Coordinating Multi-Robot Systems

Henry T. Megarry
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
mega0022@morris.umn.edu

ABSTRACT

There exists several problems in which a group of robots would be needed to generate a solution within a reasonable time. As the number of robots in a system grows, so does the complexity of coordination. A centralized control system is quickly overwhelmed by the growth in complexity. One reason is that if the central system goes down it affects the rest of system. An additional problem that is that any planning algorithm will have difficulty dealing with the sheer number of robots it needs to account for. However, researchers have found three ways to avoid such issues: the swarm method, the auction method, and the approximation method. The swarm method has roots in studies of social insects such as ants or bees. In these systems, robots' individual behavior is defined, and the collective behavior follows. The auction method is based on decentralized auctions, in which the robots "bid" for tasks based on the costs of getting to the spot and how easily the robot will be able to do the task. The approximation method is the previous two methods combined: each robot looks at the tasks around it as well as the robots around it and decides which tasks it should proceed with based on what it estimates the other robots are doing. Each method has its merits and drawbacks, and the correct method to use changes from problem to problem.

Keywords

Multi-Agent, Robotics

1. INTRODUCTION

Robots are a useful way to automate physical tasks without the input of people. However there are real world problems in which multiple tasks accomplished at the same time, that are distributed, and/or have many parts that need to be coordinated. Such systems are very complex and, as the number of tasks and number of robots increases, become untenable to handle with a centralized control system [2]. There have been several ideas and methods on how to make these systems more robust and scaleable. In this paper we will explore three of these methods, and will compare and contrast them.

Each of the following subsections contain background needed to understand the method followed by a description of the method itself. Hereinafter we discuss the advantages and disadvantages to each method and what methods would succeed in certain problems. We finish with concluding what method is most versatile and solves the problem of scaleability the best.

UMM CSci Senior Seminar Conference, April 2016 Morris, MN.

2. BACKGROUND

Several of the methods that will be discussed require some background knowledge in order for them to be fully understood. The following subsections are an explanation of Markov decision process as well as evolutionary computation. These tools are used for decision making and evolving behaviors for the robots respectively. This section will also include several examples of robotics problems these methods will be solving.

2.1 Types of Robotics Problems

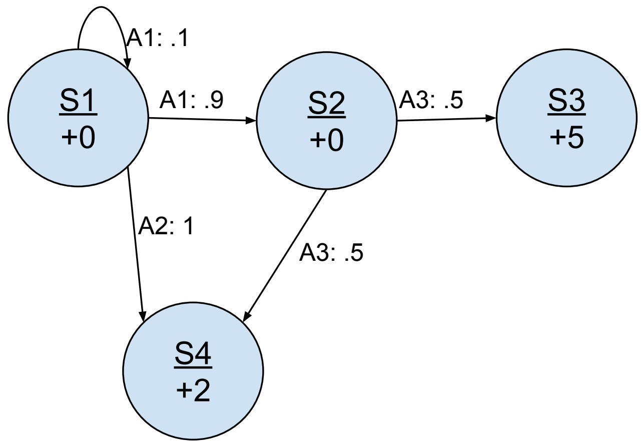
Multi-robot systems have many applications. Some of these include: sensing tasks in machines or the human body with the use of nanobots, disaster rescue missions, mining, foraging, and even interactive art [6]. In this paper we discuss the office cleaning problem in which the robots are in charge of cleaning a cubical type location. Each spot to clean will be seen as a task. The office cleaning problem will be discussed more thoroughly by each method, and will be used to judge the performance of the three methods.

A few categories of problems discussed later on are spatial organization, exploration, and collective decision. Some examples of spatial organization problems are pattern formation, aggregation, and chain formation. Exploration problems include collective exploration, coordinated motion, and collective transport. Collective decision problems are problems such as consensus achievement and task allocation [1].

2.2 Markov Decision Process

Markov decision process (MDP) is a method in solving a decision based problem with some element of randomness [5]. The process can be defined by a tuple of four elements: a set of states, a set of actions, a probability function, and an immediate reward function. The states are what the "world" looks like at that point in time. Actions lead to a new state or possibly the same state. The probability function gives the probability of actions from one state to another. As

Figure 1: a simple Markov decision process problem



illustrated in figure 1, if you start in state S1 and take action A1, there is a 90% chance of going to state S2 and a 10% chance of going to state S4. The immediate reward function gives a positive or negative value to states. For example, in figure 1 the state S3 gives an reward of 5 points, and state S4 has a value of 2 points. The point of the problem is to find the path with the maximum rewards.

To calculate the optimal solution or policy you first need to calculate the utility of the state. You can find the utility by adding the reward of the state that you are in with the utility of the following state multiplied by the opportunity cost of that action. The opportunity cost is probablilty of reaching the next state. After you calculate all the utilities, your policy will be the path with the maximum utility. Looking at state S2, it starts with 0 points. Normally we would take the action with the most points related to it but in our example we only have action A3. We then add the outcome of taking action A3 multiplied by their opportunity cost. Since states S3 and S4 don't have any actions leading to other states, their utilities are just their rewards of 5 and 2 respectively. The utility of S2 would then be 3.5. Situations like with S1 in which there are two actions that you can be taking, the utility will be the action that calculates the maximum utility. With the example S1 would have a utility of 3.5 because the utility calcuated from taking action A2 would only produce a utility of 2.

2.3 Evolutionary Computation

To help explain this term we will use the well-known problem, the knapsack problem. The knapsack problem involves taking items and filling a backpack. However, the backpack can only hold a limited weight. Each item has a weight and a value associated with it. A solution to the problem would be a subset of these items with the weights and values of each item added up. From a programming standpoint we will represent the subset of items as an array. If the item is in the solution, the array will have a one in that position, and if it does not have it the array will have a zero. The sum of the values of all items with a 1 is the total value of the solution. Some solutions are not viable because they go over the weight limit; these solutions are considered to have either a value of zero or a negative number depending on your implementation. The ultimate goal is to find the highest value that you can get into the bag.

Evolutionary computation is much like regular evolution and follows the same principle. It begins with a population of randomly generated solutions which in our example would be subsets of items. The next step is to take a couple of the solutions and "breed" them together, taking a bit of two solutions to form a new one. The individuals are picked by a tournament selection. This involves grabbing a subset of you solutions that will be breeding. This type of selection will give higher chances of grabbing a better solution then a worse one. This will allow the solutions that do not have as good of a value have a chance to breed sometimes. This is important because these solutions may have an item in them that is in the overall best solution but is not in the current best. The better solutions have a higher chance however because the overall goal is finding the best solution.

There are a few ways that breeding happens. One common way is uniform crossover. In uniform crossover each solution has a bit of it taken and put into the child solution. With the example problem we would have each item in the items arrays have a 50% chance of taking from parent one and a 50% chance of taking from parent 2. You would also add a tweak, a random change known as mutation, into the newly made solution to prevent them from all converging to the same solution.

3. METHODS

This section describes three different methods which we can use to coordinate multi-robot systems. These methods are: *Swarm* (a method inspired by social-insects), *Auction* (a method based on decentralized auctions), and *Approximation* (a method that uses decision models). The example problem will be the aforementioned cleaning problem in which the robots will be cleaning an office building. Each dirty spot that need to be cleaned and other such chores will be a task that the robots have to accomplish. The robots will have to move around the environment to each spot and clean it, whilst the environment is getting dirty and hence adding new tasks to be done.

3.1 Swarm

Swarm robotics were inspired by observing the interactions of social insects, such as ants. Currently, there is no one way to design individual behavior of a robot to produce the desired collective behavior across problems: each problem calls for a different design, and it is left up to the designer to find the exact design needed for the problem [1]. Most implementations of swarm robotics do not deal with planning ahead and are very reactive to what is happening in the current input of the sensors. There are two different ways, in general, of going about swarm robotics: *behavior-based design* and *automatic design*.

Behavior-based systems involve designing each robot to follow a certain behavior. Inside of behavior-based systems there are a few paradigms, such as probabilistic finite state machine design. In this paradigm each robot changes its state (moving, doing a task, etc.) based on its sensory input. Another paradigm is the virtual physics-based design in which each robot is seen as a particle that exerts forces on the robots around it. This approach is good for tasks such as coordinating movement and forming patterns.

Automatic systems involve using evolutionary computa-

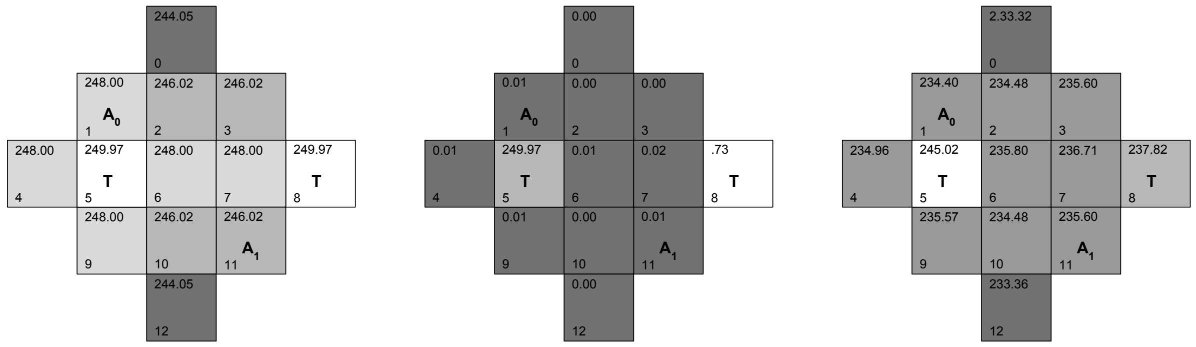


Figure 2: This is a diamond world used by the approximation method. The figure on the left is the original view for robot A_0. The middle figure is A_0s view of what robot A_1 sees. The figure on the right is the final view of robot A_0. [2]

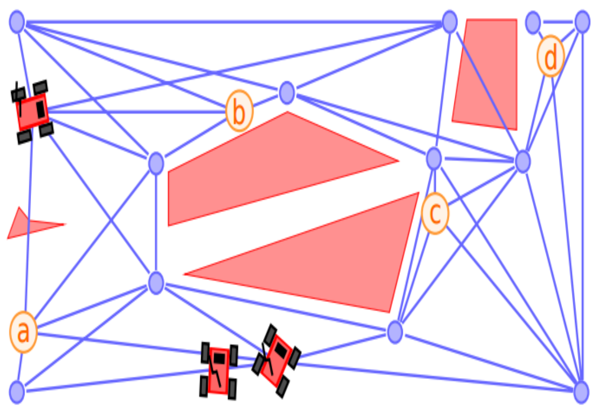
tion to evolve good behaviors for the robots. This means that they will start with a population of behaviors and iteratively "breed" them together to make better behaviors that fit the overall desired behavior. However, this approach is only realistic for situations in which you can simulate the robots because of the overhead involved with evolutionary robotics including uploading code, setting up the robot and environment etc. [4].

While using the swarm method each robot does not plan on how or where it is going to go next. To solve the office cleaning problem we will not be able to come up with an optimal answer to maximize efficiency. However this also means we don't have to compute complex equations to clean an office building. We rely on the group of robots to adequately clean the entire place. Each individual robot would only know about what its sensors could tell it about the world. They would not know where tasks were except if it was in its sensory range. Most likely the robots will be randomly running around looking for tasks: there would have some robots that tend to turn left at forks, some robots that prefer turning rights, so on and so forth. The more robots in the system the spread can be better across the office building.

3.2 Auction

Under the auction control system, each robot is seen as a buyer and each task is seen as an item. An implementation of this method formed by Lozenguez et. al. uses a protocol known as *Simultaneous auctions for coordination* (SSAC). The robots communicate what tasks will be accomplished by what robot. Auctions start when a robot has noticed a change in the set of tasks; that robot then becomes the auctioneer and contacts all other robots in communication distance. The contacted robots stop what they are doing and send a message back to the auctioneer that they are ready. After the auction has begun, it is closed and no robot can leave or enter the interaction. The next step is to determine the current state of the world, in which the robots come up with an initial starting point for the problem. Each robot then, using an MDP problem, calculates the optimal tasks and path. The auctioneer at this point will swap the tasks around to the robots with the higher rewards. These steps will repeat until there is nothing to swap. The robots then go and complete the tasks they have been assigned until the next auction happens [3].

Figure 3: This is an example map used by the auction method. Each blue line is a possible path for the robot to take, each red box is an obstacle, and each lettered dot is a point of interest or a task [3].



For this implementation the researchers had predefined paths that the robot can take. These are represented by a line connecting circle junctions shown in figure 3. Each map also had obstacles as shown by the shaded polygons. The problem that Lonzenguez et al. were trying to solve started with a map taken from a UAV. Some parts of the image were not as good and needed to be checked out and fixed by the ground robots. These spots are indicated on the figure by the lettered circles.

Although these researchers were solving a different problem, there are some parallels to the office cleaning problem. Each obstacle can be seen as a desk and each spot that needs to be checked can be seen as a place that needs cleaning. The way that auction works in the situation begins with all robots in communication with each other. One of them becomes the auctioneer and all robots compute its policy or task sequence. The auctioneer will swap policies around until no better solution can be found. The robots then begin on their assigned tasks. When one robot finishes its task it will send a message to all the robots that are within the

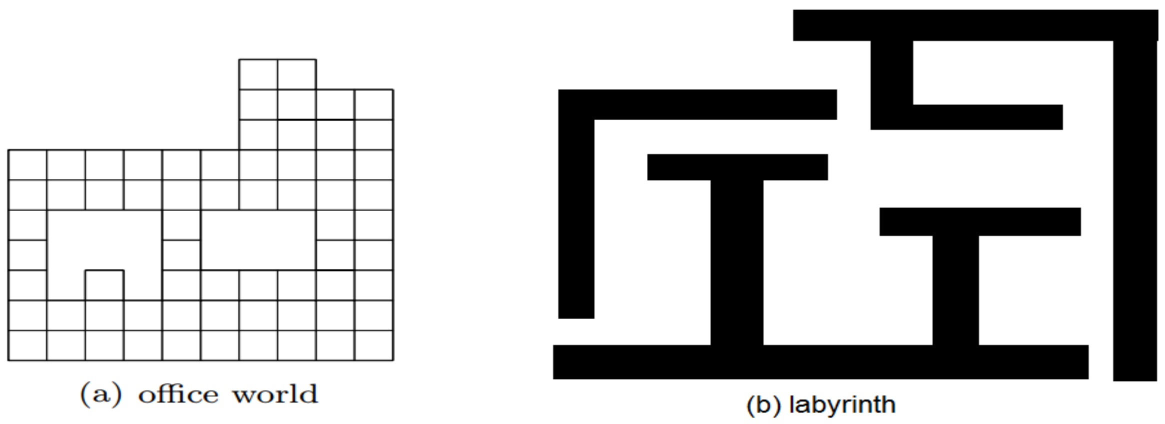


Figure 4: (a) Picture of the office world used in Claes et al. approximation method [2] (b) the labyrinth world used by Lozenguez et al. for the auction method [3]

communication distance determined by the hardware. That robot will then become the auctioneer and the process repeats itself until all tasks have been accomplished.

3.3 Approximation

The third method that we are going to discuss is known as *effective approximation*. This approach uses a variation on Markov decision processes known as *multiagent Markov decision process* (MMDP) along with some approximation methods for task delegation and planning. MMDP is defined by a five part tuple: a set of agents (robots in our example), a set of states of the environment, a set of joint actions, a transition probability function, and an immediate reward function [2]. By adding locations of other robots and task status to the states of the environment, each agent can now calculate which tasks are already being dealt with or probably going to be dealt with, and factor that into the transition and immediate reward functions. The robots will plan their routes by choosing the route with the highest reward.

This approach has to deal with the problem of complexity caused by the number of robots and the large state space caused by the large set of potential tasks. These problems cause the planning to be computationally hard, and a robot with limited capacities cannot handle it. The first assumption that we have to make is that each robot only looks at what is in its individual limited scope. This will reduce the state space and lower the complexity because there are fewer tasks and robots that have to plan for. The tasks that it eliminates will have a lower reward because of the distance it has to travel, meaning good states are not being removed. Then subjective and phase approximation methods are applied to improve on this even more. Subjective approximations are going to make the robot take into account only what it can see personally; while phase approximations are going to make them only focus on the tasks that are currently available. Each agent will only model its own scope and only considers the changes it can make to the state space. This will reduce the state space as well as simplifying the transition and reward functions because there is less the robot has to compute. Along with this each agent only needs to know what other tasks are going to be done by other robots. The individual will take in the other robots locations and make the tasks that the others are probably not going for have higher rewards. Furthermore instead of thinking of each spot as a potential task location, each robot can consider only the current task locations. If even this is not enough to reduce the state space, the individual robot can consider only the closest k tasks. These approximation methods in tandem will make the problem at hand feasible for robots.

In this method each robot knows about every task and robot around it and will plan its moves for up to 20 tasks that it will complete. This plan can change seeing that another robot gets to a task first. In figure 2 we can see three diagrams in a diamond shape. These are the steps that the robot will take in the planning phase. In the first diagram there are two robots: one in square $1(A_0)$ and one in square $11(A_1)$. There are also two tasks: one in square 5 and one in square 8. Each square is of a different shade of color which corresponds to the value of going to that square, lighter being highly valued, dark being low valued. The actual number value of the square is in the top left corner. This is robot A_0 initial view of the world to see where it wants to go. The next diagram is robot A_0 view of what robot A_1 is seeing. From this view we can then discount the original diagrams spaces to yield the last diagram's spaces. Robot A_0 will then proceed to the task in square 5 because it has the highest reward associated with it.

There is future work to be done in expanding the Approximation method. It currently cannot handle tasks that require execution in a specific order, nor can it organize the execution of tasks requiring cooperation of robots.

4. RESULTS

The implementations of swarm robotics are limited to three different areas: spatial organization problems, exploration problems, and collective decision problems. The collective decision making problems are still limited by the non-planning paradigm of swarm robotics. The advantages that this method has over the others are that they do have to compute less and work very well for foraging and spatial organization problems. Unfortunately, Brambilla and the rest of the research team did not give any quantified results to

Table 1: These are the auction method results.[3].

Tasks	Score	Worst Score	%100%
2-3	99.5	78.7	86.8
4-5	99.7	88.8	63.0
6-7	99.2	75.8	42.0
8-9	99.0	75.0	34.0
10-11	98.8	82.8	34.0
12-13	98.8	86.8	23.0

compare with the other two methods. The auction and approximation methods, however, will outperform swarm because they do find an optimal, or close to optimal, path for the robots to take.

The auction method had very good results for a similar problem to the office cleaning. The robots were tasked to investigate spots of interest in a few different map types: labyrinth and a few obstacles. They had both of these map types with a range of how many tasks they had to accomplish, from 2 to 13 total tasks. Tasks are assumed to take one time step each. Table 1 shows the scores for the number of tasks that the robots had to do for the auction method. The scores are based on the percentage of the upper bounds. The worst score is the farthest away that it got from the upper bound. The last section labeled % 100 % is the percentage of times the method got exactly the upper bound Scores are computed to a percent by taking the rewards of each robot added together over the maximum rewards that it could have achieved. In the few obstacles map the robots were able to receive a score of 99.5% for 2 to 3 tasks and sinking down to a 98.4% for 12-13 tasks. In the labyrinth map, as shown in 4b, they ranged from 99.7% for 2 to 3 tasks to 98.8 for 12 to 13 tasks as seen in table 1. These as shown later outscore approximations method but there are other factors that we need to take into account while comparing these numbers. This method will outshine the swarm robotics method because it can find the optimal solution to the problem where swarm would be a bit more variant; Swarm scores would average to be lower because swarm will not plan ahead to find the optimal, or close to optimal solution.

The approximations method delivered impressive results after several tests and changing variables to find the best results. The variables they changed are: how many tasks to consider in the MMDP, and how many steps to look ahead. The number of clean squares as shown in figure 4A at every time step is added to the teams of robots score. Table 2 shows the results when the robots were solving the office cleaning problem. The type of world is the type of environment the robots were in. % of the upper bound is how close the robots got to the maximum score they could of gotten had they picked most optimal task. In small worlds, such as a 3x3 square, this method got as high as 97.24% close to optimal solution. However, as the worlds get bigger and more complex this number drops to a reasonable 70.4% for an office environment as seen in table 2. Although these numbers are lower than the auction method, there are a few things that this method does that the other one does not. The office environment as shown in figure 4a had 66 squares in it meaning that there are 66 possible tasks compared to the 12 to 13 in auction. This method also adds in tasks while the experiment was running. Each square has a .05 probability to turn to a task during each time step. Again tasks are assumed to take one time step each.

Table 2: These are the results of the approximation method. [2].

Type of world	% of Upper Bound
Line	79.2%
Diamond	85.7%
Corridors	79.6%
2x2	98.4%
3x3	97.2%
4x4	87.8%
6x6	76.8%
Office	70.4%

The approximation method has some clear advantages over both swarm and auction. Swarm does not plan, and therefore cannot ensure as efficient of a system. Swarm also does not adapt as well to new or different tasks that appear in the world as approximations does [2]. Auctions have to stop the robots to reallocate tasks, where approximations can dynamically change at every time step and does not need to stop all robots, just the one that is changing. Auctions also do not consider sub-tasks, where a task has more than one action to complete, or tasks that need to be done in certain order where approximations can handle them without much change.

5. CONCLUSION

In multi-robot systems as the number of robots and the number of coordinated tasks that need to be completed grow, the quicker that system will outgrow a centralized control system. There have been three solutions to managing these kinds of systems: swarm, auction, and approximations. Each has advantages and disadvantages. Swarm is a very good solution to foraging and pattern-making types of problems, but is lacking in the ability to plan for upcoming tasks for efficiency. Auction is promising because it can plan and come up with optimal solutions for everything swarm robots can do and more, but is limited by the inability to dynamically change at any time and can only change the plan during the brief moments in which the robots are communicating. Approximation can plan solutions that are close to optimal and can change if new tasks appear or if another robot gets to the other tasks before it can. However, the solutions can get as low as 70.4% for some problems which is still reasonable. Each method has a type of situation that they best fit, and it up to the designer of a system to make the appropriate choice.

6. ACKNOWLEDGMENTS

I would like to express my most sincere gratitude the Elena Machkosova for advising me throughout this whole process. I would also like to extend this to the rest of the computer science professors for helping me learn and become the computer scientist I am today. I would also like to thank Wayne

Manselle and Elizabeth Nault-Maurer for thier comments on my paper as well as my friends and family for giving me the encouragement that I needed to get the paper done.

7. REFERENCES

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[2] D. Claes, P. Robbel, F. A. Oliehoek, K. Tuyls, D. Hennes, and W. van der Hoek. Effective approximations for multi-robot coordination in spatially distributed tasks. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 881–890, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[3] G. Lozenguez, A.-I. Mouaddib, A. Beynier, L. Adouane, and P. Martinet. Simultaneous auctions for "rendez-vous" coordination phases in multi-robot multi-task mission. In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02*, WI-IAT '13, pages 67–74, Washington, DC, USA, 2013. IEEE Computer Society.

[4] Schiller. Applying evolutionary computation to robotics. In *UMM CSci Senior Seminar Conference*, Morris, MN, USA, 2014.

[5] Wikipedia. Markov decision process — Wikipedia, the free encyclopedia, 2016.

[6] Wikipedia. Swarm robotics — Wikipedia, the free encyclopedia, 2016.