Proceedings of the 2005 IEEE Congress on Evolutionary Computation.

Multi-Agent Cooperation Using the Ant Algorithm with Variable Pheromone Placement

Eric Borzello[i] and Laurence D. Merkle[ii]

Abstract

The Ant Algorithm was created by examining real life ant colonies and developing an algorithm to use the concept of "stigmergy" to approach multi-agent problems with distributed control. As agents work on tasks, more agents attempt difficult tasks. Task deadlock occurs when agents attempt impossible tasks indefinitely. Previous research avoids task deadlock through adaptive attenuation factors. This research investigates increasing algorithm effectiveness through variable pheromone placement. Results of computational experiments are presented demonstrating the increased effectiveness of the new algorithm.

Key Words: multi-agent system, cooperation, ant algorithm, variable pheromone placement

I. Introduction

There are many applications of multi-agent systems, such as exploring unknown areas, cleaning a house, and automated laboratory systems. Though these applications vary in their specifics, most of them share a key feature – it would be best if the agents in the system could collaborate with each other without any outside human intervention.

A difficulty facing multi-agent system implementations is avoiding situations in which all tasks are completed successfully, but the number of agents working on each task is suboptimal. If too few agents are working then their effort is wasted until more agents arrive. If too many agents are working then the effort of the excess agents is wasted. In either case, the wasted effort implies wasted time steps.

In 2003, Ding Yingying, He Yan, and Jian Jingping [5] applied the ant algorithm discussed by Dorigo and DiCaro [1] to multi-robot system design. The algorithm assigns more robots to more difficult tasks, and fewer agents to easier tasks. This allows the number of robots actually doing useful work to increase, in the sense that the number of time steps required to complete all of the possible tasks decreases. Yingying, et al. modify Dorigo and DiCaro's algorithm by adding an adaptive attenuation constant to eliminate task deadlock. They present results of experiments in which this modification decreases the time that agents spend going back to impossible tasks instead of seeking out and performing tasks they can accomplish.

This research examines a further modification of the ant algorithm for multi-agent systems that uses a novel technique of placing pheromone on difficult tasks. This technique causes agents to place more pheromone on tasks that are near completion, making the agents more likely to choose tasks that can be completed soon. It is demonstrated experimentally that effectiveness is improved as a result.

The remainder of this paper is organized as follows. Section II gives an overview of the ant algorithm in general and a description of the specific version used by Yingying, et al. Next, Section III describes the version of the ant algorithm developed in this research. Section IV describes the software system used as a simulation tool. Finally, Section V presents the results of computational experiments evaluating the effectiveness of the proposed algorithm.

II. Background

Systems composed of multiple agents have several key features that make them desirable for many applications. First, presence of more than one agent allows agents to cooperate in order to achieve goals too difficult for any one agent alone. Furthermore, the presence of multiple agents allows for relative autonomy from human intervention, since many such systems can lose several agents without severe loss of functionality. In contrast, a system of a single agent is completely disabled if that agent is lost.

As a result of these features, many real world problems are approached relatively easily by systems of collaborating agents. Consequently, multiple agent systems are a field of significant interest in the area of artificial intelligence.

Another topic in artificial intelligence is ant algorithms. An ant algorithm is any algorithm in which the individual agents use decision making processes that are in some sense similar to those used by real ants. This general approach has been applied with some success to the traveling salesman problem [1], the single machine job scheduling problem [1], dynamic vehicle routing [2], image rendering [4], and numerous other domains.

Details of ant algorithms vary, but many share core tenets. First, agents mark tasks by placing pheromone. Also, agents are more likely to choose tasks with more pheromone. Thus, agents attract each other by placing pheromone. This indirect communication method, called "stigmergy," is surprisingly effective.

The basic process of this algorithm is as follows. When the agents enter a new area, they have no knowledge of the locations or difficulty of any tasks in that region, or of the positions of other agents. Each agent's motion is determined independently by a random walk, and the agents do not directly communicate with one another. When an agent reaches a task, it attempts that task.

If the agent makes progress on the task, the agent continues until the task is complete. On the other hand, if the agent is unable to make progress, it adds pheromone to the task. The pheromone is added on a "blackboard" that can be read from and written to by any of the agents.

Agents looking for tasks check to see if any tasks are indicated by pheromone on the blackboard. If such tasks exist, the agent will choose one of them rather than continuing its random search. If there is more than one task on the blackboard, then the probability that the agent chooses task *i*, given there are *k* tasks total, is

$$p_i(t) = \frac{\mu^{select_i}\tau_i(t)\eta_i(t)}{\sum_{s=1}^{k}\mu^{select_i}\tau_s(t)\eta_s(t)},$$

where $\tau_i(t)$ is the amount of pheromone on task *i* at time *t*, $\mu \in (0,1)$ is the attenuation rate, $select_i$ is the number of times the particular agent has tried and failed task *i*, and $\eta_i(t)$ is the heuristic value associated with task *i*. For the experiments in Yingying, et al. $\eta_i(t) = 1$.

III. Ant Algorithm Multi-Agent Cooperation

The goal of this research is to improve the effectiveness of the ant algorithm for multi-robot systems proposed by Yingying, et al. The system differs from theirs in the mechanism by which heuristics are incorporated into the algorithm. Specifically, in the system proposed by Yingying, et al., the amount of pheromone placed by an agent on a task in a time step is a constant, while in this research it varies heuristically. Two different pheromone placement functions were explored. One function, the *difficulty-ratio-proportional function*, placed pheromone depending on the difficulty of the task relative to the total strength of the agents currently working on the task, and the other, the *agent-count-proportional function*, simply placed pheromone proportional to the number of agents currently working on the task.

The function which places pheromone proportional to the ratio of task difficulty to agent strength is: $\Delta\tau = \omega + (1-\omega) * (F_c / F_n)$,

where $\omega \in (0,1]$, F_n is the amount of total force needed by the agents to push the ball, and F_c is the amount of force currently exerted by the agents pushing on the ball.

The above has the consequence that more pheromone is placed on balls that already have nearly enough force exerted on them than is placed on balls that need much more force. The goal is to have nearly complete tasks completed sooner so that the agents who are working on them can instead work on other tasks.

The second function simply placed pheromone proportional to the number of agents at the ball, so this function is: $\Delta\tau = \omega + (1-\omega) * (n / k)$,

where $\omega \in (0,1]$, n is the number of agents currently at the ball, and k is a number that bounds the difficulty of all the tasks above. This function was introduced to see if using a constant value for task difficulty, rather than requiring the agents to know the difficulty of each task, could still produce greater effectiveness than adding constant pheromone.

In order to explore these two functions, a specific problem domain was explored. This domain was as follows. A number of balls are placed throughout a flat, featureless plain. The agents' goal is to return all of these balls to the same predetermined goal location. In order to push a ball, the sum of the pushing strengths of the agents must be greater than or equal to the mass of the ball. Therefore, pushing a ball is a task, the difficulty of the task is the weight of the ball, and the sum of the strengths of the agents currently pushing the ball is how close the task is to being completed.

Furthermore, for both functions the value of $\Delta\tau$ is computed using only local information that is available to the agent placing the pheromone. Either the weight of the ball and the total amount of effort being put forth by the agents currently working on the task in one case, or the number of agents currently working on a task in the other. However, the agents that are selecting tasks use only the pheromone amounts and positions, which is information stored in the blackboard. This allows the agents to make informed decisions without needing a priori knowledge about tasks or locations of the other agents that they cannot directly sense.

Three details of both of the pheromone placement functions are noteworthy. First, when $\omega = 1$ this function corresponds exactly to the function used by Yingying, et al. Also, the probability of a agent choosing a particular task is determined by the amount of pheromone placed on that task relative to the total amount of pheromone placed, rather than by the absolute amount placed on that task. Finally, $\Delta\tau \in (0,1]$, because in the case of the first function if $F_n \le F_c$, or if enough agents are working on a task to complete it in the case of the second function, then no pheromone will placed (recall that pheromone is only placed on a task when a agent fails to complete it).

IV. Simulation

This section describes a simulation environment used to evaluate the algorithm discussed above. In this environment, agents have the characteristics discussed in Ding, et al. Specifically, they "know" their current position and the position of the goal, they "sense" agents and the boundary of the environment within their range of perception, they detect and push balls, and they place pheromone on the blackboard. Collisions are not modeled.

The environment is initialized by placing *x* agents in random positions, as well as *y* balls and one goal in specified positions. After initialization, the behavior of agent *i* is determined by its state, which is either *Wandering*, *Proceeding*, or *Working*, and whether or not there is at least one task *s* for which the following condition holds.

$$\phi(i,s) \equiv \left[\mu^{select_i}\tau_s(s) > 0.01\right]$$

- If agent *i* is in the *Wandering* state and $\phi(i,s)$ becomes true for one or more tasks, then it chooses one of those tasks as described above and enters the *Proceeding* state with the corresponding ball as its destination. If $\phi(i,s)$ is false for all tasks *s*, and the agent discovers a ball *b*, then it enters the *Proceeding* state with ball *b* as its destination. Otherwise, the agent takes a step and makes a small random change in direction. Ignoring tasks for which $\phi(i,s)$ is false prevents agents from cycling between impossible tasks.
- Agents in the *Proceeding* state take steps in the direction of their destinations until either they reach the destination or the corresponding task is completed. In the former case, they enter the *Working* state, while in the latter case they enter the *Wandering* state.
- Agents in the *Working* state attempt to move their ball towards the goal. If they are not able to do so, they increment a counter. If the counter reaches a limit (15 in this case), they add pheromone to their task and enter the *Wandering* state. They also enter the *Wandering* state if they are able to push their ball to the goal.

V. Results

The scenarios used in these experiments have the following characteristics. The number of balls initially added to the environment is a multiple of 20. Furthermore, ¼ of the balls require one agent to push, ½ require two agents to push, and ¼ are impossible to push given the number of agents. Each simulation ends when the tasks consisting of pushing the balls to the goal are all completed or determined to be impossible.

Results, given in several tables and graphs below, were averaged over 10,000 simulations per scenario, with independent starting locations for each simulation. Below are tables for both of the two pheromone placement functions discussed above, as well as graphs comparing the number of time steps needed to complete a given scenario for a variety of ω values. Each row of a table corresponds to a different ω value, while each column is labeled with the number of balls in the scenario, while every cell contains the average and the standard deviation of the number of time steps to complete the scenario with the given ω value. Next, the graphs were created by ordering the number of time steps needed to complete the scenario with a specific ω value from least to greatest. Then, a graph with one line for each ω value was created. As shown in the tables, the lower lines were created using the data from the functions computed with smaller ω values.

Several trends are noticeable in the data. First, the more tasks in a scenario the more time steps it takes to resolve them. Next, the functions computed with smaller ω values take fewer time steps to complete a scenario on average. Furthermore, the *agent-count-proportional function* needs fewer time steps on average to complete a given scenario than the *difficulty-ratio-proportional function*. However, the *agent-count-proportional function* also has a higher standard deviation than the *difficulty-ratio-proportional function*. Last, the graph from the scenario with 40 balls shows that when relatively few tasks are present in a scenario there is quite a bit of fluctuation in the number of time steps needed. The data from the 40 ball scenario has a stair-step like appearance, while the data from the later case appears more homogeneous.

VI. Conclusions

When the two new pheromone placement functions are compared with the version from Yingying et. Al. there is a large time savings in both cases. Both functions are consistently more effective than the original function, and this increased effectiveness is constant as the problem size increases. Furthermore, the *agent-count-proportional function* is found to be more effective than *difficulty-ratio-proportional function*. This difference is present in even very large problem instances.

In fact, agents using the *agent-count-proportional function* for ω = .025 completes the scenarios in far fewer time steps than the agents using the *difficulty-ratio-proportional function* on average. For the 260 ball scenario the *agent-count-proportional function* completes the scenario in 75% of the time steps needed by the other function. More importantly, the *agent-count-proportional function* seems to take 25% fewer time steps for all scenarios with a large number of tasks. This implies that as the problem instances become larger and larger the difference in effectiveness, both between the previous version of the algorithm and the current one, and between the two functions presented in this paper, will become more and more noticeable. In fact, it is in such large cases that the data indicates the *agent-count-proportional function* would come into its own. Though a 25% decrease in time steps needed is noticeable if there are few tasks, if a case with millions and millions of tasks were considered then the 25% savings would be a very large number of time steps.

Though choosing the correct pheromone placement function can yield significant time savings, the ω of the function being used also has an effect on completion time. Essentially, as ω decreases in either of the two new functions, the effectiveness of the algorithm increases. Since ω can be understood as how similar to the constant pheromone placement function the newer function behaves, small ω being better implies that moving away from a constant pheromone approach will yield much more effective algorithms.

VII. Future Work

Though two different pheromone placement functions were explored during this research, these functions are certainly not the only pheromone placement functions one could consider. The most direct progress that could be made from this research is to simply consider new pheromone placement functions, and compare them to the ones described in this paper.

Furthermore, both the Yingying et.al's algorithm and the one outlined in this paper allow for a heuristic function to be used during the computation of $p_i(t)$. The use of heuristics was not explored during this research. However, creating ways to improve how agents choose between the tasks will almost certainly be as productive as improving how the agents place pheromone.

REFERENCES

[1] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz, Swarm Intelligence: From Natural to Artificial Systems. (New York: Oxford University Press, 1999)

[2] Darren M. Chitty and Marcel L. Hernandez, "A Hybrid Ant Colony Optimization Technique for Dynamic Vehicle Routing," *Proceedings, 2004 Genetic and Evolutionary Computation Conference,* Seattle, WA, June 2004, 48-59.

[3] Marco Dorigo and Gianni DiCaro, "The Ant Colony Optimization Meta-Heuristic", *New Ideas in Optimization* 1999. (New York: McGraw-Hill)

[4] Yann Semet, Una-May O'Reilly and Fredo Durand, "An Interactive Artificial Ant Approach to Non-photorealistic Rendering," *Proceedings, 2004 Genetic and Evolutionary Computation Conference,* Seattle, WA, June 2004, 188-200.

[5] Ding Yingying, He Yan, and Jiang Jingping, "Multi-Robot Cooperation Method Based on the Ant Algorithm", *Proceedings, 2003 IEEE Swarm Intelligence Symposium,* Indianapolis, IN, April 24-26, 2003, 14-18.

Table 1. Average Time Steps to Complete Simulation with *difficulty-ratio-proportional function*

ω Value	40	80	120
.025	7722.629 ± 779.13	16231.06 ± 768.54	24706 .00 ± 744.42
.05	7729.829 ± 785.17	16277.88 ± 776.70	24772.16 ± 753.25
.1	7734.17 ± 784.76	16307.35 ± 794.42	24851.67 ± 755.02
.2	7752.618 ± 789.96	16373.12 ± 788.76	24941.65 ± 769.67
.5	7806.883 ± 807.64	16462.51 ± 776.13	25058.52 ± 776.89
1.0	7933.934 ± 833.39	16824.07 ± 821.25	25639.84 ± 814.80

Table 1. (Continued)

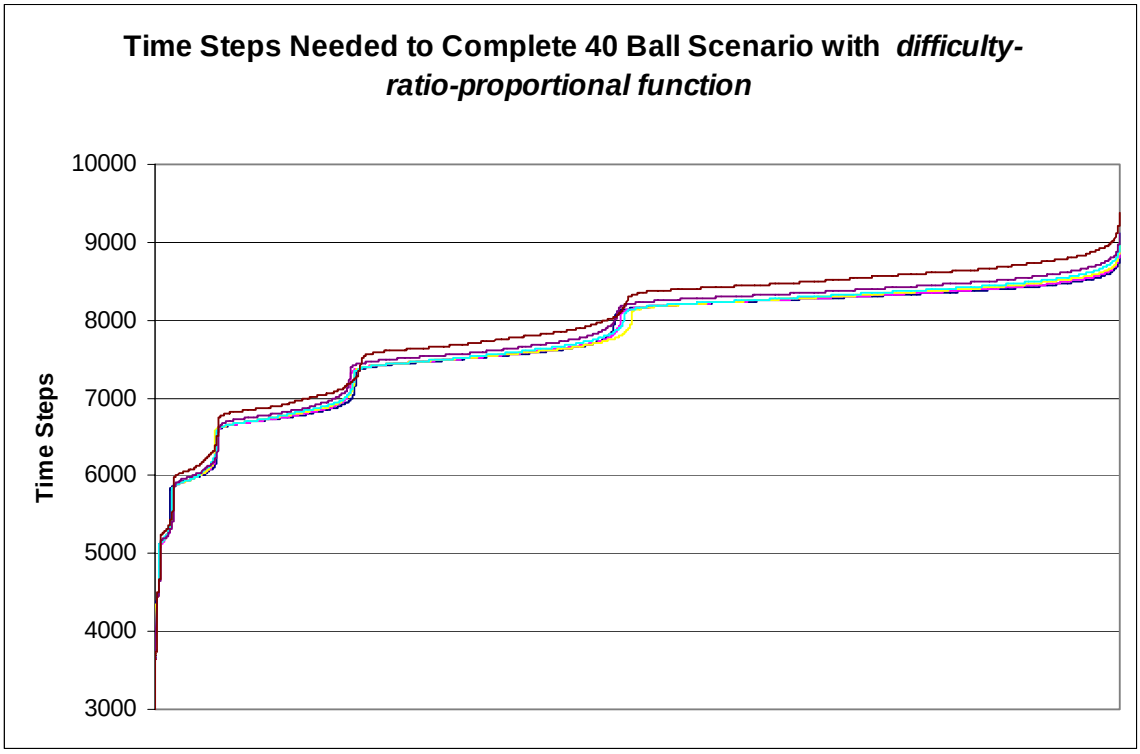
ω Value	200	240	280
.025	41770.6 ± 721.11	50269.61 ± 720.41	58762.66 ± 741.50
.05	41897.55 ± 711.23	50432.18 ± 720.43	58962.77 ± 759.44
.1	42056.29 ± 740.53	50643.61 ± 746.34	59203.74 ± 776.31
.2	42258.19 ± 767.46	50893.1 ± 786.13	59525.77 ± 823.72
.5	42383.3 ± 752.94	51057.38 ± 764.74	59691.52 ± 811.17
1.0	43483.11 ± 814.44	52408.92 ±855.86	61297.78 ± 890.73

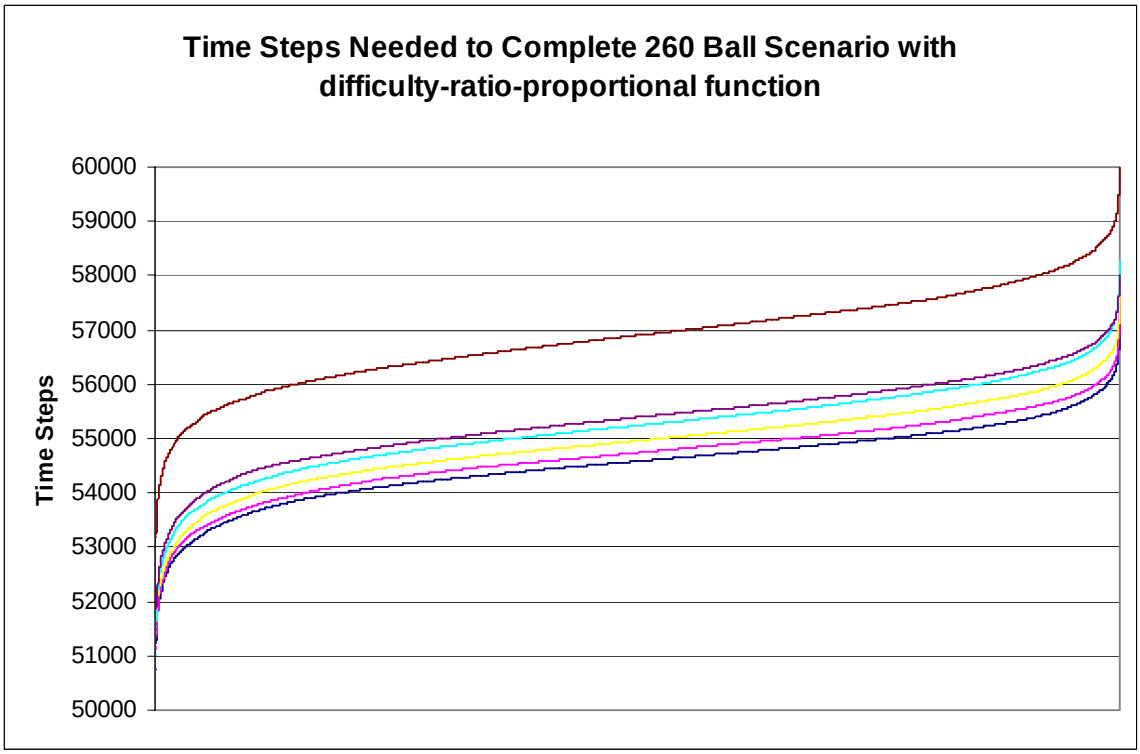
Table 2. Average Time Steps to Complete Simulation with *agent-count-proportional function*

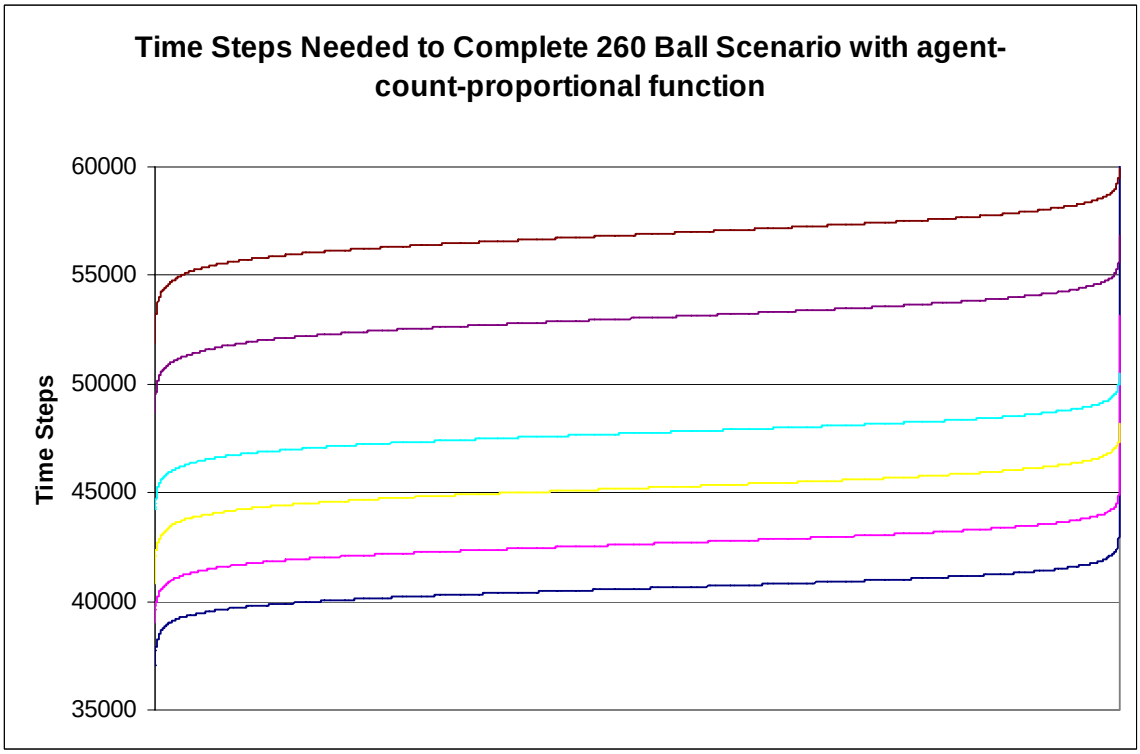
ω Value	40	80	120
.025	5731.206 ± 582.71	12055.49 ± 584.49	18355.02 ± 572.69
.05	5990.564 ± 598.04	12631.18 ± 605.90	19238.56 ± 615.74
.1	6350.047 ± 648.25	13381.95 ± 655.99	20406.93 ± 650.67
.2	6740.698 ± 684.21	14187.91 ± 674.48	21598.86 ± 679.04
.5	7432.0 ± 756.69	15665.7 ± 772.95	23894.94 ± 779.65
1.0	7961.183 ± 816.69	16820.42 ± 825.73	25641.56 ± 826.32

Table 2. (Continued)

ω Value	200	240	280
.025	31089.06 ± 698.0443	37404.85 ± 684.2929	43757.13 ± 666.0753
.05	32633.99 ± 698.0784	39287.28 ± 720.0807	45946.62 ± 718.7256
.1	34586.9 ± 699.2618	41675.6 ± 710.5038	48734.81 ± 744.4938
.2	36542.44 ± 681.3128	44004.62 ± 688.7817	51445.34 ± 724.5629
.5	40535.92 ± 781.5535	48859.5 ± 805.2174	57149.89 ± 867.3206
1.0	43473.13 ± 825.1548	52392.41 ± 867.5637	61312.37 ± 894.6442







[i] Rose-Hulman Institute of Technology, Terre Haute, IN, 47803, Eric.Borzello@Rose-Hulman.edu.
[ii] Rose-Hulman Institute of Technology, Terre Haute, IN, 47803, Laurence.D.Merkle@Rose-Hulman.edu.