynamically as a result of scents, this is not always a guarantee.

After obtaining the correct answer, avoiding the local optimum problem is the top issue that must be addressed when developing a swarming artificial intelligence.

4 Experimental Results

To validate the methodology used in creating a Kakuro-solving program based on swarm intelligence, testing was performed to validate whether the heuristics developed throughout the research phase of the project would indeed yield a viable puzzle solver. The AI was developed in the Java programming language and tested using a computer with the following specifications:

Table 1: Experimental Environment Specifications

Processor:	Intel Core 2 Quad Q9300 - 2.50 GHz per Core, 1333 MHz Front Side Bus
RAM:	8 GB DDR3 (4 x 2 GB)
Hard Drive:	1 TB SATA 3.0 Gb/sec, 7200 rpm
Graphics Card:	ATI Radeon HD4350 512 MB DDR2 Memory
Operating System:	Windows 7 Home Premium (64-bit)
Java IDE:	NetBeans IDE 6.8

4.1 Experiment 1: Solving a Kakuro Puzzle With 15 Solving Agents

The first, and most obvious experiment to perform involves the time it takes for the artificial intelligence to solve a standard Kakuro puzzle. The test puzzle, shown in Figure 4 is an 11 x 9 Kakuro puzzle developed by Ogawa Minori.[7]

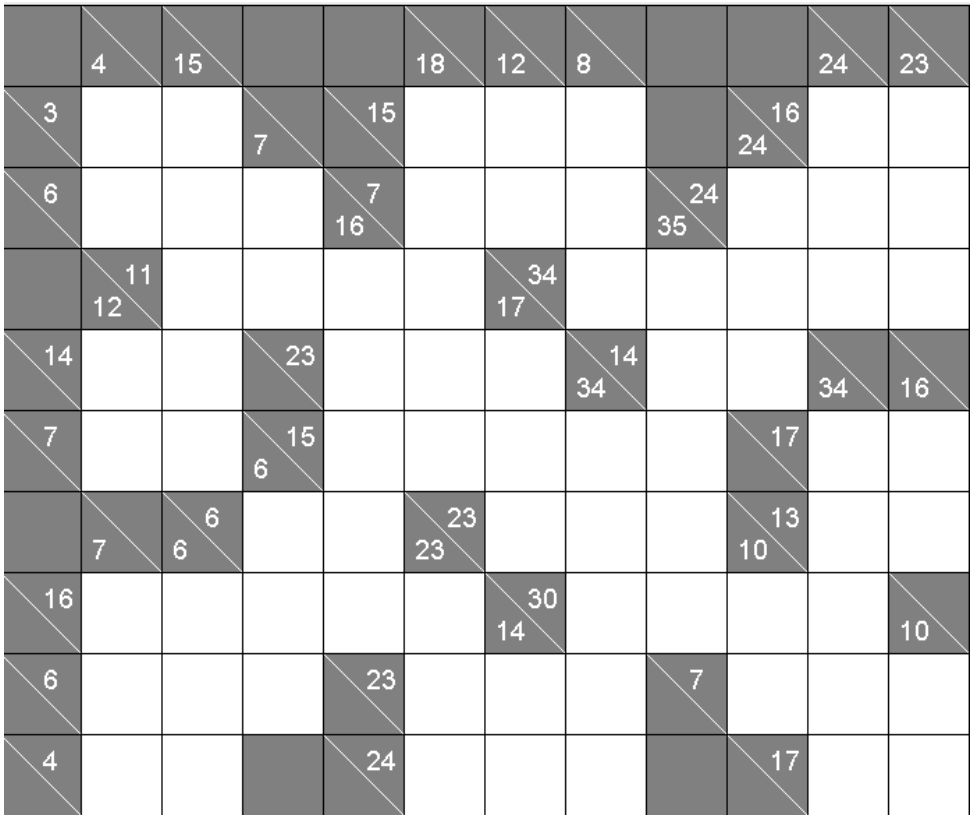

Figure 4: Kakuro Experiment Puzzle

In each experiment performed, the AI uses an open source timing object which begins tracking execution time as soon as the ants initiate their solving iterations and ends immediately after a correct puzzle solution has been found. The first experiment, used to gather baseline data on how the AI performs, involves a series of 100 runs, using a swarming AI with 15 solving agents, each one performing the tasks described in Section 3.1. The results of the experiment are as follows, with times given in milliseconds.

Table 2: Experiment 1 Results - 15 Solving Agents

Min Run Time:	2968 ms
Max Run Time:	169979 ms
Average Run Time:	78098.7 ms
Median Run Time:	71230.5 ms

The time taken by the AI to solve the Kakuro puzzle in Figure 4 ranged anywhere from 169979 milliseconds, or about 2 minutes and 50 seconds, down to roughly 3 seconds. This deviation is derived almost exclusively from a combination of the local optimum problem discussed in Section 3.3 and the heuristics used to handle it. As the AI optimizes the solution and some, if not all of the ants utilize the behavior that is given a 90 percent probability of being conducted, the potential solutions that arise are similar and do not yield a viable answer - the AI has fallen into a local optimum. Once this cycle of local optimization is broken using the heuristics described in Sections 3.1 and 3.2, the scents on the puzzle change, allowing the ants to

potentially find the correct solution via further optimization. This may lead to an increase in execution time, which can be seen in the experimental results. Run 2 had a solving time of 3 seconds, the shortest of any of the trials. The reason it was able to solve the puzzle so quickly was due to the absence of the local optimum issue. The ants, during this trial, were able to learn from one another, and the values they chose to insert into the puzzle happened to be correct, or nearly correct. Thus, the AI was optimizing the solution toward the puzzles answer. In run 4, the longest of the trials, the ants were unable to generate a valid answer until they had performed several optimizations. Thus, the functions given a 10 percent probability of being used, with respect to the ant's cell and possible-value choosing heuristics, must have been executed, effectively breaking the cycle of optimization until the puzzle's answer could be derived from a potential solution.

Given the experimental results in Table 2, it is not hard to see how ants perform their functions based on heuristics that utilize probability. Scents could be developed differently for the cells and possible values each time the program is run. For instance, if the AI immediately develops a solution with only 2 conflicts, the scent for the chosen values in that solution will be such that the other ants may pick different values, essentially ensuring that other possible solutions are tested. After each ant has run, if the solution with 2 conflicts is still the best solution, its values and scents will be used to modify the global puzzle from which each ant makes a local copy, and the process repeats. Thus, the time it takes to solve the puzzle during this run may be far less than a trial in which the AI immediately develops a solution with 50 conflicts. Due to the unpredictability of the AI, the opposite could, in effect, also be true. Even though the best solution may contain 2 conflicts, during the next solving iteration, the ants may choose wildly different values, thus, the best solution may then contain 10 conflicts. Conversely, the AI with a best solution containing 50 conflicts may be optimized immediately to one containing 4 conflicts. Given this example, the second trial may have a better chance of being solved faster, due to its optimization to 4 conflicts, while the first trial, although starting out with fewer conflicts, may take longer. The heuristics are developed in such a way that this should not be the case, but it is, in fact, still a possibility. This lends credence to the assertion that heuristics, and the way scents are modified, are the key to determining how fast and effective a swarming AI is at developing a solution to the Kakuro, or any other NP-complete problem.

4.2 Experiment 2: Solving a Kakuro Puzzle With 25 Solving Agents

The next experiment performed on the Kakuro-solving swarming intelligence also focuses on the time taken to solve a puzzle, albeit with 25 solving agents, rather than the 15 used in Experiment 1. The same testing environment and Kakuro puzzle were utilized. The results of Experiment 2 are shown in Table 3 with times given in milliseconds.

Table 3: Experiment 2 Results - 25 Solving Agents

Min Run Time:	9869 ms
Max Run Time:	327155 ms
Average Run Time:	165016.3 ms
Median Run Time:	143617.5 ms

Logic would seem to dictate that 25 ants should be able to solve a Kakuro puzzle faster than 15 ants, given the extra solving iterations involved, and the increased capacity of the AI to develop more potential solutions. However, this does not seem to be the case. Given the running times observed, it can be inferred that more solving agents within the swarm does not constitute a more time-efficient AI. In fact, quite the opposite appears to be true. Visual observations made during each trial seem to support the assertion that due to the increase in solving agents, the swarm was more susceptible to finding a local optimum solution, rather than the global optimum solution. During most trials in Experiment 2, the AI attempted to optimize its best solution, and is some cases developed a puzzle answer with only 2 conflicts. However, after several solving iterations, and failing to develop a better solution, the heuristics broke the cycle of faulty optimization and the AI began developing a new solution. This scenario seemed to occur more frequently when the swarm consisted of 25 solving agents. Experiments 1 and 2 clearly illustrate this point.

4.3 Experiment 3: Solving a Kakuro Puzzle with a Simplified Value-Choosing Heuristic

The ant behavior discussed in Section 3.1 relies on two heuristics - one that determines which cell to modify, and one that chooses the value for the cell chosen. Their main purpose is to keep the AI from optimizing to the local optimum. However, their implementation as specified in Section 3.1 and given the experimental results in Sections 4.1 and 4.2 as well as the behavior observed during program execution hints that there may be a more effective way to choose values for cells. This assertion arises from the fact that if the program is continually reaching a local optimum rather than the desired global optimum, execution time increases. The best way to test this hypothesis was to modify the heuristics.

In this case, the ants value-choosing heuristic has been modified to no longer include two branching functionalities based on probability. Thus, each value has an equal probability of being chosen as the new cell value. Using the new behavior, the ants will simply choose a possible value that is not equal to the current entry. The results of the experiment that tests this new heuristic shown below in Tables 4 and 5. Table 4 shows a summary of the execution times for an intelligence with 15 solving agents, while Table 5 shows experimental results for an AI with 25 solving agents. In each case, 100 trials were performed.

Table 4: Simple Value-Choosing Heuristic - 15 Solving Agents

Min Run Time:	1993 ms
Max Run Time:	77858 ms
Average Run Time:	17020.38 ms
Median Run Time:	12430.5 ms

Table 5: Simple Value-Choosing Heuristic - 25 Solving Agents

Min Run Time:	3224 ms
Max Run Time:	160523 ms
Average Run Time:	35262.93 ms
Median Run Time:	23965.5 ms

Comparing these results to those found in sections 4.1 and 4.2 yields an interesting result. The simplified value-choosing heuristic actually improves the execution times for the AI in both instances, with the swarm containing 15 solving agents still running faster overall when compared to the AI with 25 agents.

The experimental results seem to indicate that the original heuristic, developed specifically to eliminate the local optimum problem is extraneous. The explanation for such an assertion is relatively simple. Because each ant works in the same way as in Experiments 1 and 2, albeit with a different value-choosing algorithm, it still uses Equations 4 and 5 to decrease cell and possible value scents. It also uses the same cell-choosing heuristic as before, which picks the most conflicted cell 90 percent of the time, and chooses a random cell 10 percent of the time. What is important to note is that the conflict of a cell is directly influenced by the possible value that it contains, and how that value affects conflicts within the room and the puzzle as a whole. Because the cell scent is determined based on the number of conflicts in a room (as illustrated in Equation 2), the AI is still able to choose the cell that is most conflicted and modify its value, this time without choosing a value based on its scent. In effect, the removal of the probability-based value-choosing heuristic places more importance on the cell picked rather than the possible value picked for a cell. If a possible value within a cell is nearly correct, this will be reflected in the scent of the cell, thus it will have a lower probability of being picked. However, if the value in a cell causes puzzle conflicts, the cell scent will be larger, thus it will be more likely to be chosen for modification. The new cell value is guaranteed to be different from the current value, therefore either increasing or reducing the number of conflicts, which the cell's scent will reflect accordingly. The process will continue, with conflicted cells being chosen over less-conflicted cells, eventually yielding the correct puzzle solution, in a time faster than that of the original AI specification. The Kakuro-solving artificial intelligence, therefore, has been modified to reflect this change in value-choosing heuristics.

4.4 Experiment 4: Comparison between Algorithm and AI-based Kakuro Solving

For the purposes of this experiment, SAAM's Kakuro Solver v1.00, a Java applet, was used in order to compare and contrast the algorithmic and AI-based approaches to solving a Kakuro puzzle.[8] Using the puzzle in Figure 4, SAAM's Kakuro Solver was able to determine the correct Kakuro puzzle in less than 1 second, contrasting greatly with the results of Experiments 1, 2, and 3, where the AI-based Kakuro solver took anywhere from three seconds to four and a half minutes. To better understand the algorithmic Kakuro puzzle solver and its advantages and disadvantages over a swarm intelligence, the steps it takes to find the puzzle solution should be discussed.

4.4.1 The Algorithm-based Kakuro Puzzle Solver

SAAM's Kakuro solver makes use of a 6 step process that allows it to determine a puzzle's solution. During the first step, the solver determines the number of cells in each room, and uses each to determine the possible sequences of values that may occur within the room.[8] For instance, if a room has a restriction of 3, and the room contains 2 cells, the possible sequences found will be {1,2} and {2,1}. The next step the solver takes involves determining the possible values in each cell by using the sequences developed in step 1 and comparing them with those of the intersecting room.[8] By observing which numbers occur in both sequences, it establishes which numbers are possible for each cell. Step 3 in the algorithmic sequence involves assigning a number to a cell if there is only one possible value for it. Step 4 builds off of this action by traversing the puzzle removing the value determined in step 3 from the possible values list of each intersecting cell, removing possible sequences in the process.[8] From here, the algorithm iterates over steps 2, 3, and 4 each time there is a change made to the possible sequences and values lists in the cells, indicating that further optimization toward the puzzle solution may be made.[8] When there can be no more optimization of possible sequences and values, the algorithm then guesses a number for an empty cell, then re-iterates over the previous four steps until there are no more conflicts and each cell contains a value[8].

4.4.2 Comparing the Algorithm and AI-based Approaches

The primary difference between the algorithm-based and AI-based approaches to solving a Kakuro puzzle, other than the obvious implementation details, is the fact that the algorithm used to solve the puzzle by the former focuses on constant optimization toward the puzzle solution whereas the artificial intelligence focuses on the learning throughout execution as to how to solve the problem. One of the disadvantages of the algorithmic approach is its use of a lookup table to determine possible values for a cell. Given the example puzzle in Figure 5, each cell, according to the algorithmic solver, will have a possible range of values from 1 to 9.

	45	45	45	45	45	45	45	45	45
45									
45									
45									
45									
45									
45									
45									
45									
45									

Figure 5: Algorithmic Solver Disadvantage Puzzle

The number of possible sequences developed in the first step will be 362,880 (or 9!) for each room, given the fact that each integer 1 through 9 may appear in any order. During its next step, the algorithmic solver will not be able to eliminate possible values for a cell because each value is possible. The next step, step 3 is not possible because no cell will only have 1 possible value, therefore the algorithm-based solver must guess numbers for empty cells, and then reiterate over the previous optimization steps to determine possible values in the other cells. This process will continue until, eventually, the solver has guessed each cell's value, taking a time to the order to minutes rather than seconds to complete. A swarming AI, however, will use scents to determine whether a value is better in one cell rather than the other, and its solving time will be, generally, shorter. This depends, obviously, on the effect the local optimum problem, and the heuristics designed to deal with it, has on the solving time. If a local optimum is continuously reached instead of the global optimum, then the solving time will be longer, and vice versa. The issue, then, hinges on the size of the puzzle. For small to medium-sized puzzles with high-coupling and many restrictions, the algorithmic solver would be best suited as the solving application of choice. For larger puzzles, with fewer restrictions, the AI-based solving approach shows an advantage, and should be used.

5 Conclusions

The heuristics developed, given the experimental results given in Section 4, provides a viable, time-efficient solution for solving the Kakuro problem. With it, a vast array of other problems may be addressed, as discussed in Section 3.3. Although the solving of Kakuro itself does not present an explicit benefit, the research performed and the program developed may be modified to apply to different domains, which may benefit society. In situations where a large problem involves highly-coupled data, swarm intelligence provides a unique advantage over similar algorithmic solutions in that it does not rely on lookup tables and explicit solving steps. Rather, it utilizes a set of highly-tuned heuristics' that allow its solving agents to make educated decisions on cell and value modifications. Also, as stated in Section 2.3, the swarm intelligence-based Kakuro-solver is highly modifiable, another advantage over similar, algorithm-based solvers.

Future work with regards to the AI-based Kakuro-solver will focus on two facets: optimization of heuristics to increase puzzle compatibility, and observation of solving patterns to better develop local and global AI behaviors.

The heuristics developed throughout this research project, while sufficient to solve some puzzles in a time-efficient manner, were not viable for other, more complicated puzzles. They serve as a good basis for research, and given the data collected during Experiments 1, 2, and 3, modifications can be performed to change how ants modify scents and how they choose cells and values in order to decrease solving time. This is clearly evidenced by the results of Experiment 3, where a simpler value-choosing heuristic was used which decreased solving time. If heuristics are indeed modified, puzzles with increased combinatorial complexity may be solved much faster, and with fewer optimizations performed toward a local optimum. Thus, the singular aim of this facet of future work focuses on increasing puzzle compatibility and time-efficiency via changes to the existing heuristics.

Another, important focus of future work involves the optimization of the application's local and global behaviors by observing solving patterns. The local optimum problem, regardless of the specific swarming AI implementation, plays a major role in such a programs time-efficiency. As could be observed throughout the experiments performed, ants are likely to reach local optimums numerous times before finding the puzzle's correct solution. In an effort to minimize the effect this problem has, the solving patterns should be observed such that changes can be made to local and global AI behaviors. Although simple heuristic changes may result in faster solving times and increased compatibility with complex puzzles, the solving agents themselves should be modified to increase the likelihood of solving in a faster manner. The global AI, too, should be modified if solving patterns indicate a problem, or efficiency issue. For example, perhaps the way the ants are executed, or the manner in which global learning is can be changed to decrease solving time.

These two avenues of future work are possible because of the basis of research developed via this project. They will undoubtedly lead to an AI that is even more capable of helping society, the primary aim of project.

References

1. Gordon P (2006). *The Kakuro Challenge*. Sterling Publishing Co., Inc. New York, NY, USA. pp 3.
2. Bonabeau E, Dorigo M and Theraulez G (1999). *Swarm Intelligence: From Natural to Artificial Intelligence*. Oxford University Press, Inc.: New York, NY, USA. pp 6-7
3. Black P.E, "NP-complete", in *Dictionary of Algorithms and Data Structures*, U.S. National Institute of Standards and Technology (2008). Available from: http://www.itl.nist.gov/div897/sqg/dads/HTML/npcomplete.html
4. Papadimitriou C.H and Steiglitz K (1998). *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Inc. Englewood Cliffs, NJ, USA. pp 343.
5. McCormick, S. Thomas. Smallwood, Scott R. Spieksma, Frits C.R. "Polynomial Algorithms For Multiprocessor Scheduling with a Small Number of Job Lengths". Society for Industrial and Applied Mathematics. September 25, 1996.
6. Levine, J. Ducatelle, F. "Ant Colony Optimzation and Local Search for Bin Packing and Cutting Stock Problems". Journal of the Operational Research Society (2004).
7. Minori O. "Sample Problems of Kakuro Puzzle: Sample Problem 2". Available from: http://www.nikoli.com/en/puzzles/kakuro/.
8. "SAAM's Kakuro Solver v1.00" (2008). Available from http://www.saam007.com/java/?JApp=KakuroSr

ACADEMIC VITA

Matthew J. Shuster

Matthew J. Shuster
183 Bryson Road
Butler, Pennsylvania 16001
mjs5250@psu.edu

Education:

Bachelor of Science Degree in Software Engineering, Penn State University, Spring 2010
Minor in Management Information Systems
Thesis Title: Swarm Intelligence - How the Use of Artificial Intelligence Based on Collective Behavior Solves the Kakuro Problem
Thesis Supervisor: Xiaocong Fan

Related Experience:

Internship with PPG Industries, Inc. in the Architectural Finishes Information Technology Department
Supervisor: Mike Koerbel
Summer, 2009

Awards:

Chancellor's Scholarship Award
Dean's List
Penn State Behrend Honors Award

Presentations/Activities:

Member of Phi Kappa Phi Honor Society
Member of Tau Beta Nu Engineering Honor Society
IEEE Local Chapter Presentation: Process Automation on Code Understanding, Reporting, and Analysis