Learning and Exploiting Knowledge in Multi-Agent Task Allocation Problems

Adam Campbell
School of EECS
University of Central Florida
Orlando, FL 32816-2362, USA
acampbel@cs.ucf.edu

Annie S. Wu
School of EECS
University of Central Florida
Orlando, FL 32816-2362, USA
aswu@cs.ucf.edu

ABSTRACT

Imagine a group of cooperating agents attempting to allocate tasks amongst themselves without knowledge of their own capabilities. Over time, they develop a belief of their own skill levels through failed attempts at completing the tasks they are assigned. How will various task allocation approaches perform when there exists this added level of complexity? In particular, we compare two task allocation strategies: a greedy, first-come-first-serve approach, and a more intelligent, best-fit method. By varying the number of tasks along with the amount of time it takes to complete those tasks, we find that the different task allocation methods work better in different situations. Because of the way the tasks are allocated by the two methods, the greedy approach does a better job of giving agents opportunities to learn their capabilities. Thus, the greedy approach allows for quicker learning and performs better on problems where the task durations are short, whereas the best-fit method performs better on problems where the task quantity and durations are large. What is needed is a hybrid method that balances between the exploration of the greedy approach and the exploitation of the best-fit method.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*

General Terms

Experimentation

Keywords

multi-agent systems, task allocation

1. INTRODUCTION

Given a group of cooperative agents, each with a different task completion ability and a set of tasks that vary with difficulty, how can the agents learn their own abilities so that when new tasks arrive they are assigned to agents in the most efficient manner? To answer this question, we develop a simple model that allows us to compare different task allocation methods in the presence and absence of learning. When learning is used, agents will be able to estimate their skill level through a series of trial-and-error tests. Because agents do not know their true skill level, they are unable to perfectly discern their ability to complete any particular task. In this paper, we look to discover how two types of task allocation methods perform under this type of scenario. The first task allocation method is a first-come-first-serve, greedy approach that assigns the task to a randomly chosen available agent that believes it can complete the task. The second approach attempts to assign the agents to tasks in a more intelligent manner by giving the tasks to agents that are believed to be best fit for the task.

We find that the method of task allocation has a large effect on the learning capabilities of the agents. Because of the way the best-fit task allocation method assigns tasks to agents, it does a poor job of distributing the learning to the agents, whereas the randomness of the greedy method allows for very quick learning. However, the best-fit method has the upper hand when the agents have an accurate estimation of their abilities. Agents with little skill take on the easier tasks, which frees up the highly skilled agents to take on the difficult tasks. What would be best is a hybrid method that allows agents to quickly learn their skills early on and then later exploits this knowledge through an efficient task allocation procedure.

2. RELATED WORK

Social insects are typically used as the canonical example of massively distributed problem solving because of their ability to manage task distributions without the need for centralized control [1]. As an example of how social insects allocate tasks, Franks [2] shows that ants of different sizes take on tasks of different difficulties. Foragers that are larger are apt to carry larger food, while the smaller ants take on more manageable tasks.

Work on task allocation within the multi-agent research field is a subset of the distributed problem solving work in the Artificial Intelligence community. In [8], Smith and Davis describe task-sharing, where processing nodes decompose tasks into subtasks and then send the subtasks off to other nodes. Difficulty arises when deciding which nodes should take which tasks so as to minimize processing time.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

Early multi-robot architectures that were focused on solving the task decomposition and allocation problems were designed to be flexible, fault tolerant, and decentralized [7, 5]. To determine task allocations, negotiations take place through bidding systems, where available agents place higher "bids" on tasks they are well suited to work on. Agents with higher bids are then assigned to these tasks. Tasks can be dynamically created, decomposed into smaller tasks, and assigned at run-time. These early architectures paved the way for the more recent research.

As claimed by Gerkey and Matarić in [3], most of the task allocation work up to the point of the writing of their paper had been empirical and contained little theoretical foundation. In their work, Gerkey and Matarić present a formal framework for task allocation. Their taxonomy of the multi-robot task allocation problem uses three criteria, with each criteria containing two values, thus, creating eight classes of task allocation problems. The complexity of each classification is analyzed separately, so that once new problems arrive, they can be classified, and their theoretical complexity will be already known. Lau and Zhang [4] present a similar work that gives a taxonomy for what they call the Task Allocation Coalition Formation problem.

Matarić *et al.* [6] describe a multi-robot task allocation problem where the cooperating robots receive noisy data from the environment; thus, causing the robots to bid incorrectly on tasks. Four task allocation methods were described in [6], and the authors found that the best allocation method was dependent upon the type of noise in the robots' sensor information. If agents are ignorant of their capabilities, then they too will produce erred bids on tasks. Our goal is to get a better understanding of a system where a group of autonomous agents are working together, allocating tasks based on what they *believe* their skill levels to be and not on what the actual skill levels are.

3. METHOD

To understand how a group of agents, ignorant of their own capabilities, can allocate tasks efficiently, a simulation model was constructed. The simulation consists of a set of agents and a set of tasks. Each agent contains the following three variables: *actualSkillLevel*, *estSkillLevel*, and *timeBusy*. The *actualSkillLevel* is a randomly chosen, floating point number in the range of *0.0* to *1.0*. The *estSkillLevel* (estimated skill level) is initialized to *1.0* and represents what the agent believes its *actualSkillLevel* to be. If learning is present it will be updated by the agent through a series of trial-and-error tests. The *timeBusy* value of an agent represents the amount of time that the agent is occupied with its current task. When an agent's *timeBusy* value is greater than zero, it cannot be assigned a new task.

Each task has a *difficulty* and a *duration*. An agent can only complete a task if its *actualSkillLevel* is greater than or equal to the *difficulty* of the task. As a practical consideration, *difficulty* ranges from *0.0* to *0.9* in order to reduce the possibility of a task being too difficult for any agent to complete. The *duration* is a positive integer that indicates the amount of time an agent will be occupied when it has been assigned a task. One may wonder why the *duration* and *difficulty* of tasks are not related. It would seem to make sense that the *difficulty* of a task would define its *duration*; however, this is not necessarily the case. In a foraging scenario where the tasks are the retrieval and transportation of target items back to a home base, the *difficulty* could represent how heavy or big the target object is, whereas the *duration* of the task would be a function of the distance between the object and the home base. Since we assume that it will take any agent the same amount of time to complete some task, T, we can ignore T's *duration* when computing an agent's ability to complete it. Also, because we are interested in the general, behavioral trends that occur when the previously mentioned parameters are varied, our model does not take into account the mobility of agents and the problems that agent-agent interference creates.

Algorithm 1 shows the basic outline of the simulation model:

Algorithm 1: `Model(`$AGENTS$`,` $TASKS$`,` *learning*`)`

```
(1)  t := 0
(2)  if TASKS = ∅ then
(3)     return t + max{A_i.timeBusy : A_i ∈ AGENTS}
(4)  ∀A_i ∈ AGENTS : A_i.timeBusy > 0
(5)     A_i.timeBusy := A_i.timeBusy − 1
(6)  AVAIL = {A_i : A_i ∈ AGENTS &&
                    A_i.timeBusy = 0}
(7)  if AVAIL = ∅ then
(8)     GOTO Line 18
(9)  select random task T from TASKS
(10) if ∄A_i : A_i ∈ AVAIL &&
          A_i.estSkillLevel > T.difficulty then
(11)    GOTO Line 18
(12) find an agent A to complete task T
(13) if A.actualSkillLevel >= T.difficulty then
(14)    TASKS = TASKS − T
(15)    A.timeBusy := T.duration
(16) else if learning = true then
(17)    A.estSkillLevel := T.difficulty
(18) t := t + 1
(19) GOTO Line 2
```

The variable t is returned by the model and represents the amount of time it took for the agents to complete all tasks. Lines 2 and 3 of Algorithm 1 are used to check if the task set is empty. If it is empty, then the program returns the number of iterations it has run in addition to the maximum *timeBusy* value that any agent currently has. To simulate the agents working on their tasks, the *timeBusy* value for each unavailable agent is decremented by one in Lines 4 and 5. Line 6 is where the set of available agents, *AVAIL*, is created. Lines 7 and 8 are used to check if *AVAIL* is empty, and if so, then the program jumps to Line 18 where t is incremented by one and then the process starts again through the jump at Line 19. Next, Line 9 selects a random task T from *TASKS*. If there does not exist an available agent that has an *estSkillLevel* greater than the difficulty of T, then the program jumps to Line 18. In Line 12, an agent A is chosen to complete T. If the agent finds it can complete the task (ie. its *actualSkillLevel* is at least as large as the *difficulty* of T), then T is removed from *TASKS* and A has its *timeBusy* value set to the *duration* of T (Lines 13, 14, and 15). If *learning* is *true* and A's *actualSkillLevel* is below the *difficulty* of T, then A adjusts its *estSkillLevel* to equal the *difficulty* of T. This trial-and-error process is how the agents learn their own skill level over time. Line 12 is where the task allocation method comes into play and is where the remainder of the paper will be focused.

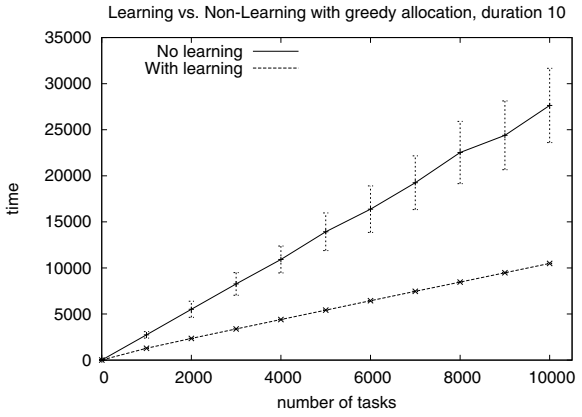

Figure 1: These two plots show the comparison between the average completion times of the learning and non-learning, greedy task allocation methods when the duration of each task is 10.

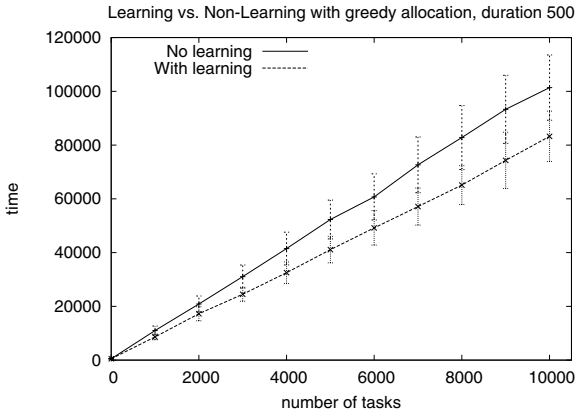

Figure 2: These two plots show the comparison between the average completion times of the learning and non-learning, greedy task allocation methods when the duration of each task is 500.

Each task allocation method is given *AVAIL* and T as parameters and returns an agent to complete T. It should be noted here that the order of the agents in *AVAIL* is randomized before the task allocation method is called so that there is no implicit bias towards any of the agents.

The greedy task allocation method works by looping through each agent in *AVAIL* and returning the first one that has an *estSkillLevel* greater than the *difficulty* of T. Because Lines 10 and 11 of Algorithm 1 ensure that at least one agent in *AVAIL* believes it can complete T, we know that at least one agent in *AVAIL* will have an *estSkilLevel* greater than T's *difficulty*. Notice that the *estSkillLevel* must be strictly greater than the task's *difficulty*. Because an agent's *estSkillLevel* is equal to the *difficulty* of the least difficult task it could not complete (Line 17 of Algorithm 1), we know that the agent cannot complete a task with a *difficulty* equal to its *estSkillLevel*.

The best-fit method loops through all agents in *AVAIL* and returns the agent A that has the minimum, positive value for $A.estSkillLevel$ - $T.difficulty$. The agent that estimates its skill level to be closest to that of the task, without being below or equal to the task difficulty, takes the task. Whether or not it can actually perform the task is not known to the agent, but our belief is that with enough trials the agents can begin to get a good estimate of their capability and allocate themselves to tasks that best suit their ability.

4. EXPERIMENTS

To determine the behaviors of the task allocation methods with and without learning, the task quantity and task duration time will be varied. By modifying these two parameters, different problem domains can be modelled.

The first set of experiments will compare the non-learning greedy approach to the learning greedy approach. The non-learning greedy task allocation method will serve as a baseline comparison and allow us to see the effects learning has on the task allocation procedure. In each of the following experiments, the number of agents remains at 100 and the plots are obtained by averaging the time-to-complete values over 100 runs. Figure 1 shows the comparison between the learning and non-learning methods when the duration of each task is 10 time units. As the number of tasks increases, the difference between the two lines in the plot also increases, indicating that the benefits of using the learning method are also increasing. Figure 2 shows what happens when the duration of the tasks increases. Because the duration of each task is much longer than those in the previous experiments, agents are occupied with tasks for a much longer time. Thus, it becomes more costly to assign an agent with a high skill level to a simple task since that agent will not be able to complete the more difficult tasks that come along during those 500 time steps it is working on the simpler task. The two lines in Figure 2 are closer together than the ones in Figure 1, indicating that the learning method provides a smaller benefit over the non-learning method when task duration increases.

num. tasks	10 ticks	100 ticks	500 ticks
20	1.085	1.055	.997
100	1.308	1.414	1.233
200	1.585	1.630	1.198
1000	2.170	2.172	2.174
5000	2.558	2.430	1.251
10000	2.692	2.378	1.260

Table 1: Ratio of non-learning times to learning times: $time_{withoutLearning}/time_{withLearning}$.

Table 1 shows the ratio of non-learning to learning times ($time_{withoutLearning}/time_{withLearning}$) for several experiments. The top row of the table shows the length of the task durations, while the left most column denotes the number of tasks used for the experiments. When the number of tasks is as small as 20, there is no significant difference between the average completion times of the two methods. However, when the task duration is small, a steady increase in speedup can be seen, indicating that when the tasks take little to no time to complete, a more sophisticated method of task allocation may not be necessary. However, when the task duration and task quantity are large, the benefits of using the learning method decreases. This can be seen by comparing the values in the right-most column. One possible explanation for this is due to the inefficient way resources are allocated by the greedy method. By using a more sophisticated method of task allocation that assigns agents to

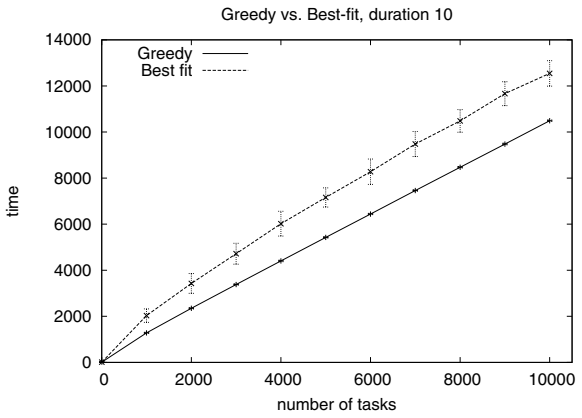

Figure 3: These two plots show the comparison between the average completion times of the greedy and best-fit task allocation methods when the duration of each task is 10.

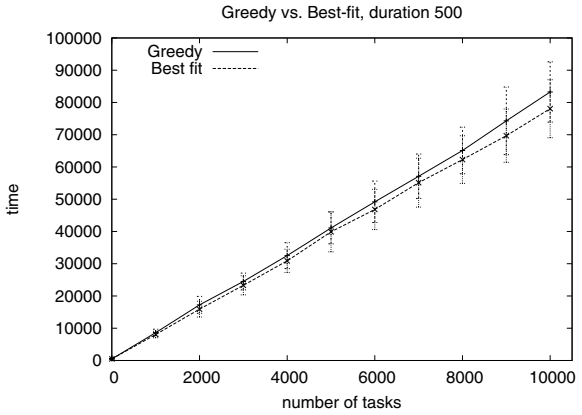

Figure 4: These two plots show the comparison between the average completion times of the greedy and best-fit task allocation methods when the duration of each task is 500.

tasks in an intelligent manner, we may find a speedup when the task durations are long and the number of tasks is large.

The second set of experiments will compare the learning greedy task allocation method described above to the best-fit method. In both cases, learning is used, the number of agents is 100, and the data will be gathered by averaging the output from 100 runs. The plots from the first set of experiments are shown in Figure 3.

The greedy method, actually outperforms the best-fit method when the task duration is 10 time steps. Because of the way the best-fit method selects agents, it may actually hinder the learning process. This hypothesis will be examined further in an experiment below. When the duration of the tasks is increased to 500, and the penalty for not allocating agents efficiently is higher, the best-fit method does slightly outperform the greedy method, although this result does not look to be statistically significant (Figure 4).

To determine if learning is hindered by the best-fit task allocation method, we will look at the average difference between *estSkillLevel* and *actualSkillLevel* throughout time.

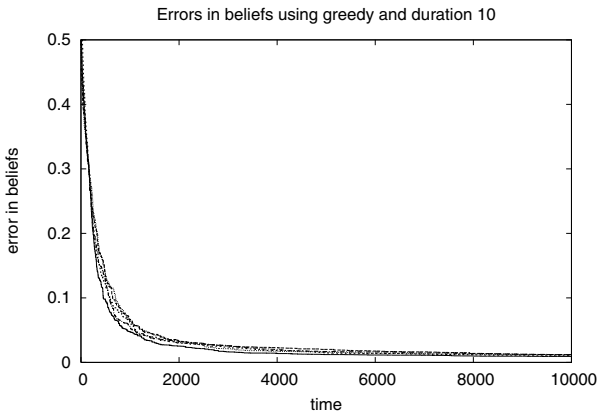

Figure 5: In these plots, the average difference between all agents' *estSkillLevel* and *actualSkillLevel* are plotted over time. In each of the five runs the following parameters are used: 100 agents, 10000 tasks, and a task duration of 10. The plot gives an indication of how fast the learning method takes place when greedy task allocation is used.

The idea being that when the difference between an agent's estimated skill level and actual skill level are small the agent has learned its capabilities. If in fact the best-fit task allocation method is somehow slowing down the learning process, we will see the difference between beliefs and actual skill level getting smaller at a slower rate than when the greedy method is used. Figures 5 and 6 show just that.

In each of these figures the data for five runs are shown; with each run consisting of 100 agents, 10000 tasks, and tasks with durations of 10 time units. Figure 5 shows how fast learning occurs when the greedy method is used, and Figure 6 shows this for the best-fit method. When the greedy method is used, the difference between beliefs and actual skill level drops extremely fast and then asymptotically approaches zero. The learning rate for the best-fit method is more of a linear curve. Thus, it has been shown that the best-fit method does prevent learning from taking place at a fast rate. What is it about the way that agents are selected in the best-fit method that makes learning slower?

One possible explanation for why the best-fit method harms learning could be due to scenarios similar to the following. At the beginning of a new run, an agent, A, with an *actualSkillLevel* of *0.2* gets selected to perform a task with a *difficulty* of *0.85*. Because A cannot complete the task, it will set its *estSkillLevel* to *0.85*. Now, let's imagine that in the next time step, a task with difficulty *0.80* is chosen. When the greedy method is used, each of the 100 agents has an equal opportunity to take on this task since they all have their *estSkillLevels* greater than the task's difficulty. Chances are, that the agent that got chosen for the first task will not get chosen for the second one. Since the agents' *actualSkillLevel* values were randomly chosen between *0.0* and *1.0*, about 20% of the agents will be able to complete the task; the other 80% will not. So, it is very likely that the next agent chosen by the greedy method will have an *estSkillLevel* equal to *1.0*, and an *actualSkillLevel* less than *0.80*. The agent chosen (if it is one that cannot complete the task) will then adjust its *estSkillLevel* to *0.80*. After two iterations of the simulation,

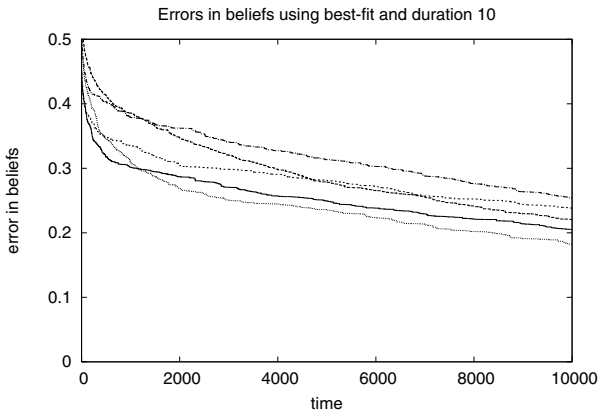

Figure 6: In these plots, the average difference between all agents' *estSkillLevel* and *actualSkillLevel* are plotted over time. In each of the five runs the following parameters are used: 100 agents, 10000 tasks, and a task duration of 10. Here, the best-fit method is used. When compared to Figure 5, it can be seen that the best-fit method slows the learning process.

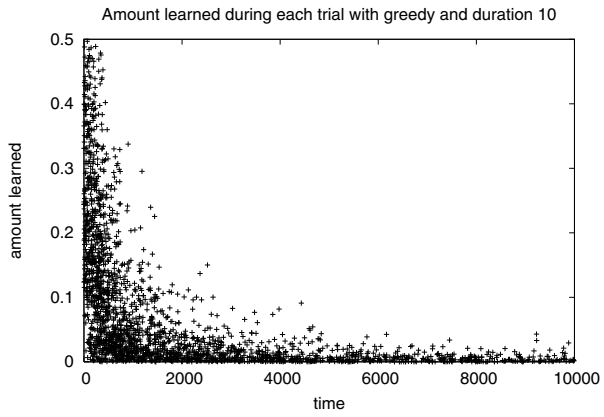

Figure 7: This plot shows when a majority of the learning takes place when the greedy task allocation method is used. A point was generated on the graph each time Line 17 of Algorithm 1 was executed. The point gives the difference between the agent's *estSkillLevel* and the *difficulty* of the task it could not complete. Thus, points higher on the y-axis show when an agent adapted its *estSkillLevel* by a large amount. The data points were gathered from five separate runs.

one agent will have an *estSkillLevel* of *0.85* and the other will have an *estSkillLevel* of *0.80*. What would happen in this scenario when the best-fit method is used? After the first iteration, the chosen agent would set its *estSkillLevel* to *0.85*. In the second iteration, the same agent would get chosen for the *0.80* difficulty task since it estimates its skill level as being closest to the task difficulty (remember, all of the other agents have an *estSkillLevel* of *1.0* at this point). Therefore, after two iterations, with the best-fit method, one agent would have an *estSkillLevel* of *0.80* and all others would have *1.0 estSkillLevels*. On average, two agents would have learned with the greedy method, whereas only one learned when the best-fit method is used. Thus, because the greedy method allows more agents the opportunity to be assigned tasks, it allows for quicker learning. The following experiment will try to test the validity of this explanation.

The following experiments were set up to determine the amount of learning that takes place over time. Each time an agent adjusts its *estSkillLevel* (Line 17 of Algorithm 1), the amount learned ($estSkillLevel - difficulty$) will be printed before Line 17 is executed. If the greedy method allows agents to adjust their *estSkillLevels* faster than the best-fit method, then we should see higher values for $estSkillLevel - difficulty$ early on when the greedy method is used. Figures 7 and 8 show how learning takes place over time in the greedy and best-fit methods, respectively.

Both of these plots were obtained through five runs. Each time Line 17 of Algorithm 1 was reached, a point on the graph was made. The point shows the current simulation time and the $estSkillLevel - difficulty$ value. Large values of $estSkillLevel - difficulty$ mean that the agent is adjusting its *estSkillLevel* by a large amount, i.e., it is learning a lot. Figure 7 shows that a large amount of learning takes place early on when the greedy task allocation method is used. When the best-fit method is used, the learning takes place at a much slower rate (Figure 8). It is also interesting to see the large number of points along the *y=0.1* line. This is most likely because of the way the best-fit method works along with the fact that the highest *difficulty* for a task will be near *0.9*. When the best-fit method is used, agents that are not selected early on will have little chance to be selected later on because their *estSkillLevel* will be at *1.0* and their $estSkillLevel - difficulty$ values will be high. However, when a task comes along with a high difficulty (near *0.9*), most of the adapted agents will have *estSkillLevels* below *0.9*, so the agents to take on those tasks will most likely be the ones with *estSkillLevels* at *1.0*. When they are chosen, their $estSkillLevel - difficulty$ value will be near *0.1*, and this is why we see a lot of points around the *y=0.1* line.

5. CONCLUSION

Four sets experiments were conducted so that we could get an understanding of the behaviors in a multi-agent task allocation problem when the agents are ignorant of their own capabilities. The first experiment showed a comparison between a system with learning and one without learning when a greedy, first-come-first-serve, task allocation method was used. It was shown that the learning method outperforms the non-learning method significantly when the duration of the tasks is not very long. However, when the duration of the tasks becomes longer, the benefits of using the learning method decrease. This indicates that a more sophisticated method of task allocation that utilizes resources efficiently may be needed when task duration is long.

The second set of experiments compared the greedy task allocation method to a best-fit method that assigns tasks to agents that believe they are best for the job. When the durations of the tasks were short, the greedy method outperformed the best-fit method. This indicates that the best-fit method slows down the learning process of the agents so much that the benefits gained from allocating resources efficiently are not seen. By increasing the duration of the task, and in so doing, increasing the need to allocate resources efficiently, the best-fit method does begin to outperform the greedy method.

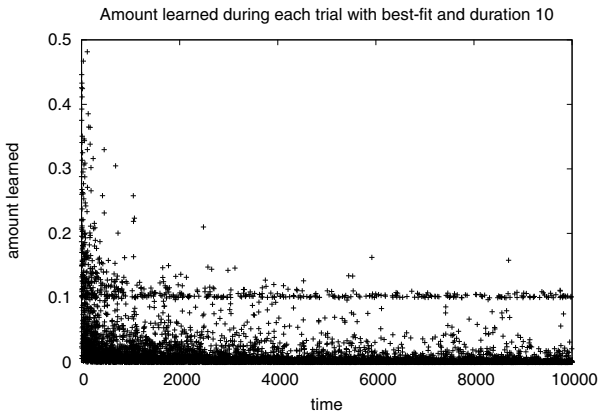

Figure 8: Compare this plot to Figure 7 to see the differences in learning when the best-fit method is used instead of the greedy method.

In the third set of experiments, it was shown that the best-fit learning method does reduce the ability for the agents to learn. Because of the way agents were selected, they adjusted their *estSkillLevel* at a much slower rate than when the agents were using a greedy task allocation approach. The fourth experiment supported this hypothesis.

It has been shown here that when agents are initially ignorant of their own capabilities, the best task allocation method used has a drastic effect on the learning capabilities of the agents. The greedy task allocation method does a good job of allowing the agents to explore, and thus allows them to quickly learn their capabilities. However, once these agents are adapted, they can do little to exploit this knowledge and allocate the resources efficiently. The best-fit task allocation method behaves in a different manner. Because of the way it assigns tasks to agents, it does a poor job of allowing the agents to learn their capabilities. The best-fit method does a better job of allocating resources efficiently, but because it prevents the agents from learning quickly, it provides little benefit over the simple, greedy method when task durations are short. But, when the task durations become long, we find that the benefits of the best-fit method are noticed. What would be best is a hybrid method that allows the agents to explore and learn early and then exploit this knowledge later on.

Another direction for future research is to take the existing simulation and modify it so that it can model decentralized decision making. For simplicity, the model presented in this paper used centralized control. However, real multi-agent teams work in a decentralized fashion. Due to the dispersal of the agents throughout their environment and the physical constraints on agent-agent communication distance, agents will only be able to communicate with their immediate neighbors.

This will limit the number of agents that can bid on any particular task. We believe that this may be able to be modeled by limiting the size of the *AVAIL* set in Algorithm 1. This modified version of the model may allow for experimentation on decentralized multi-agent task allocation problems in the presence of learning.

6. ACKNOWLEDGEMENTS

This work was sponsored, in part, by the US Army Research Laboratory under Cooperative Agreement W911NF-06-2-0041. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARL or the US Government. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Thanks also to Randall Shumaker and the Institute for Simulation and Training.

7. REFERENCES

[1] C. Anderson and N. R. Franks. Teams in animal societies. *Behavioural Ecology*, 12(5):534–540, 2001.

[2] N. R. Franks. Teams in social insects: group retrieval of prey by army ants (eciton burchelli, hymenoptera: Formicidae). *Behavioral Ecology and Sociobiology*, 18(6):425–429, 1986.

[3] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, September 2004.

[4] H. C. Lau and L. Zhang. Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In *ICTAI '03: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, page 346, Washington, DC, USA. IEEE Computer Society.

[5] T. C. Lueth and T. Laengle. Task description, decomposition and allocation in a distributed autonomous multi-agent robot system. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 1516–1523, September 1994.

[6] M. J. Matarić, G. S. Sukhatme, and E. H. Østergaard. Multirobot task allocation in uncertain environments. *Autonomous Robots*, 14(23):255–263, 2003.

[7] F. R. Noreils. An architecture for cooperative and autonomous mobile robots. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2703–2710, May 1992.

[8] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. SMC-11(1):61–70, January 1981.