Multi-Agent Task Allocation: Learning When to Say No

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

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

Randall Shumaker
Institute for Sim. and Training
University of Central Florida
Orlando, FL 32826
shumaker@ist.ucf.edu

ABSTRACT

This paper presents a communication-less multi-agent task allocation procedure that allows agents to use past experience to make non-greedy decisions about task assignments. Experimental results are given for problems where agents have varying capabilities, tasks have varying difficulties, and agents are ignorant of what tasks they will see in the future. These types of problems are difficult because the choice an agent makes in the present will affect the decisions it can make in the future. Current task-allocation procedures, especially the market-based ones, tend to side-step the issue by ignoring the future and assigning tasks to agents in a greedy way so that short-term goals are met. It is shown here that these short-sighted allocation procedures work well in situations where the ratio of task length to team size is small, but their performance decreases as this ratio increases. The adaptive method presented here is shown to perform well in a wide range of task-allocation problems, and because it requires no explicit communication, its computational costs are independent of team size.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

multi-agent systems, task allocation, adaptive systems

1. INTRODUCTION

Imagine two autonomous rovers foraging for rocks on Mars. *RoverS* is very strong and can pick up heavy rocks, whereas *RoverW* is weak and can only pick up small ones. The rovers approach a small rock that both of them can carry and must make a decision about who should take it. If the agents are able to communicate with each other, then each can inform the other about their valuation of the task, and a decision can be reached about who is best fit for the job. *RoverS* is found to be "over-fit" for this simple task, and so, they decide it is best for *RoverW* to take it. Their decision allows *RoverS* to be available for more appropriate jobs that might come along in the near future.

Negotiation-based task allocation procedures require agents to communicate with one another in order to come to a consensus about who should take on which tasks. However, in certain situations, it may be prohibitive, or impossible, for agents to explicitly communicate their task valuations with each other. Complicating the matter further is the fact that it is not always best to assign tasks in a greedy manner because agents may not know what sort of tasks they will see in the future. For example, should *RoverS* pick up a small rock if *RoverW* is currently occupied, or is it better for *RoverS* to ignore this simple task and try to find bigger rocks? If *RoverS* assumes that all rocks are small because it has never come across a big one, then it might be best for it to take a small rock. However, if *RoverS* assumes that it will see another big rock because it has already seen plenty of them, then it should become less apt to pick up small ones in order to use itself more appropriately.

Both empirical and analytic techniques will be used to answer the following questions: Under what conditions should an agent behave greedily by always taking tasks when given the opportunity? When should an agent ignore tasks because past experience indicates that a more appropriate task will present itself in the near future? In particular, we will explore the relationship between team size and task length in problems where agents have varying capabilities, tasks have varying difficulties, and the distribution of those task difficulties is unknown to the agents.

The results show that a non-greedy task allocation procedure requiring no explicit communication can, in certain situations, perform better than communication-based, greedy approaches. The presented task allocation procedure is adaptive, and so, is able to handle many different task difficulty distributions. Also, because there is no communication between the agents, the computational cost of the algorithm is independent of team size.

2. BACKGROUND

Multi-agent task allocation is the problem of assigning tasks to agents so that some overall goal or goals are met. This definition is left general because the problems

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

a roboticist faces when designing a multi-robot decision-making process in a foraging domain are analogous to the problems an operating systems designer faces when writing algorithms for deciding how best to assign processes to CPUs on a multiprocessor machine. In each case, there exists a set of agents, a set of tasks, a utility measure for each agent-task pair, and the problem of divvying up the tasks to agents so that some overall goals are satisfied. The task allocation problem is central to cooperative multi-robot systems research [9, 11, 13, 14, 15, 18], multiprocessor scheduling design [3], problems in economics [6], and even of interest to biologists as they learn about division of labor within insect colonies [2, 5, 17].

Gerkey and Matarić [9] give a taxonomy for multi-robot task allocation problems and present the computational complexity analysis of the problems in each class. Although their goal was to classify multi-robot problems, their study is general enough to apply to most multi-agent situations. Some task allocation problems are nothing more than instances of the *Optimal Assignment Problem*, and thus, can be solved with both centralized linear programming techniques [10] and distributed, auction-based algorithms [1, 8]. In both cases, agents must be able to determine their own valuation of the tasks, communicate these with each other, and then make a decision about who should take what task so that the goals are met. More information on these types of problems and their solutions can be found in Gale's [6] work on using linear algebra to perform economic analysis, Kuhn's [10] Hungarian Method and Bertseka's [1] distributed auction-based algorithm for solving assignment problems, and Gerkey and Matarić's [8] auction-based algorithm for multi-robot task allocation problems.

The problems described above are examples of *assignment problems* where the goal is to maximize the overall utility of the mapping between agents and tasks. These types of problems have been shown to be in the computational complexity class *P*. However, task allocation becomes much more difficult when each task requires a certain amount of time to complete, there are more tasks than there are agents, and we must come up with a *schedule* for assigning tasks to agents so that some deadline is met. These types of problems are *scheduling problems* and are known to be *NP-complete* [3, 7]. Complicating the matters further are *on-line problems* (or dynamic task allocation problems [12]), where the set of tasks is not known *a priori* and task allocation decisions must be made without knowledge of what tasks the future will bring [4]. El-Rewini *et al.* [3] present variants of the scheduling problem, heuristics and algorithms that solve these variants, and also take into account how the costs of communication between processors affect the problems. The algorithms in [3] are centralized and may not be applicable in the distributed, multi-agent context where agents are not always able to communicate with each other or with a central controlling device.

Because we cannot rely on a centralized unit to take care of all processing needs in a distributed, multi-agent system, new types of task allocation methods must be designed. Realistically, these methods must be able to deal with communication errors and other types of noise that could result from the agents obtaining erroneous input from their environment. MURDOCH [8] is a distributed, auction-based task allocation method designed for cooperative multi-robot teams in noisy environments. Agents utilizing MURDOCH use anonymous *publish/subscribe* broadcast communication to determine who is best for the task at hand. One known drawback of MURDOCH is that it is a greedy solution, i.e., it makes no predictions about the future, and thus, may assign tasks in non-optimal ways. In [8], the designers of MURDOCH state that "... from the perspective of the robots, which are model free, the tasks appear to be *randomly* generated ... Without a model of the world, planning provides no benefit."

Biologists have reported that insects can use past experience to create divisions of labor within the colony [2, 5, 17]. These studies show that agents can appropriately manage the ratio of *TypeA* to *TypeB* workers. For example, an ant that sees a large amount of food while foraging becomes more likely to forage in the future, whereas an ant that rarely came across food will have reduced its likelihood to forage again and instead will concentrate on other tasks such as cleaning the nest or tending to the brood. This self-adaptive, threshold idea has been used by roboticists to create a *forager/loafer* allocation scheme in a multi-robot foraging domain in order to help alleviate the robot-robot interference problem [11]. Lerman *et al.* [12] use mathematical models and experimental results to show that robots can use past experience to create an appropriate division of labor based on the proportion of *TypeA* to *TypeB* tasks in the environment. Our work investigates how agents can use past experience to make non-greedy decisions to improve the performance of the team when there is one type of task and its difficulty spans a continuous range.

3. THE PROBLEM

Before describing the multi-agent task allocation problem used throughout the remainder of this paper, we give the formal definition of the MULTIPROCESSOR SCHEDULING (MS) problem from which it is derived [7]:

> PROBLEM: MULTIPROCESSOR SCHEDULING
>
> INSTANCE: Set T of tasks, number $m \in Z^+$ of processors, length $l(t) \in Z^+$ for each $t \in T$, and a deadline $q \in Z^+$.
>
> QUESTION: Is there an m-processor schedule for T that meets the overall deadline q, i.e., a function $\sigma : T \rightarrow Z_0^+$ such that for all $u \geq 0$, the number of tasks $t \in T$ for which $\sigma(t) \leq u < \sigma(t) + l(t)$ is no more than m and such that, for all $t \in T$, $\sigma(t) + l(t) \leq q$?

MS, known to be *NP-complete* [3, 7], is asking for a mapping of tasks to processors such that no processor is computing two tasks at once and all of the tasks are completed before the deadline is reached. We introduce a simple extension to MS that takes into account situations where tasks are too difficult for some agents. This new problem will be referred to as CAPABILITY CONSTRAINED MULTI-AGENT TASK ALLOCATION (CCMATA).

> PROBLEM: CCMATA
>
> INSTANCE: Set of tasks T, set of agents A, difficulty $d(t) \in [0,1]$, length $l(t) \in Z^+$, maximum capability $maxC(a) \in [0,1]$ for each $a \in A$, and a deadline $q \in Z_0^+$.
>
> QUESTION: Do functions $\sigma : T \rightarrow Z_0^+$ and $\omega : T \rightarrow A$ exist, such that

1. $\forall t \in T$, $d(t) \leq maxC(\omega(t))$,
2. $\forall t_1, t_2 \in T$, $(\sigma(t_1) \leq \sigma(t_2) \ \wedge$
   $\sigma(t_1)+l(t_1) > \sigma(t_2)) \Rightarrow \omega(t_1) \neq \omega(t_2)$, and
3. $\forall t \in T$, $\sigma(t) + l(t) \leq q$?

Constraint 1 ensures that each task is completed by an agent that is capable of working on it, e.g., an agent that can lift fifty grams will not pick up rocks that weigh greater than fifty grams. Constraint 2 ensures that each agent is taking one task at a time, and Constraint 3 makes sure that the deadline is met.

CCMATA can model a wide range of task assignment and task scheduling problems. Assignment problems, where the goal is to assign at most one task to each agent so as to optimize some cost function, can be modeled by setting $q = 1$ and the lengths of all tasks to one. Traditional scheduling problems, where more than one task may need to be assigned to each agent, can be modeled by setting the maximum capability of each agent to one.

CCMATA does not provide a way to "hide" tasks from agents so that they are ignorant of what the future will bring. Because of the way decision problems like CCMATA and MS are constructed, an algorithm to solve an instance of these problems is aware of each task up-front and can create a schedule accordingly. However, in dynamic, uncertain environments, not all of the tasks are known *a priori*. To explore these types of *on-line problems*, a simulation based on the CCMATA problem is constructed.

4. EXPERIMENTAL SETUP

4.1 Simulation

A simulation based on CCMATA is used to understand how the performance of task allocation procedures is affected by team size, task length, and task difficulty distribution. The simulation returns the amount of time it took for the agents to complete all of the tasks. At each time step of the simulation, the agents not working on a task are presented with a randomly selected task. Depending on the task allocation procedure being used, the agents will either communicate with each other to decide who should take the task or will make individual decisions about whether to take or ignore the task. There will be times when all of the agents are busy working on their respective tasks and other times when the given task is too difficult for any available agent to complete. In both of these cases, the task is put back into the task pool and will be made available some time in the future. An agent is made unavailable to complete any other task while it is working on its current one.

The simulation is used to determine when it is best for agents to become selective about which tasks they take. It allows for the exploration and understanding of how the decision and learning episodes during each time step affect the performance of the task allocation procedures. The general, abstract design of the simulation allows us to focus on the behaviors of the task allocation procedures without having to worry about the details of designing navigation controllers for mobile agents or figuring out how the agents actually complete the tasks.

4.2 Agent capabilities

In the experiments, each of the agents, $a_1, a_2, ..., a_{|A|}$, is assigned a maximum capability equal to $maxC(a_i) = \frac{i}{|A|}$. This assignment gives an even distribution of capabilities, and because $maxC(a_{|A|}) = 1$, we are ensured that each task can be completed by at least one agent. This method of assigning agent capabilities was chosen over a more random approach in order to allow for a better analysis of the results.

To model an agent's tolerance to simple tasks, each agent is given a $minC$ (minimum capability) value. The $minC$ value of an agent will always be greater than or equal to zero and less than or equal to its $maxC$ value. An agent can take a task only when the difficulty of the task is greater than or equal to its current $minC$ value and less than or equal to its $maxC$ value. The details of how the initial $minC$ values are assigned and how they change over time are different for each task allocation procedure.

4.3 Task allocation procedures

Task allocation procedures should attempt to keep the agents busy most of the time in order to maximize parallelism. The task allocation procedures described below will be tested on various task difficulty distributions to determine when and why a certain procedure performs well or poorly.

4.3.1 GreedyTAP

GreedyTAP (Greedy Task Allocation Procedure) models the market-based task allocation procedures described in [9] and works by having agents communicate with each other in order to decide who is best fit for the current task. The agent whose $maxC$ value is closest to, but not less than, the difficulty of the task will take it.

GreedyTAP will always assign a task to an agent as long as one of the available agents can complete the given task, i.e., all agents have a $minC$ value of zero that never changes. In certain situations this may not be optimal as there are times when it is best for a capable agent to ignore easy tasks even when it is the only one currently available to complete the task.

4.3.2 StaticToleranceTAP

StaticToleranceTAP is a first step towards a strategy that attempts to determine when it is best for an agent to ignore easy tasks in the hopes that the near future will bring along tasks more appropriate for the agent. **StaticToleranceTAP** makes each agent responsible for a mutually exclusive range of task difficulties. For example, if we have three agents A, B, and C with $maxC$ values of $\frac{1}{3}$, $\frac{2}{3}$, and 1, respectively, then A's $minC$ value will be 0, B's $minC$ value will be $\frac{1}{3}$, and C's $minC$ value will be $\frac{2}{3}$. This will work well in situations where the task difficulty distribution is uniformly distributed because it will give each agent an equal share of tasks.

One drawback of **StaticToleranceTAP** is that it requires *a priori* knowledge of the capability of each agent in order to determine which range of tasks each agent should take. Also, if the task difficulty distribution is heavily weighted towards easy tasks, the agents with high $minC$ values will be underutilized because they will be ignoring most of the tasks. A method that allows for agents to adapt their tolerance to tasks of low difficulty is required to handle these situations.

4.3.3 DynamicToleranceTAP

DynamicToleranceTAP allows agents to adjust their tolerance to easy tasks based on the types of tasks they have seen in the past. This procedure makes for a much

more robust solution than **StaticToleranceTAP** because it allows agents to pick up the slack of their teammates that are unable to complete tasks or are overwhelmed with an abundance of tasks.

DynamicToleranceTAP can be implemented in many ways, but we chose a very simple implementation so that we can focus on the types of scenarios where it benefits to have a method like **DynamicToleranceTAP** and leave the job of finding different implementations as future work. Initially, the $minC$ value of each agent is set to equal its $maxC$ value. An agent lowers its $minC$ value by a predefined constant α when it sees a task with a difficulty below its current $minC$ value. An agent raises its $minC$ value by β after it accepts a task. (If $minC + \beta$ is greater than the difficulty of the task the agent just accepted, then $minC$ is only raised to equal the difficulty of the task.) This increasing tolerance for easy tasks can be seen as a simplified version of the *impatience* value in Parker's ALLIANCE architecture [16]. However, unlike ALLIANCE, the method presented here does not require agents to communicate with or observe each other.

This procedure allows agents to determine when to ignore easy tasks in order to utilize themselves in an efficient manner. If the distribution of task difficulties is skewed so that most of the tasks are easy, agents will repeatedly see easy tasks, and so, will lower their $minC$ values so they can become useful and take these tasks. However, when agents are finding that they can complete most of the tasks they see, their $minC$ values will stay fairly constant, and the agents will continue taking the tasks that are appropriate for them. These two dynamics should allow for **DynamicToleranceTAP** to perform relatively well, regardless of team size, task length, and task difficulty distribution.

4.4 Task difficulty distributions

The task difficulty distribution of a multi-agent task allocation problem refers to how the difficulty of each task is assigned. The three task difficulty distributions chosen for this paper are diverse enough to give a good indication of how the three task allocation procedures would perform in a range of situations.

4.4.1 Task Distribution 1

Task Distribution 1 (TD1) gives each task a difficulty chosen from a random, uniformly distributed number in the range of $[0, 1)$. This models unpredictable situations where agents have no idea of what types of tasks they will come across.

4.4.2 Task Distribution 2

With Task Distribution 2 (TD2), most of the tasks will have a low difficulty. Specifically, 40% of the tasks will have a randomly chosen difficulty in the range of $[0, \frac{1}{4})$, 30% in the range of $[\frac{1}{4}, \frac{1}{2})$, 20% in the range of $[\frac{1}{2}, \frac{3}{4})$, and the final 10% of the tasks will have a difficulty in the range of $[\frac{3}{4}, 1)$. This distribution allows us to determine whether **DynamicToleranceTAP** has any merit, since the agents with high $maxC$ values should reduce their tolerance to easy tasks in order to help out their teammates.

4.4.3 Task Distribution 3

Task Distribution 3 (TD3) assigns to half of the tasks a difficulty just above zero, while the other half have a difficulty slightly above 0.8. Any agent with a maximum capability at or below 0.8 will only be able to complete the easy half of the tasks, while the other agents can complete all tasks. An efficient task allocation procedure would assign an equal number of the easy tasks to the agents with a maximum capability at or below 0.8. The other half should be split evenly among the remaining agents.

5. EXPERIMENTS AND RESULTS

The experiments are used to examine the effects of team size, task length, and task difficulty distribution on the performance of the three task allocation procedures. For each run, the number of tasks is 1000, and to determine how team size affects performance, experiments will be run with both 10 and 50 agents. When the team size is 10 the length for all the tasks is either 2, 5, 10, 20, or 50, and when the team size is 50 the length is either 10, 25, 50, 100, or 250. For each combination of parameters, we record the amount of time it takes for the agents to complete the 1000 tasks. These times are then averaged over 1000 runs. When using **DynamicToleranceTAP**, α and β are set to equal $\frac{1}{|A|}$. Experiments not shown here indicate that α and β have large effects on performance and that $\frac{1}{|A|}$ seems to be a good value for both parameters. Because the goal of this paper is to determine when a method like **DynamicToleranceTAP** is useful, and not to design an optimal **DynamicToleranceTAP**-like procedure, the study of optimal α and β values is left as future work.

Table 1 shows the data collected from the experiments. Each pair of numbers represents the average time and standard deviation required for all 1000 tasks to be completed over 1000 runs. Table 1 is broken up into six sub-tables. Sub-Tables 1 and 2 show the data for Task Distribution 1 for team sizes 10 and 50, respectively. Sub-Tables 3 and 4 give the data for Task Distribution 2, and the remaining two sub-tables show the data for Task Distribution 3. The column headings show the task length (l) for the runs. The bold value in each column shows the best (lowest) average time for a task distribution and task length combination. Italicized numbers show the worst (largest) times.

To begin the analysis, we define r to be the ratio of task length to team size for any particular run. The results show that as r approaches 1, a change in the ordering of performance for the three task allocation procedures occurs. For example, for Task Distribution 1 and a team size of 10 (Sub-Table 1 in Table 1) **GreedyTAP** is more efficient than the other two methods when $r \leq 1$; however, when $r > 1$, **GreedyTAP** performs worse than the other two procedures.

If we compare the bolded and italicized numbers in the corresponding columns of team size 10 and team size 50 for a particular task difficulty distribution, we find that they almost always match. (The only instances where the orderings are different is when $r = 1$ for Task Difficulty Distribution 1 and 3.) This analysis indicates that the relative performances of the task allocation procedures are dependent on the ratio of task length to team size and not on the absolute values of team size or task length. Because the performance of a task allocation procedure is shown to be dependent on r, we focus our analysis on the results of one team size, namely team size 10.

Sub-Table 1	Task Distribution 1, Team Size 10				
	$l = 2$, $r = \frac{1}{5}$	$l = 5$, $r = \frac{1}{2}$	$l = 10$, $r = 1$	$l = 20$, $r = 2$	$l = 50$, $r = 5$
GreedyTAP	**1014.40 (4.36)**	**1161.22 (27.71)**	**1913.22 (66.22)**	*3791.42 (144.57)*	*9590.06 (362.54)*
StaticTolTAP	*1102.29 (9.56)*	*1408.91 (16.87)*	1935.91 (27.71)	**3031.01 (64.52)**	**6438.08 (197.20)**
DynamicTolTAP	1053.51 (7.42)	1275.66 (32.28)	*1959.79 (68.60)*	3556.38 (141.55)	8459.45 (358.01)

Sub-Table 2	Task Distribution 1, Team Size 50				
	$l = 10$, $r = \frac{1}{5}$	$l = 25$, $r = \frac{1}{2}$	$l = 50$, $r = 1$	$l = 100$, $r = 2$	$l = 250$, $r = 5$
GreedyTAP	**1018.25 (6.45)**	**1187.93 (72.33)**	2281.04 (179.34)	*4936.52 (373.55)*	*12605.87 (1000.70)*
StaticTolTAP	*1195.40 (13.66)*	*1552.40 (29.08)*	**2210.53 (70.57)**	**3663.10 (182.67)**	**8184.04 (520.26)**
DynamicTolTAP	1045.47 (15.08)	1364.00 (85.58)	*2370.42 (182.02)*	4745.61 (369.37)	11902.61 (944.43)

Sub-Table 3	Task Distribution 2, Team Size 10				
	$l = 2$, $r = \frac{1}{5}$	$l = 5$, $r = \frac{1}{2}$	$l = 10$, $r = 1$	$l = 20$, $r = 2$	$l = 50$, $r = 5$
GreedyTAP	**1003.68 (1.33)**	**1020.15 (5.31)**	**1409.28 (30.45)**	2713.64 (67.85)	6788.68 (171.45)
StaticTolTAP	*1122.68 (10.07)*	*1516.12 (19.83)*	*2259.30 (42.19)*	*3849.06 (94.23)*	*8808.73 (291.21)*
DynamicTolTAP	1034.98 (4.21)	1109.00 (12.16)	1503.61 (34.95)	**2637.85 (68.87)**	**6224.52 (172.84)**

Sub-Table 4	Task Distribution 2, Team Size 50				
	$l = 10$, $r = \frac{1}{5}$	$l = 25$, $r = \frac{1}{2}$	$l = 50$, $r = 1$	$l = 100$, $r = 2$	$l = 250$, $r = 5$
GreedyTAP	**1010.71 (1.08)**	**1030.15 (8.68)**	**1461.51 (91.49)**	3057.94 (201.19)	7668.53 (558.74)
StaticTolTAP	*1238.61 (15.17)*	*1715.13 (43.45)*	*2635.70 (104.80)*	*4653.69 (257.42)*	*10857.25 (716.05)*
DynamicTolTAP	1022.69 (4.25)	1091.91 (33.06)	1573.84 (99.72)	**3028.17 (214.52)**	**7439.18 (546.55)**

Sub-Table 5	Task Distribution 3, Team Size 10				
	$l = 2$, $r = \frac{1}{5}$	$l = 5$, $r = \frac{1}{2}$	$l = 10$, $r = 1$	$l = 20$, $r = 2$	$l = 50$, $r = 5$
GreedyTAP	**1002.00 (0.00)**	**1446.62 (10.21)**	2596.28 (10.74)	5182.60 (30.99)	13198.79 (94.69)
StaticTolTAP	*1500.53 (15.99)*	*3002.20 (20.20)*	*5504.87 (21.90)*	*10509.17 (23.21)*	*25512.14 (25.43)*
DynamicTolTAP	1003.96 (0.75)	1448.91 (9.68)	**2593.29 (8.95)**	**5059.23 (10.64)**	**12568.44 (49.00)**

Sub-Table 6	Task Distribution 3, Team Size 50				
	$l = 10$, $r = \frac{1}{5}$	$l = 25$, $r = \frac{1}{2}$	$l = 50$, $r = 1$	$l = 100$, $r = 2$	$l = 250$, $r = 5$
GreedyTAP	**1010.00 (0.00)**	**1346.53 (7.61)**	**2553.61 (5.18)**	5145.07 (36.42)	13150.31 (145.75)
StaticTolTAP	*5506.20 (22.58)*	*13008.68 (23.62)*	*25511.37 (25.53)*	*50511.87 (26.01)*	*125513.55 (25.53)*
DynamicTolTAP	1014.41 (1.17)	1349.85 (7.29)	2556.28 (5.40)	**5042.19 (4.91)**	**12539.15 (4.94)**

Table 1: The six sub-tables show the average performance of the three task allocation procedures for a particular task difficulty distribution, team size, and task length combination. The standard deviation of that average is shown in parenthesis. l is the length of the tasks, and r is the ratio of task length to team size. Bolded values indicate the lowest (best) average found for a particular task distribution and task length, and italicized values indicate the highest (worst) times out of the three task allocation procedures.

5.1 Analysis of TD1

Task Distribution 1 assigns task difficulties by generating a uniformly distributed, random number in the range of $[0, 1)$. Sub-Table 1 of Table 1 shows that **GreedyTAP** performs better than the other two task allocation procedures when $r \leq 1$, but when $r > 1$ **StaticToleranceTAP** performs best. Except for when $r = 1$, **DynamicToleranceTAP** never performs worse than both of the other two procedures, indicating that the adaptive procedure works well in the uniformly distributed case, regardless of the size of r.

Why does the change in performance for **GreedyTAP** and **StaticToleranceTAP** take place at $r = 1$? When a task is assigned to an agent, the agent has to wait the whole length of that task before it can take on another one. As l increases, the agent has to wait a longer time before it can take other tasks. While an agent works on its current task, more difficult tasks, which may be more appropriate for the agent, could come along. However, because the agent is busy working on its current task, these more appropriate ones must be put back into the task pool if no other capable agents are available. If the agent would have just ignored the easy task, it could have waited and taken on a task more fitting to its capability. To test this hypothesis, we count the number of times there were agents available for work but none of them chose to complete the current task because the task difficulty was not in any agent's $minC$ to $maxC$ range. If the hypothesis is true, then this count should be greater in the cases where the performance is worse. Figure 1 confirms this hypothesis, as the plots directly correlate to the results from the Sub-Table 1 in Table 1. When the team size is less than 10, **GreedyTAP** has agents available to complete the current task more often than **StaticToleranceTAP**, and so **GreedyTAP** performs better, but, when the team size is greater than 10, **StaticToleranceTAP** performs best.

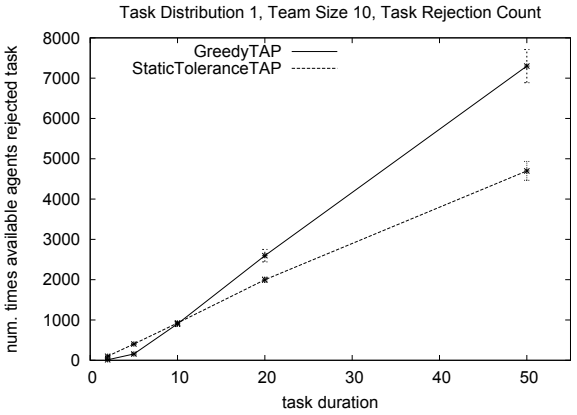

Figure 1: This plot shows the number of times there were agents available for work but none of them decided to take the current task. Each point shows the average of 1000 runs. Larger values indicate worse performance.

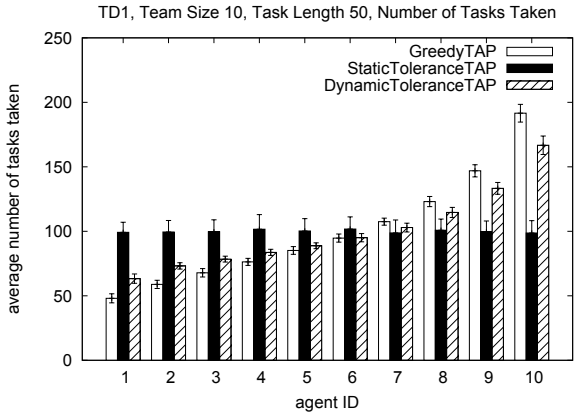

Figure 2: This graph shows the average number of tasks each agent was assigned for Task Distribution 1, team size 10, and length 50. The agents on the left are the least capable, whereas the agents on the right are the most capable ones. Results are averaged over 100 runs.

Another way to analyze the behaviors of the allocation procedures is to see how many tasks they allocate to each agent. Because each agent takes the same amount of time to complete each task, task allocation procedures should try to allocate the same number of tasks to each agent. Figure 2 shows the average number of tasks each of the ten agents took for the three task allocation procedures when the team size is 10 and $l = 50$. Recall that Agent 1 has a maximum capability of 0.1, Agent 2 of 0.2, Agent 3 of 0.3, etc. Figure 2 shows **StaticToleranceTAP** assigning an equal share (about 100) of tasks to each of the ten agents. On the other hand, **GreedyTAP** assigns about 4 times more tasks to the most capable agent (a_{10}) than the least capable one (a_1). **DynamicToleranceTAP** does a better job than **GreedyTAP** at evenly distributing the tasks, as it assigns about 2.5 times more tasks to a_{10} than a_1.

When tasks take a relatively short time to complete, it is best for the agents to take on tasks even if the difficulty of those tasks is well below their maximum capability. As the task length increases, agents should be more careful about which tasks they take in order to prevent themselves from being tied up with a simple task for a very long time.

5.2 Analysis of TD2

Task Distribution 2, which gives most of the tasks a low difficulty, allows us to determine if **DynamicToleranceTAP** can appropriately adapt the highly capable agents' tolerance to easy tasks so that they assist their teammates. The analysis of Task Distribution 1 showed that when task difficulties are uniformly distributed and r is large, **StaticToleranceTAP** performs best because each agent is responsible for an equal share of tasks. This is fine for the uniformly distributed case, but in situations where the task distribution is biased towards a small range of task difficulties, the performance of **StaticToleranceTAP** will greatly decrease because a small set of agents will be responsible for a large set of tasks and the other agents have no way of assisting their teammates.

Sub-Table 3 of Table 1 shows that when $r \leq 1$, **GreedyTAP** has the best performance, and **DynamicToleranceTAP** has the best performance when $r > 1$. Once again, the greedy approach works well as long as the task lengths are short, whereas a more appropriate task assignment is required to improve the performance when the task lengths are long. Figure 3 shows the average number of tasks each of the ten agents took for the three different task allocation procedures when $l = 50$. As expected, **StaticToleranceTAP** assigns most of the tasks to the agents with low $maxC$ values. Both **GreedyTAP** and **DynamicToleranceTAP** do a good job at evenly distributing tasks to the agents. The even distribution of tasks is the reason for **DynamicToleranceTAP**'s good performance and indicates that this method allows agents to adapt to situations where task difficulties are not evenly distributed.

5.3 Analysis of TD3

Task Distribution 3 is used to examine how well **DynamicToleranceTAP** can distribute tasks evenly among agents. For these experiments, 50% of the tasks have a difficulty just above zero, while the remaining 50% have a difficulty just above 0.8. In the case of 10 agents, only 2 of them (a_9 and a_{10}) can complete the more difficult tasks. An optimal task allocation strategy would assign half of the difficult tasks to one of those two agents and the other half to the second agent, while the tasks with low difficulty would be split evenly among the remaining 8 agents. Figure 4 shows that **DynamicToleranceTAP** and **GreedyTAP** are able to divide up the two sets of tasks evenly. **StaticToleranceTAP** is unable to divide up the tasks among the agents in these types of problem scenarios. **DynamicToleranceTAP** appropriately assigns about one half of the 0.8 tasks to each of the two most capable agents and assigns an even amount of the remaining 500 tasks to the other 8 agents. **GreedyTAP** assigns a slightly larger number of tasks to the two agents with the highest $maxC$ values.

With 10 agents, an optimal strategy takes around $\frac{|T|}{4} * l$ time steps to complete, where $|T|$ is the number of tasks, and

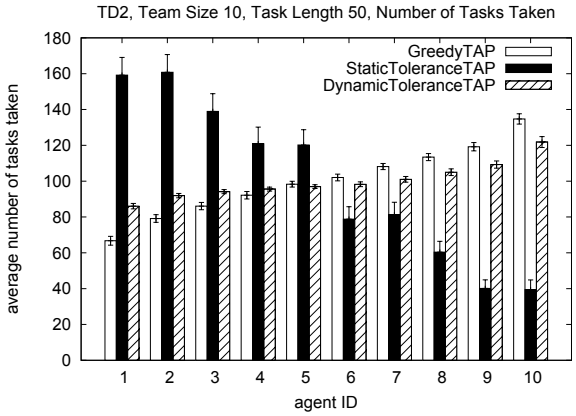

Figure 3: This graph shows the average number of tasks each agent was assigned for Task Distribution 2, team size 10, and length 50. The agents on the left are the least capable, whereas the agents on the right are the most capable ones. Results are averaged over 100 runs.

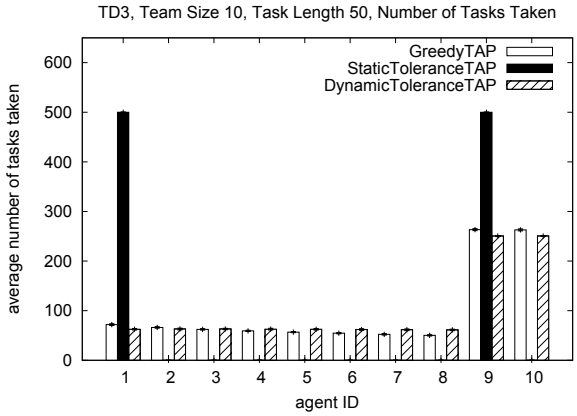

Figure 4: This graph shows the average number of tasks each agent was assigned for Task Distribution 3, team size 10, and length 50. The agents on the left are the least capable, whereas the agents on the right are the most capable ones. Results are averaged over 100 runs.

l is the task length. Since there are 1000 tasks, an optimal strategy should take around 12500 time steps to complete if $l = 50$. Sub-Table 5 shows that **DynamicToleranceTAP** takes about 12568 time steps to complete. This is very near what an optimal strategy could do. The results have shown that **DynamicToleranceTAP** is a viable solution to the task allocation problem, regardless of task difficulty distribution, team size, or task length.

6. CONCLUSION

This paper addresses an issue known to be inherent to greedy, multi-agent task allocation procedures [8]:

> ... from the perspective of the robots, which are model free, the tasks appear to be *randomly* generated ... Without a model of the world, planning provides no benefit.

Given a task and a set of available agents, a greedy solution will always assign the task to an agent, even when all of the available agents are "too fit" for the task. These approaches need a mechanism that allows agents to ignore, or put aside, tasks when they see them. By putting aside tasks that are not well suited for the available agents, a task allocation procedure is, in a trivial sense, planning for the future. It is shown here that, even when tasks are randomly generated, agents can build up simple models of their world that allow them to appropriately ignore tasks that they are not suited for. In situations where it takes a long time to complete tasks, this simple ignoring-task strategy is shown to reduce the amount of time required to complete all tasks when compared to greedy approaches.

An extension of the `MULTIPROCESSOR SCHEDULING` (`MS`) problem is used to model problems where agents have varying capabilities and tasks have varying difficulties and lengths. A simulation based on the new problem is designed to model real-world scenarios where agents are ignorant of the tasks that will be presented in the future. The comparison of the performances of three task allocation procedures on three different task difficulty distributions show that the ratio of task length to team size has a large impact on the performance of the team.

The first task allocation procedure models the greedy, market-based approaches used in cooperative multi-robot domains. This procedure assigns the current task to the agent that is found to be best-fit for the task and requires agents to be able to communicate their task-valuations with each other in order to come to a consensus about who should take the task. A key problem with this method is that it can end up assigning a very trivial task to a very capable agent, thus occupying the agent with a task it probably should not be working on. This inefficient use of resources is shown to cause a decrease in performance when the amount of time it takes to complete a task is large relative to team size.

To remedy this problem, a task allocation procedure that never assigns simple tasks to very capable agents is examined. With this method, agents are assigned mutually exclusive ranges of tasks that they can complete. In order for this method to be efficient, the assignment of ranges to the agents must correlate with the distribution of task difficulties. If the latter is not known, the system designer cannot assign the former to guarantee optimal performance. A more significant problem is that because there is no redundancy in this method, it will completely break down if an agent dies off and can no longer complete its tasks. A more robust solution is required.

An adaptive, biologically inspired task allocation procedure is examined to determine when agents can act in a non-greedy manner to improve the overall efficiency of the team. An agent's tolerance to easy tasks is a function of the difficulty of the tasks it has seen in the past. If an agent sees a large number of very simple tasks, it can adjust its tolerance so that it can make it self more useful by taking easy tasks. Agents use their environment (i.e., the tasks they see) as a way to determine how it should spend its time. The adaptive method performs better than the greedy and static-tolerance methods when the ratio of task length to team size is large and when the task difficulties are not uniformly distributed. Unlike the greedy method, the adaptive one

requires no explicit communication between the agents, and so, it scales well to teams of any size. Another advantage of this method is that it is robust to changes in team composition, i.e., when agents break down or when their maximum capabilities change, the team members can adjust their tolerances accordingly.

7. FUTURE WORK

The main factor determining whether or not a non-greedy method should be used is the ratio of task length to team size, and with the current system there is no way for agents to know this ratio. Future work will look to design task allocation methods where agents learn this ratio and adapt their task allocation strategy for the problem at hand.

How will the performance of the task allocation procedures be affected when only a subset of the agents can see each task? So far, it has been assumed that each task is seen by every available agent; however, in spatially distributed problems physical limitations prevent agents from viewing each task.

Finally, it may be possible to use the methods presented here in combination with current multi-robot task allocation architectures in order to produce more efficient solutions than were possible with the greedy methods alone. It is not yet known how the adaptive method could benefit from interagent communication.

8. 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.

9. REFERENCES

[1] D. P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1:7–66, December 1992.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Santa Fe Institute Studies in the Sciences of Complexity, New York, NY, 1999.

[3] H. El-Rewini, H. H. Ali, and T. Lewis. Task scheduling in multiprocessing systems. *Computer*, 28(12):27–37, 1995.

[4] U. Faigle. Some recent results in the analysis of greedy algorithms for assignment problems. *OR Spectrum*, 15(4):181–188, December 1994.

[5] 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.

[6] D. Gale. *The Theory of Linear Economic Models*. McGraw-Hill Book Company, Inc., New York, 1960.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[8] B. P. Gerkey and M. J. Matarić. Sold!: Market methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, October 2002.

[9] 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.

[10] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1):83–97, 1955.

[11] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Efficiency and task allocation in prey retrieval. In *Proceedings of the 1st International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pages 274–289, 2004.

[12] K. Lerman, C. Jones, A. Galstyan, and M. J. Mataríc. Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Research*, 25(3):225–241, 2006.

[13] 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.

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

[15] E. H. Østergaard, M. J. Matarić, and G. S. Sukhatme. Distributed multi-robot task allocation for emergency handling. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Wailea, Hawaii, October 2001.

[16] L. E. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[17] F. Ravary, E. Lecoutey, G. Kaminski, N. Châline, and P. Jaisson. Individual experience alone can generate lasting division of labor in ants. *Current Biology*, 17:1308–1312, 2007.

[18] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *AAAI*, pages 1752–. AAAI Press, 2007.