# A Modified Ant Colony Optimization Algorithm for Implementation on Multi-Core Robots

Timothy Krentz Chase Greenhagen Aaron Roggow Danielle Desmond Sami Khorbotly

Department of Electrical and Computer Engineering
Valparaiso University
Valparaiso, IN, USA
Sami.khorbotly@valpo.edu

***Abstract*— The Ant Colony Optimization (ACO) algorithm is an evolutionary algorithm that bio-mimics the behavior of ants in finding the shortest path between an origin and a destination within a set of pre-determined constraints.**

**The goal of this work is to create a small-scale application of the ACO using a swarm of small autonomous robots. We investigate the practical applicability of the algorithm in real-life situations by addressing the issues and challenges encountered in the transition from the modeling/simulation level to the real-life application of the algorithm. We also suggest some modifications that will make feasible the implementation of the algorithm on the robots' limited computing systems.**

**The results show that the suggested modified algorithm, when implemented on the robotic swarm, enables them to successfully identify the shortest path between two points. These results open the door to a wide variety of applications like search & rescue, path planning, and mass transportation.**

***Keywords — swarm robots, ant colony optimization, multi-core computing.***

## I. Introduction

The concept of using a swarm of small robots to perform tasks that are either impossible or too hard for a single robot has been discussed for the last few decades. The various ways a swarm of robots can be organized is discussed in [1]. A wide variety of algorithms have been investigated to control the operation of these swarms. Some of these algorithms, are bio-inspired and try to mimic the evolutionary behavior of living colonies [2]. While many swarm algorithms have been presented and tested on computing/simulation tools, only a few of them have been actually implemented on real-life systems [3].

The focus of this work, the Ant Colony Optimization (ACO) algorithm [4] is one of the prominent algorithms in the fields of computer science and operations research. The idea is to find the optimal path in a graph based on the behavior of food seeking ants. The algorithm is used in a wide variety of applications including solving the famous traveling salesman problem [5], data mining [6], industrial assembly lines [7], and VLSI circuit design [8]. These applications, use the ACO algorithm as a metaheuristic tool to find solutions to their various problems. In these problems, a physical implementation of the ACO is never necessary because the pursued solutions can be found by running a simple computer simulation of the algorithm.

In this work, we investigate the real-life applicability of the algorithm on a small swarm of robots. The importance of this real-life implementation is that it, not only finds the optimal mathematical solution, but also allows the use of the robots to solve a wide variety of real-life problems. One of these problems is using the robots to optimally transport a large number of objects from point A to point B. Another application is to have the robots evacuate a disaster zone by finding the shortest path to safety.

While the ACO algorithm has been extensively studied, the real-life robotic implementation introduces a number of practical challenges and issues that are not encountered, and therefore not addressed at the mathematics/simulation level. Some of these issues include robotic collision avoidance, the error between the actual and the estimated positions, and most importantly, making decentralized decisions on the robots' limited computing resources compared to the powerful CAD and simulation tools. This paper presents a modified version of the ACO that was implemented to simplify the decision making and reduce the computing workload without compromising the performance of the algorithm. The paper also discusses the measures utilized to overcome the other challenges still without compromising the performance of the algorithm.

The rest of the paper is organized as follows: section II briefly discusses the original ACO algorithm while section III describe our modified version. Section IV describe the robots and the experimental setup used to test the modified algorithm. Section V presents the obtained results and the paper is concluded with the conclusion and future works in section VI.

## II. The Ant Colony Optimization Algorithm

The ACO algorithm is bio-inspired by the behavior of ants exploring several possible paths between a source and a destination. Each of these ants will start by traveling on one of these paths. The ants have no global awareness of the different paths and have no means to measure the length of the traveled

paths. The optimization is achieved based on ants depositing pheromone (a chemical they produce) on the traveled paths. As time passes, the shorter paths will have more completed round trips than the longer ones and will therefore have higher concentrations of pheromone. Returning ants, looking to start a new trip, lean towards selecting paths with higher pheromone concentrations leading up to convergence in finding the shortest path.

Several variations of the ACO algorithm exist in the literature. The common theme amongst all versions is that the probability of an ant selecting a certain path is proportional to the ratio of the pheromone concentration on that path to the concentrations on the other paths. In a generic implementation of the algorithm, if *N* ants are exploring *K* possible paths, the *N* ants start the process by taking arbitrary paths in their first runs out. In successive iterations, the chance of an ant going on a path *k* is

$$P_k = \frac{(\rho_k + c)^h}{\sum_{i=1}^{K} (\rho_i + c)^h}. \quad (1)$$

Where $\rho_x$ denotes the pheromone concentration on the path *x*. The variables *c* and *h* are simulation parameters and can change depending on the nature of the application.

The amount of pheromone on each path is a time dependent variable. It increases with additional ants traveling that path and decreases with time because of the evaporation phenomenon. The pheromone concentration at path *k* is updated every simulation cycle using the equation

$$\tau_k(n+1) = (1-e) \cdot \tau_k(n) + \Delta\tau. \quad (2)$$

With $\tau_k$ being the pheromone concentration on path *k, e* being the evaporation rate, and $\Delta\tau$ being the amount of pheromone deposited by the ants on that path during a simulation cycle.

## III. The Modified Algorithm

Just like the original ACO algorithm, the modified algorithm used in this work utilized probability measures in order to mimic the ant behavior. The main difference is that whereas an ant tracks their path by secreting pheromones, the robots are programmed to log the completion of a path with a designated base station. As a robot travels an "efficient" shorter path more quickly, the frequency of travel on such a paths is expected to be higher than the longer ones. As a result, a robot that is starting a new trip, is more likely to take a more efficient path at a higher rate and, in turn, logs more data points for that path.

Whenever a robot is starting a new trip, a decision is needed to determine which of the possible paths the robot must take. The base station has the data from the previous trips of the whole swarm. This data is used to calculate the probability of the robot taking each of the possible K paths using a modified version of equation (1). In this version, the number of completed trips on a path *i* is considered to be an indication of the pheromone level of that path $\rho_i$. In other words, the modified algorithm assumes zero pheromone evaporation and that all the robots/ants have equal weights in the decision making. As a result, the values of the pheromone levels on all paths are always positive integers. The value of the simulation parameter *c* was originally chosen to be zero, meaning that the pheromone level is the exclusive factor to determine the probability of taking a path. In order to ensure that no path has a zero probability of being taken, the process is started with the first *K* robots taking all *K* paths, one robot for each path. This guarantees that, upon the return of all robots from their initial trips, no path has a zero pheromone level. Other values for the parameter *c* were also investigated, and are discussed in the results section.

When it comes to the other simulation parameter *h*, multiple values were used to investigate its effect on the performance. In the basic model (*h* = 1), each data point linearly contributes to the overall chance of that path being taken. In this model, unless a path is significantly shorter than the others, it will take a very long time and a very large number of runs for a path to emerge with a dominant probability. For example, Table I shows a sample of 3 paths pheromone distribution and the corresponding path probabilities for various models. In this sample, Path 2 is the shortest path and adequately has five completed trips while only three and two trips have been completed on path 1 and path 3, respectively. The numbers in the table show that a robot starting a new trip has a 50% chance of taking the shortest path. These probabilities will theoretically lead to the emergence of the shortest path when a large enough sample is used. However, due to the limited number of robots in our small scale implementation, there is a significant risk of a longer path being selected. Also, even if the algorithm converged to select the shortest path, it is expected that a very long time is needed for that convergence.

Considering the size constraint, and considering the numbers in Table I, larger values of *h* seemed to be necessary to increase the chance of the shortest path gaining dominance in a reasonable amount of time. However, while increasing the value of *h* would increase the dominance of the shortest path (as shown in Table I) and consequently the speed of the algorithm, it would also increase the chance of error because the algorithm may be favoring the wrong path.

TABLE I. A SAMPLE PHEROMONE DISTRIBUTION AND THE CORRESPONDING PATH PROBABILITIES FOR VARIOUS MODELS

| | Path 1 | Path 2 | Path 3 |
|---|---|---|---|
| Pheromone Level ρ | 3 | 5 | 2 |
| Basic Model (*h* = 1) | 30% | 50% | 20% |
| Quadratic Model (*h* = 2) | 23.9% | 65.8% | 10.5% |
| Cubic Model (*h* = 3) | 16.9% | 78.1% | 5% |

To understand this issue, it is important to remember that in a real-life implementation like the one in this work, the length of the path is not the only parameter affecting the round trip time. Other real-life inconvenient issues may lead a robot finishing a path in an exceptionally short or long period of time. For example, a robot on a certain path could have taken more time to finish the trip because it slowed down to avoid a collision. Therefore, while the algorithm is supposed to dominantly favor paths with higher pheromone levels, it should also be robust enough to filter out and recover from skewed data.

The numbers in Table I, supported by empirical testing showed that $h = 2$ appeared to be the right design decision as it did lead to a faster convergence of a dominant path without compromising the robustness of the algorithm. Further increase in the exponential value of $h$ would increase the speed of convergence of the algorithm at the expense of introducing higher chance of error. The algorithm could also be manipulated in other ways to fine tune the performance based on the needs of the designers and the specific criteria of their applications. One possibility is introducing data decay. This would mimic the pheromone evaporation process in the original algorithm that is modeled in equation (2). This decay makes a longer path, not only unpopular, but also forgotten. In an ideal case, this would lead to only one path eventually having all of the data points and all other paths being forgotten. The addition of data decay can have a significant effect on a small-scale application like the one in this work. However, in a large-scale implementation, with many more robots, a change like this would become negligible, as paths that are less popular will gradually lose their statistical effectiveness anyways.

## IV. The Experimental Setup

### A. The Testing Environmnet

To implement the algorithm in a small-scale real-life application, an 8'x4' wooden arena was fabricated to serve as the testing environment for the robots. The arena can be seen in Figure 1. In this arena, the testing environment was setup by designating an origin (the nest) and a destination (the food source). Three possible paths of different lengths can lead a robot from the origin to the destination. The map of the arena can be seen in Figure 2. The red and the green colored areas indicate the origin and the destination, respectively. The three possible paths are also indicated by three different line styles as specified in the figure. Each path includes two non-overlapping lanes for two directions of travel (origin→destination and destination→origin). The two lanes per path system was used to minimize robot collisions. The direction on each lane is indicated by an arrow in figure 2.

### B. The Robots

For the robots, the starting point was the Propeller Activity Bot [9] from Parallax, Inc. This robot was chosen because it is easy to program in C with the *SimpleIDE* software. It also includes high-speed servo motors with optical encoders to provide accurate and consistent maneuvering. The robots also include mounted breadboards allowing the necessary additions of radio modules and ultrasonic proximity sensors that are discussed below in further detail.

### C. The Communication System

Since the robots can neither deposit chemicals nor sense their concentrations, the chemical pheromone was replaced by digital pheromone or electromagnetic signals [10]. In digital pheromone, rather than laying down the chemical on the traveled path, each robot communicated via airwaves to a centralized home base. The home base recorded the number of times each path had been completed. It was critical that this home base was stationary and located near the origin because in a scaled up application, with a larger number of robots, it may not be possible to establish communication between all the robots. The home base was situated near the origin because that would be where a robot needs to report a completed path and acquire the updated data for the next trip.

Another benefit of the home base was using it for traffic control at the congestion area near the origin. All the robots came back to the origin and then had the choice to either go back on the same path or transition to another one. This increased the chance of robots running into each other on this congested part of the course. To solve this issue, a First In First Out (FIFO) queue was set up where the home base stored the requests from various robots and only provided the next robot in the queue with a new path to execute when the traffic section was clear.

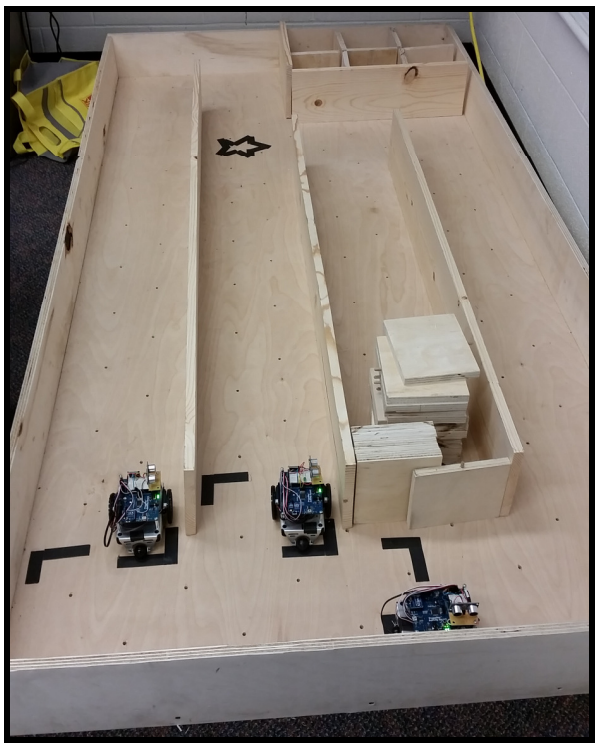

Figure 1. The testing environment, with three Propeller Activity Bots with peripherals at their starting locations.

The robots communicated with the home base through the XBee-PRO Module 802.15.4, which were mounted on every robot including the stationary home base. One of the nice features of this module is that it allows the robot to communicate selectively. The XBee modules can be set such that the mobile robots can only communicate with the home base. Similarly, the home base can selectively choose to send exclusive data to a specific robot.

In the implementation, once a robot completed a path, it would send a signal to the home base notifying it of the robot's unique identity and the path that it just completed. In response, the home base would reply to that robot with the robot's unique identity followed by the path on which to take. It would also tell the robot whether to wait or proceed, depending on whether or not the home base area was clear. When cleared, the robot would transition to the new path to be taken, and once on the new path, it would notify the home base, which in turn proceeded to communicate with the next robot waiting in the queue.

### *D. The Proximity Sensors*

Another issue that had to be addressed was making sure all the robots stayed on course during their trips. These robots were programmed to know their way on pre-determined paths. Once a path is chosen, a robot should ideally be able to find its way from the origin to the destination and back. The problem is that the robots, even with the position encoders, did not accurately stay in their lane in their chosen path.

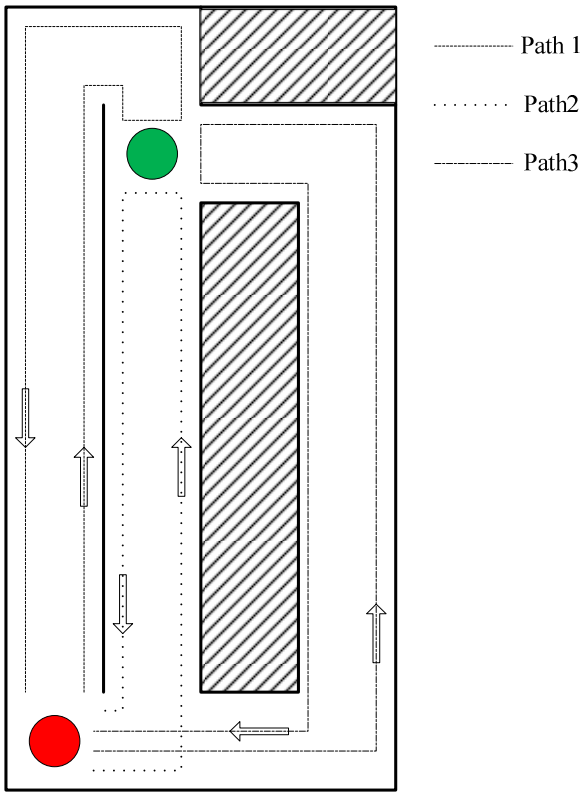


Figure 2. Map of the arena showing the origin, the destination, and the three different paths.

This was a big obstacle to implementing the algorithm, especially because the robots were not driving in a straight line, and inconsistency is expected when a robot makes multiple turns in a trip. In a large-scale application, this issue could be addressed using geolocation techniques such as adding a Global Positioning System (GPS) on each robot. This solution was unfortunately not feasible in a small scale indoor environment like the one in this work. Therefore, another solution was needed to address these inconsistencies and prevent the robots from veering off course and even driving into each other in some cases.

This problem was solved by adding an ultrasonic sensor to the right side of each mobile robot. The ultrasonic sensor, interfaced to the robot's microcontroller would send an ultrasonic burst and receive the echo returned to determine the distance to a nearby object. The robot would use that data to determine if it is too far or too close to the wall and correct itself to maintain a pre-determined distance to the wall. Multiple robots with sensors mounted on the right side can be seen in figure 3. The figure shows, from a robot perspective, two robots on opposing lanes of path 1. The addition of the ultrasonic sensors guaranteed each robot to drive alongside the wall, which helped overcome position inaccuracies.

### *E. The Microcontroller and the Code*

The Propeller microcontroller allows the use of multi-threaded code, as the Propeller has 8 logical cores with shared memory. This turned out to be necessary as described below.

The application of a robot swarm working in cohesion required precise motion control to avoid collisions on the path. Proper motor speed and accurate orientation information were mandatory to control each robot with a high degree of precision; this was done best with hardware timers. Each core of the Propeller has two timer/counter peripherals. The robots used in this work, however, required precise timing of three components: two servo-motors, where each servo-motor needs an independently timed PWM signal, and an ultrasonic sensor, that also requires an accurate dedicated timer to perform the crucial distance measurement.

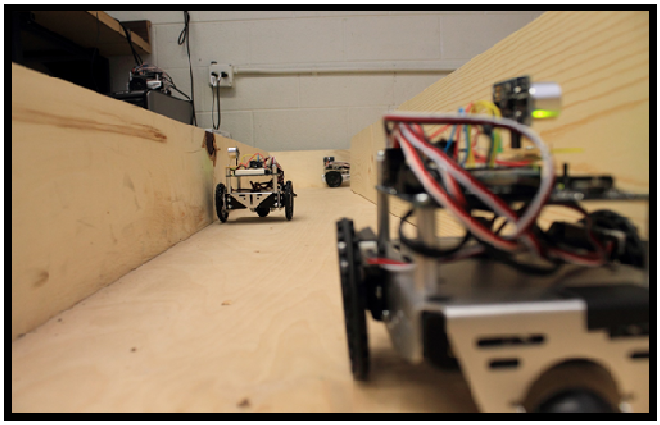

Figure 3. A robot-level action shot of three Robots with the peripherals in the testing environment.

In order to accurately time the three components with separate hardware timers, the code was broken into a dual-threaded program to take advantage of additional hardware timers available on other cores of the Propeller microcontroller. This dual-threaded code used a main program core and a speed adjustment core. The main program controlled the two motors and the communication module. The main program also called a function to start on a new core within the microcontroller. This function set up the ultrasonic timer and used one of the timer/counter peripherals on the second core in use to time each ping, thus measuring distance. This function then calculated drive corrections to be made, and loaded new motor speed values into memory. The main program core accessed that same memory when driving the two motors, each controlled by a timer peripheral on the main core. This design gave accurate measurements from the ultrasonic sensor while maintaining fluid speed control of the two motors.

## V. Results

When implementing the modified ACO in real life, the main constraint was the size of the swarm. Due to the limited dimensions of the available testing environment, only three robots were running simultaneously. Nevertheless, even with the smaller number of robots, a proof of concept was accomplished and the results showed that, in the majority of the test runs, the modified algorithm converged and the robots were successfully able to identify the shortest available path.

The implementation of the work in a physical system, also showed that there are many important considerations unexplored in simulations that greatly impact the behavior of the algorithm. The first, perhaps more plain, implication of a physical implementation is the possibility of collisions between robots, particularly concerning congestion around the home base. It was found that using the queue system that makes robots wait for traffic to clear adds a fairly consistent amount of time to each path traveled. This addition across the board decreases the relative dominance of a shorter path, leading to a slower convergence. For example, if a shorter path can be traveled in 10 seconds while a longer path is traveled in 20 seconds. A round trip on the longer path is twice as long as a similar trip on the shorter path. Adding a 2 second queue time for both paths lead to changing the proportions and make the longer path only 1.83 times longer than the shorter one. The longer the queue times, the slower the convergence of the algorithm.

Another important factor that was noticed in the physical implementation is the speed of the individual robots. While these robots are identical, and must ideally run at the same speed, real-life imperfections must be taken into consideration, especially since the whole algorithm depends on travel times. Fortunately this was addressed by choosing suitable values for the parameters $c$ and $h$ in equation (1). As explained in section III, setting the value of $h$ =2 resulted in significant increase in the speed of convergence. In fact, convergence often happened without ever actually making a 'wrong' decision. In other words, immediately after the initial run, where each of the three robots travels one of the three paths, all three robots will indefinitely converge to the shortest path. This could be considered good news. However, we felt that such a fast convergence was associated with a high chance of error, such as some cases where the robots actually converged on a wrong path. To remedy this, the value of the parameter $c$ was increased to one. In the updated model ($h$=2, $c$=1), the robots took at least two trips exploring possible paths before eventually converging to the shortest path.

It is worth noting that the severity of the tradeoff between the two models above is heightened by our small sample size. The difference between various models, while existing, should be much less significant with larger robot populations.

## VI. Conclusion & Future Works

In this work, we presented a small-scale implementation of the well-known ant colony optimization algorithm on a small swarm of robots. The algorithm needed to be modified to overcome a number of physical implementation issues that are otherwise not encountered in a computer simulation.

The proof of concept implementation was successfully accomplished. The results showed that the implementation parameters can be adjusted in a tradeoff between the speed of convergence and the robustness of the algorithm. Based on their application, a system designer must adjust those parameters based on how efficient does the system need to be in order to be considered viable, and how much error are they willing to tolerate for an efficient convergence.

In the future, one important development for this work is to move beyond pre-defined paths by equipping the robots with the means to look for their prospective targets. One of the possibilities is to add a thermal sensor and include one or more heat sources in the testing environment.

Moreover, for an effective rescue mission, it is important to be able to adapt to never before seen terrains. For this purpose, a combination of the ACO with Dijkstra's algorithm might prove useful. One possible implementation could have the robots create their own node mesh through exploration, creating several possible paths to an end goal. Dijkstra's algorithm would not be able to solve the entire problem, as not all edges are found, nor their lengths or weights. The ACO algorithm would be useful to find optimal paths within this mesh, using frequency analysis rather than strict distances.

## Acknowledgment

This work was possible thanks to the generous support received via the Frederick F. Jenny Professorship of emerging technologies.

## REFERENCES


[1] G. Dudek, M. Jenkin, E. Milios, & D. Wilkes, "A taxonomy for swarm robots." *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1993.

[2] S. Nouyan, R. Groß, M. Bonani, F. Mondada, & M. Dorigo, "Teamwork in self-organized robot colonies." *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, 2009.

[3] J. Lindsay, S. Givigi, G. Pieris, G. Fusina, & G. Labonte, "Experimental validation of swarms of robots." *IEEE International Conference on Systems.* 2012.

[4] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization by a colony of cooperating agents." *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, vol. 26, no. 1, 1996.

[5] M. Dorigo and L. Gambardella. "Ant Colony System: A cooperative learning approach to the traveling salesman problem." *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997.

[6] R. Parpinelli, H. Lopes, and A. Freitas. "Data mining with an ant colony optimization algorithm." *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, 2002.

[7] J. Bautista and J. Pereira. "Ant algorithms for a time and space constrained assembly line balancing problem." *European Journal of Operational Research*, vol. 177, no. 3, 2007.

[8] S. Alupoaei and S. Katkoori. "Ant colony system application to marcocell overlap removal." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, 2004.

[9] https://www.parallax.com/product/32500

[10] H. Parunak, S. Brueckner, J. Sauter, "Digital pheromone mechanisms for coordination of unmanned vehicles." *International joint conference on Autonomous agents and multiagent systems: part 1,* 2002.