DYNAMIC OPTIMIZATION OF A HETEROGENEOUS SWARM OF ROBOTS

Miles C. D. Pekala
Department of Electrical Engineering
Lehigh University
Bethlehem, PA
mcp7@lehigh.edu

Eugenio Schuster
Department of Mechanical Engineering
Lehigh University
Bethlehem, PA
schuster@lehigh.edu

ABSTRACT
The control mechanisms of swarms of cooperating robots have been studied for some time now, but usually assuming homogeneous swarms. This paper presents a control algorithm for a heterogeneous swarm of cooperating robots based on a modification of the ant colony optimization (ACO) method. We consider the possibility of a partially disabled robot, we analyze the effect it would have on a heterogeneous swarm of robots, and we discuss how to work around the disability. We examine the effectivness of this control algorithm through a simulation study. For this purpose, a biologically inspired strategy is used for the movement of the robots.

KEY WORDS
Heterogeneous Swarm Optimization, Ant Colony Optimization

1 Introduction

There has been a great deal of research recently with regard to the application of emergent intelligence in swarms of robots. Many different researchers have illustrated cooperation between different individuals in a swarm. For instance, researchers managed to build a team of similar robots that surmount holes when encountering them [1, 2]. They accomplished this by letting the robots form chains when encountering a hole that was too wide for a single robot to traverse. Researchers at the University of Alberta made robots clear a platform for possible other construction [3]. Other accomplishments include getting a robot to autonomously choose a construction site [4] and many others.

Most cooperating swarms have been homogeneous with regard to their component robots, i.e., any particular robot can take on any particular task. A significant problem arises when one wants to introduce heterogeneous robots into the swarm; some robots may not be physically capable of doing all the jobs a task requires. For example if a job requires a flying robot, a robot unable to fly would be physically incapable of completing the job. A similar problem occurs when considering a robot that becomes disabled during the course of a task. The main goal of this paper is to develop a method or algorithm that can control a heterogeneous swarm in an optimal manner to accomplish any particular task without human intervention. To accomplish this, we propose a modified version of a popular aggregation strategy, the Ant Colony Optimization (ACO) method, set forth in [5] by Dorigo and coworkers.

The ACO method was developed by considering the movement of ants. If there are two paths between two points, say a piece of food and a home, the ants pick the shorter one. This is due to the fact that ants reach the goal and return on the shorter path before they do on the longer path, resulting in a greater amount of pheromones on the shorter path. Ants choose which path to traverse by the amount of pheromone on each path, and thus, they are more likely to take the shorter, more-pheromone-laden path.

A number of papers ([6], [7], [8], [9], [10], and [11]) deal with this sort of optimization. Two of them ([8] and [9]) deal with a straightforward application of the ACO method in the Traveling Salesman Problem (TSP). Due to the effectiveness of this method in solving TSPs, ACO is used extensively in telecommunications for routing. In this paper, a modified version of the ACO method is used to solve a TSP where the nodes are robot jobs instead of cities. The modified (or dynamic) ACO method proposed in this work allows the drop and addition of nodes, an incomplete circuit, and the modification of job efficiencies. This allow us to optimize each robot in our swarm to work through the task in the most efficient order, and to effectively drop any job the robot is incapable of doing.

2 Dynamic ACO

The dynamic ACO method used in this paper is a modification of the classic ant system, described in [5], to account for dynamic alteration. A dynamic ACO can be best explained as an ACO implementation in which either the nodes or path length between the nodes vary during the operation of the ACO. Nodes may either be deleted or come into existence during the operation of the ACO, and path lengths may increase or decrease depending on external factors beyond the control of the ACO system. The question becomes how does one optimize the ACO system when such things are occurring?

Already some research has been done on dynamic ACO algorithms with some notable work done by Merkle [12] and Guntsch [13, 14]. These researchers explored the possibility of a dynamic ACO system that solves

a TSP that has nodes removed or added during the ACO process, developing several strategies to deal with such a system. Others such as Montemanni [15] explored a dynamic vehicle routing problem in which the new orders for a delivery service are received and must be incorporated optimally in each vehicles route. Although this prior work is not specifically relevant to our particular problem, it provides us with a good starting point in developing a dynamic strategy for job selection.

3 A Modifed Dynamic ACO Algorithm for Job Selection

Since our optimization algorithms attempt to fit any robot with a particular job, we shall divide the task in to a set of sub-tasks (called "jobs" in this paper) which, when completed, make up the entire task. The different jobs are represented as nodes in the ACO algorithm.

To design the Modified Dynamic ACO (MDA) algorithm, we first define the "distance" between job nodes as the quality of the job done by the robot. In the case that we consider here, the quality is defined as the time it takes for the robot to finish each job. Upon completion of each job, the quality is incorporated into a quality matrix. The quality matrix is a square matrix of order equal to the number of jobs. Each column represents the current job and each row represents the next job. Each position in each column represents the probability for the robot to switch to the job represented by the associated row. After the completion of each job, the job quality is averaged with the quality in the row corresponding to the current job and in the column corresponding to the previous job. In doing this, the MDA method judges itself on the quality of the job it does after transitioning from a previous job. A quality matrix can be defined for each agent as

$$Q = \begin{bmatrix} q_{1,1} & \cdots & q_{1,n} \\ \vdots & \ddots & \vdots \\ q_{n,1} & \cdots & q_{n,n} \end{bmatrix}, \qquad (1)$$

where n is the number of jobs. Each entry in this matrix represents the quality of each job the robot completes when transitioning from another job. Since this matrix essentially represents the edge lengths between nodes, this is essentially a TSP. Running the ACO on this TSP produces the quickest cycle though these jobs.

We run the ACO through the values in the transition matrix as the edge lengths between nodes. Once the ACO is completed, a job cycle can be generated by simply letting a single ant walk through the weighted paths of this TSP. The ant keeps track of which nodes it has visited, and simply walks through with the same probabilistic functions as the ants who generated the weighted paths. Once the ant has visited all the nodes, the sequence of nodes it has visited is what we call a job cycle.

An issue arises when an robot cannot complete a job due to either injury or design. What is it to do then? The job remains in the robot's undone job queue, and the robot keeps attempting to do it until the job is completed. Since the robot cannot do the job, it will wait in it states in this situation forever. To fix this, a job timeout can be added, where the agent attempt to do a job for a certain amount of time and if the job remains unfinished when the timeout occurs, the robot switches to another job. However, the problem remains that eventually the robot returns to the job that it cannot complete. If the robot cannot physically complete the job, then it should not try to do the job in the future. Allowing the ACO to do an incomplete circuit (a "loop" or "looping" as refered to in this paper) enables the robot to do another job without completing an entire cycle. Another problem arises here if the problem is time based, i.e., something has not occurred so the robot cannot complete the job at the given time but may at some point in the future. This eventually resolves itself as the robot completes all possible jobs and it is only left with the job it previously could not complete. The likelihood of it switching to the job it could not do previously would then increase due to the inability to do any other job.

4 Simulator Design

A simulator was programmed to test the MDA. Using this simulator, the MDA can be compared against an unoptimized approach based on a basic sequential job selection, and data can be gathered without the need for a large amount of expenditure on robots. Each robot of the simulator should accurately represent a robot in real life, properly simulating all of its attributes. The robots should also be implemented in such a way that a large amount of different robots can be achieved with relative ease.

4.1 Tasks and Jobs

To test the MDA we require a task, which is defined in this case as a simple food-gathering task. The overall goal of the system is to gather food at the home base (or "kitchen" as it is referred to in this paper) from any food source in the surrounding area. The task is further complicated by requiring that the food brought to the kitchen be stacked (piled) on one another. Therefore, we have three jobs:

1. Scouting: Agents with the "Scouting" job will wander around looking for a source of food. When food is found, the agents will return to the kitchen with the glorious news.
2. Transporting: Agents with the "Transportation" job will look for the source of food that the scouts found, grab a piece of food, and take it to the kitchen.
3. Stacking: Agents with the "Stacking" job will take a piece of food in the kitchen and place it on another piece of food.

These three tasks then have the following job transition conditions:

1. When "Scouting" is finished and the agent has returned to the kitchen, switch to another job.

2. When "Transportation" is finished (i.e., a piece of food is brought to the kitchen), switch to another job.

3. When another piece of food is found in the kitchen, and the agent is currently doing the "Stacking" job, stack the food in the cargo and switch to another job.

If the agent is "Stacking" and the job timer expires, drop the food in the cargo and switch to another job.

The basic route through this task consists on each job being done in turn (i.e., $1, 2, 3, 1, 2, 3, \ldots$). This is referred to as basic sequential (BS) job selection in the rest of the paper, and it is used to be compared with the MDA algorithm proposed in this work.

Along with the robot, the food attributes must also be defined with regard to the simulator. Although significantly simpler than the robot design, a piece of food should have similar properties, that is, a physical representation in the field and a location within the arena. For this simulator, a field is required for the robots to act upon. The field is a simple arena consisting of walls, obstacles, and a kitchen. The field does not specify food and robots, nor does any other possible dynamic object. Rather, the field is simply a static representation of the environment.

4.2 A Pheromone Based Movement Strategy

The movement strategy is biological inspired by the ant behavior, i.e., it is based on the pheromone concept explained above. This movement algorithm produces a faster task completion when compared with a simple collision avoidance algorithm. This is because the algorithm directs the robots towards the food more often than not. As such, this movement strategy is used in conjunction with the MDA. The movement algorithm works without knowledge of the surrounding area, it requires little pre-processing, and it is a dynamic method.

For this pheromone inspired movement strategy, a normal ACO cannot be used as it is not a general enough implementation. While the normal ACO would be excellent if we were attempting to find the fastest method to solve a TSP, it is not very good when we try to implement movement. Each pixel would have to be considered a node (assuming we do not use some sort of map division strategy) and the map would have to be known. However, using the fact that our robots are meant to more probably follow a pheromone than not, we can generate a movement strategy.

First, let us define the pheromone the robot lays down. Each robot lays down a pheromone trail of width W and of strength S. The width is in pixels since we are working in a digital environment. The strength S is a number that represents the amount of pheromone that is laid down. For the simulator, each robot lays down this trail on a pheromone map that is of the same height and width as our field. As the robots run around the field, they lay pheromone down behind them. After each "tick" (.1666th of a second) of the simulator, the pheromone currently on the field evaporates with a predefined evaporation rate. The robot sees the pheromone in its proximity and more probably generates a heading towards the area with a larger amount of pheromone. We use a truncated normal distribution to choose a direction based on the pheromone levels.

Next, let us define how each robot moves on the board. Each robot has a probabilistic weight associated with its movement based on the amount of pheromone it sees. It is more probable to move in the direction of the pheromone than not. We must also include a weight to prevent the robot from crashing into some obstacle in the field, that is, a weight dependant on a non-collision strategy. For our purposes, a basic non-collision movement strategy can be used. For each robot with an array of distance sensors, a weighted combination of the angle of the sensor, with respect to the normal direction of the sensor, and its distance is generated. From this weighted value an overall heading can be obtained. This strategy is simple and serves our purpose well while not interfering with the robot moving on the pheromone trails.

The pheromone-based movement (PBM) algorithm combines the heading from the sensors with the heading generated by the pheromone. This combination provides a heading that has a good probablity to follow the pheromone avoiding collisions. When applied, this movement algorithm generates very good pheromone following robots.

It is also possible to generate a simpler movement strategy by just using the weighted sensor information and disregarding the pheromone data. This basic movement (BM) algorithm is also considered for the simulation cases presented below.

4.3 A simulation run

In figure 1 we can see a typical simulation run. Five robots are working together to find 15 pieces of food, bring them back, and stack them in a designated kitchen area. Each robot is represented by a colored rectangle with a point on the end (different colors indicate different jobs being executed by the robots), while the black polygons are simple obstacles, and the yellow circles are pieces of food. This configuration of 5 robots and 15 pieces of food are used in every simulation run presented below. The lines being drawn from each robot are the pheromone trails for the movement strategy.

Figure 2 shows the consequence of an inappropriate pheromone evaporation rate. The robots create pheromone "black holes" during the simulation of the pheromone-based movement strategy. The robots initally move in a circle because of some reason (usually object avoidance) and then get stuck. This is because the pheromones do not evaporate quickly enough, and they reinforce each time the robots go around. This can be fixed by adjusting the evaporation constant, and the probability for the robots to follow a pheromone.

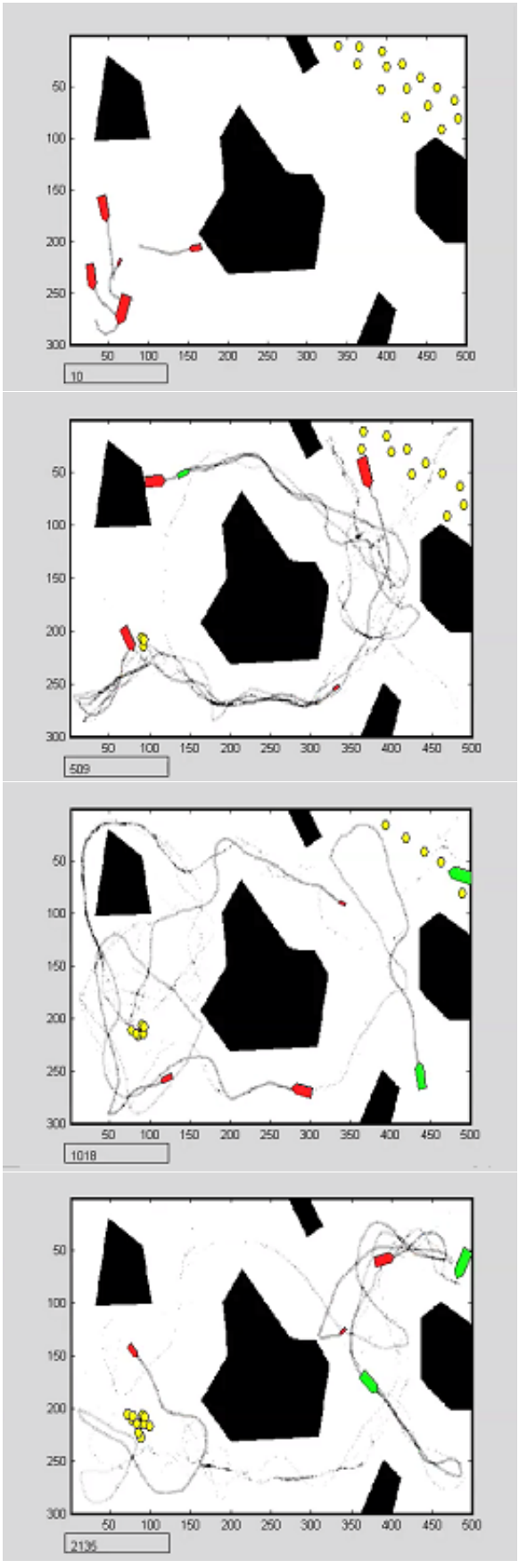

Figure 1. A typical simulation run of one set of robots

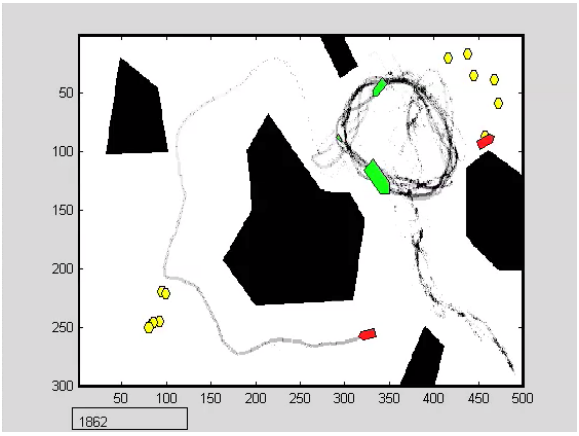

Figure 2. A circle being generated by the pheromone movement strategy

5 Discussion of results

The simulator was run several times with several different configurations to ensure the validity of our results. Figures 3 and 4 show some of the runs (denoted as "series" in the figures), illustrating at what time (vertical axis) each piece of food (horizontal axis) is stacked in the kitchen, and therefore the progress of the task. Figure 3 illustrates the Modified Dynamic ACO (MDA) job selection algorithm, while figure 4 illustrates the simpler basic sequential (BS) job selection algorithm. Both cases are simulated using the pheromone based movement (PBM) strategy. Since a total of 15 pieces of food are laid out, when 15 pieces of food are stacked, the simulation is considered to be completed. The disparity in the two different runs in figure 3 are due to the randomness associated with the MDA algorithm, in particular at the beginning of the task (see below).

Figure 5 compares the food delivery times for different algorithms: MDA job selection algorithm with pheromone-based movement strategy (MDA/PBM), basic sequential job selection algorithm with pheromone-based movement strategy (BS/PBM), and basic sequential job selection algorithm with a basic movement strategy that disregards pheromone information (BS/BM). In the beginning of the simulation, the MDA shows a lack of optimization. This is because the MDA has yet to adjust itself to the task, and all jobs are equally probable. The MDA algorithm starts out with poor effiency due to the fact that the job is randomly selected at the beginning. This creates a period at the beginning of the run where a robot may be in an uncompletable task. In the sequential selection algorithm, the first job is completable at the beginning of the run, and the first piece of food can be grabbed faster.

In this simulation example, the sequential job cycle turns to be the most efficient way to complete the task. It is possible to note from figures 5 and 6 that the MDA algorithm eventually optimizes itself to be sequential. Figure 5

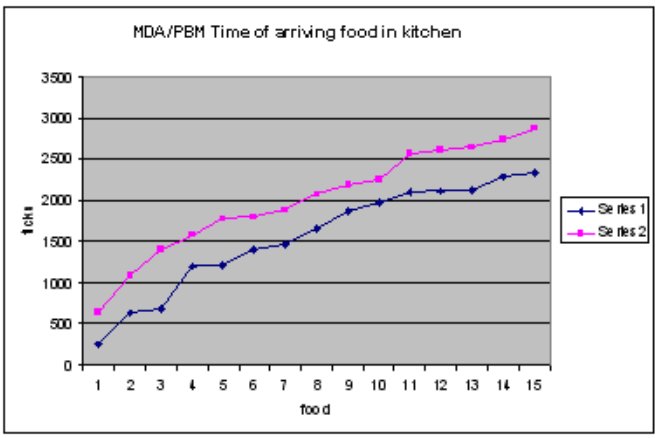

Figure 3. Food delivery times for a Modified Dynamic ACO (MDA) job selection algorithm with a Pheromone Based Movement (PBM) strategy

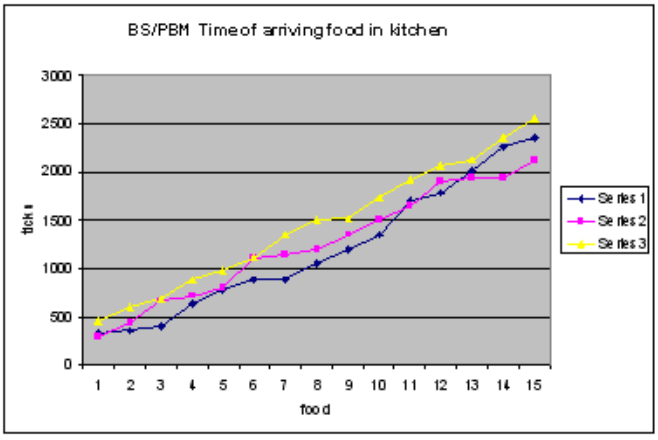

Figure 4. Food delivery times for a basic sequential (BS) job selection algorithm with a Pheromone Based Movement (PBM) strategy

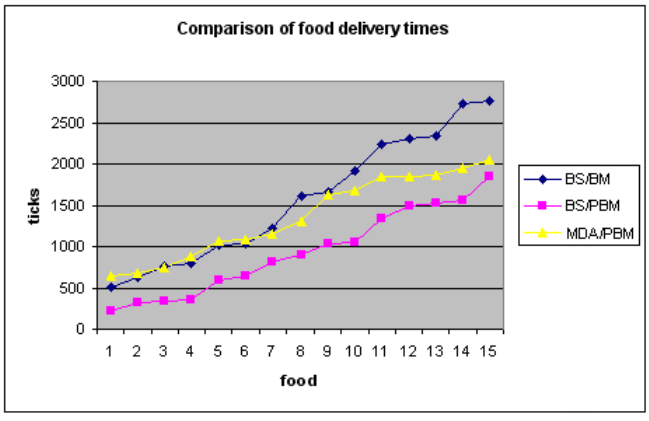

Figure 5. Comparison of food delivery times for different job selection methods

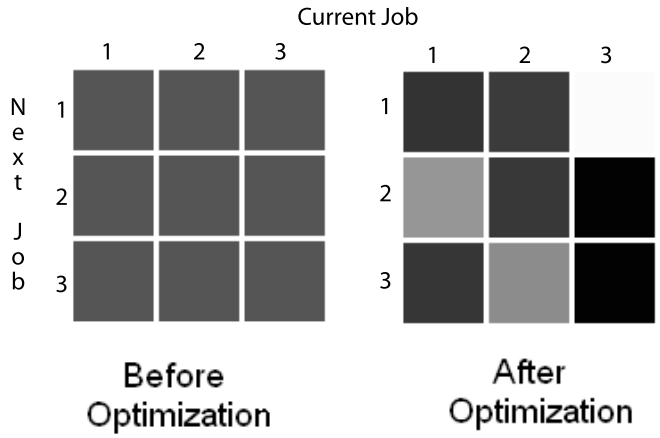

Figure 6. Optimized and unoptimized transition (quality) matrices

shows that the necessary time to complete the task (i.e., to gather the 15 pieces of food) for the MDA/PBM algorithm is comparable with that for the BS/PBM algorithm. Figure 6 shows the unoptimized and optimized job transition matrices, which evolve during the run of the simulator. The black color represents a probability of 1, a white color represents a probability 0, and shades of gray are intermediate probabilities. The colors in the optimized job transition matrix indicate a sequential job selection.

Since a pure sequential job selection appears to be the optimal route through the task in this simulation example, the loss of time in the beginning of the optimization causes most of the MDA job selection simulations to be slightly longer than the sequential job selection counterparts. This can be seen in figure 7, where the task completion times for different methods are compared in several simulation runs. Besides the MDA/PBM, BS/PBM, and BS/BM algorithms described above, a reverse sequential job selection algorithm with a basic movement strategy that disregards pheromone information (RS/BM) is also presented. In this case, the jobs are executed sequentially in the 3-2-1 order. It is easy to note that the reverse sequential algorithm is outperformed by the sequential algorithms and the MDA algorithm, which in fact evolves into a sequential algorithm in this simulation example.

In this simulation example, each robot is fully capable of doing every single job. When the simulation is run with some disabled robots, the MDA algorithm absolutely outperforms the sequential job selection algorithms independently of the movement strategy considered for the simulation. This is because the MDA algorithm allows the job selection to loop, and complete the task without having to wait for jobs to time out.

Another interesting thing occurs when looping is allowed in the job selection algorithm. In this case, any redundant job is no longer attempted to be completed. In many of the simulations using looping, the first job, “scouting”, was ignored by the majority of the robots.

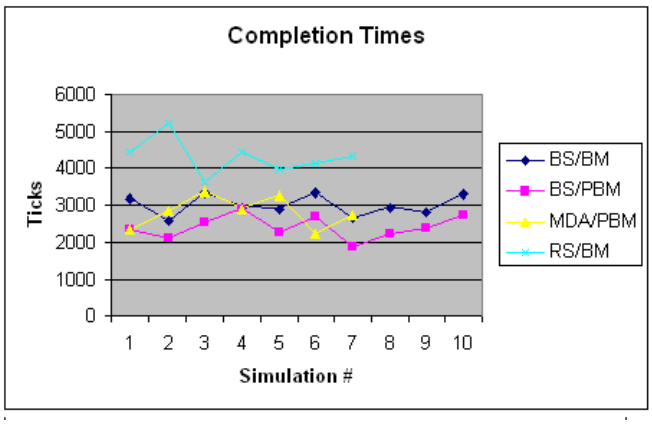

Figure 7. Task completion time comparison

6 Final Thoughts

Upon review of the data, one should see that there is significant promise in the MDA method for the optimization of jobs. The MDA job selection seems to perform poorly at first when compared to the sequential job selection, although this is due to the fact that sequential is indeed the most efficient way to complete the task for the example considered in this work. One can imagine a more complex task in which the most optimal job order is not as readily apparent. Additionally, it has been proved in simulations that the MDA algorithm beats the sequential job selection algorithm when broken or partially disabled robot are considered. This is explained by the fact that each robot is able to drop whatever job it cannot complete (looping), and go on to another job without having to let its job timer expire. This is interesting, since in dealing with a TSP and the ant colony simulation, looping is undesirable, yet here it is very desirable. In the conventional ACO/TSP, looping confines your solution set, while here looping is almost required for completing the task in an optimal way.

References

[1] Nouyan S., Dorigo M., Chain Formation in a Swarm of Robots, *Technical Report TR/IRIDIA/2004-18*, IRIDIA, March 2004.

[2] Trianni V., Nolfi S., Dorigo M., Cooperative Hole Avoidance in a Swarm-bot , *Robotics and Autonomous Systems*, Feb. 2006, vol. 54, no. 2, pp. 97-103

[3] Parker, Chris A. C., Zhang, Hong and Kube, Ronald C. Blind Bulldozing: Multiple Robot Nest Construction, *In proceedings of IROS2003. (2003)*, Oct. 2003, pp. 2010- 2015

[4] Sahin E., Franks N.R., Simulation of Nest Assessment Behavior by Ant Scouts,*Proceedings of ANTS 2002: From Ants Colonies to Artificial Ants, 3rd International Workshop on Ants Algorithm*, 2002, vol. 2463, pp. 274-282

[5] M. Dorigo, V. Maniezzo, and A. Colorni, The ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics* , Feb 1996, Part B, vol. 26, no. 1, pp. 29-41

[6] H.M. Botee and E. Bonabeau, Evolving ant colony optimization, *Advances in Complex Systems*, 1999, vol. 1, no. 2/3, pp. 149-159

[7] A. Colorni, M. Dorigo, and V. Maniezzo, Distributed optimization by ant colonies, *Proc. First European Conference on Artificial Life* , 1992 , pp. 134-142

[8] L.M. Gambardella and M. Dorigo, Solving symmetric and asymmetric TSPs by ant colonies, *Proc. IEEE Intenational Conference on Evolutionary Computation*, May 20-22 1996, pp. 622-627

[9] M. Dorigo and L.M. Gambardella, Ant colonies for the traveling salesman problem, *BioSystems*, 1997, vol. 43, pp. 73-81

[10] H. Kawamura, M. Yamamoto, K. Suzuki, and A. Ohuchi, Multiple ant colonies algorithm based on colony level interactions, *IEICE Transactions on Fundamentals* , Feb. 2000, vol. E83-A, no. 2, pp.371-379

[11] .D. Taillard, Ant systems, *Technical report IDSIA-05-99* , IDSIA, Lugano, 1999, Handbook of Applied Optimization.

[12] D. Merkle, M. Middendorf, A New Approach to Solve Permutation Scheduling Problems with Ant Colony Optimization,*Applications of Evolutionary Computing : EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, 2001, vol. 2037, pp. 484

[13] M. Guntsch, M. Middendorf, Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP, *Applications of Evolutionary Computing : EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, 2001, vol. 2037, pp. 213

[14] M. Guntsch, M. Middendorf, and H. Schmeck., An Ant Colony Optimization Approach to Dynamic TSP,*In L. S. et al., editor, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001, pages 860-867

[15] R. Montemanni,, L.M. Gambardella, A.E. Rizzoli, A. V. A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System, *Technical Report IDSIA-23-02, IDSIA*, 2002