Tasks with Cost Growing over Time and Agent Reallocation Delays

James Parker
Computer Science and Engineering
University of Minnesota
200 Union St SE, Minneapolis, MN 55455, USA
jparker@cs.umn.edu

Maria Gini
Computer Science and Engineering
University of Minnesota
200 Union St SE, Minneapolis, MN 55455, USA
gini@cs.umn.edu

ABSTRACT

To efficiently complete tasks whose completion costs change predictably over time requires agents that can take into account these changes. When there are more agents than tasks the problem is how to coordinate the allocation of agents to prevent tasks from growing so much that they become unsolvable. This work focuses on a subset of cost functions for modeling tasks whose cost grows over time and provides an optimal solution when agents can be allocated to tasks instantly. We present both the Latest Finishing First (LFF) algorithm, which is suitable when the cost of reallocating agents is high, and the Real-Time Latest Finishing First (RT-LFF) algorithm, which adapts to agent reallocation delays and new tasks appearing. These algorithms are compared against the optimal zero travel time solution with varying delays in reallocation in a simple environment. We then show how to apply this model to the complex problem of allocating agents to extinguish fires in the RoboCup Rescue simulator and show how RT-LFF solves the problem efficiently.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

Keywords

Multi-robot systems, task allocation, coordination

1. INTRODUCTION

Multi-agent systems are getting used in many areas, such as wide-area surveillance, exploration and mapping, transportation, and search and rescue. The use of multiple agents provides greater robustness, flexibility and efficiency over single-agent systems, but requires coordination between the agents to be successful.

Our work focuses on groups of identical homogeneous agents that need to be efficiently assigned to tasks. We introduce a general model for the interaction between agents and tasks, where the cost of tasks grows over time. This type of problem can occur in nature, such as invasive species or forest fires, where if a task is not completed quickly, then it can become difficult or impossible to complete later. If too few agents are assigned to these growing tasks, they will not be able to counteract the growth and the cost of the task will grow towards infinity. This means that inefficient assignments could not even produce a feasible solution for all tasks.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

We first discuss related work (Section 2), followed by a formal description of our problem and notation (Section 3). We give the optimal solution (Section 4) assuming that agents can be instantly assigned to any task. Zero travel time is a strong assumption when agents need to physically move to a task, so we relax this constraint and provide the *Latest Finishing First (LFF)* algorithm, which is resistant to variations in the travel time (Section 5). We then extend LFF to a real-time version, *Real Time Latest Finishing First (RT-LFF)*, which accounts for changes in the number of agents and tasks. (Section 6). A comparison between our algorithms, the optimal and some baseline metrics is given in a simple simulator (Section 7). We then present a practical application of RT-LFF in RoboCup Rescue and describe the modeling involved (Section 8), and evaluate experimentally our work against other methods (Section 9). Our contributions are then summarized (Section 10).

2. RELATED WORK

Multi-agent task allocation is a well known NP-hard problem, giving rise to many different approximate solutions. Two methods for task allocation have emerged as the main paradigms: threshold and auction based methods. In threshold methods, agents individually assess the constraints and their ability to complete each task. If an agent's abilities surpass a threshold on the constraints, then the agent assigns itself to the task. If not, the agent passes the information to other agents. An example is [8], which uses distributed constraint optimization (DCOP) as a basis for task allocation. A comparison between DCOP and swarm techniques is provided in [3]. On the other hand, market inspired auction methods typically require more communication and are more centralized. Zhang et al. [9] present an auction based approach to form executable coalitions, allowing multiple agents from different locations to reach a task and compete it efficiently. Recently, decentralized applications have been designed that add flexibility to the system (e.g., [5]). Our work strikes a balance between distribution and centralization, each agent is directed to an area by a central authority, but upon reaching the destination, agents act on their own logic.

Other approaches have been developed, such as modeling task allocation as a potential game [2]. Sandholm et al. [7] present a generalized coalition formation algorithm which produces solutions within a bound from the optimal via pruning. The work in [10] focuses on tasks that require multiple agents to complete, while simultaneously trying to efficiently use the agent's resources and time. Our approach also assumes multiple agents are required, but we allow the requirements of tasks to change over time.

Our work is most similar to Ramchurn et al. [6], except we reformulate the problem so task resources change over time. Instead of having deadlines for tasks that expire at specific times, we can consider each task having a minimum agent deadline which means a specific amount of agents must be assigned to the task by this time in order for the task to be doable. Urban search and rescue is a major focus of our work and we use the RoboCup Search and Rescue Simulator [4], which provides simulations on street and building maps of real cities. Emergency situations are very time critical and often lacking in information, as outlined in [1]. Most notably, when an emergency occurs agents are spatially spread out and must quickly coordinate with each other to accomplish tasks.

3. PROBLEM DESCRIPTION

Our problem focuses on the assignment of identical homogeneous agents to tasks which have a cost that changes over time. We make no assumptions on the spatial locations of agents or tasks other than an agent must be on a task's location in order to apply work. In order for our methods to be effective, there should be more agents than tasks since we assume multiple agents must be assigned to a task. This assumption is not too restrictive. As we show later in Section 8, one can define a task as a cluster of smaller subtasks in order for this property to hold.

We denote the set of identical homogeneous agents by $A = \{a_1, \ldots, a_{|A|}\}$ and the set of tasks by $B = \{b_1, \ldots, b_{|B|}\}$. For our problem, we assume $|A| > |B|$ or that B represents clusters of smaller subtasks to make this inequality true. The set of active agent assignments is denoted by $N^t = \{n_1^t, \ldots, n_{|B|}^t\}$, where n_i^t is the set of agents from A that are currently working on task b_i at time unit t. An agent can only work on one task at a time, so $n_i^t \subseteq A$ and $\forall i \neq j, n_i^t \cap n_j^t = \emptyset$. All agents and tasks have a spatial location in the environment and the travel time, $TT(x, y)$, between two locations, x and y, is assumed to be computable.

Each identical agent provides the amount of work w per time unit if the agent has finished traveling to that task. Every task $b_i \in B$ has a cost defined with the following recursive relationship:

$$f_i^{t+1} = f_i^t + \Delta f_i^t, \tag{1}$$

where Δf_i^t has the form:

$$\Delta f_i^t = h_i(f_i^t) - w \times |n_i^t|, \tag{2}$$

where f_i^t starts at some initial cost f_i^0 and $h_i : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ is a monotonically increasing function. Here we treat f_i^t as a sequence due to discrete time steps, but it could be treated as a continuous function if that is a better model for the domain.

If at some time t the cost of task b_i, namely f_i^t, reaches or passes zero, we denote this time as ct_i for the completion time meaning that at this point the task is complete. For this reason, when f_i^t is non-positive, $h_i(f_i^t)$ is assumed to be zero and we do not allow agents to be assigned to this task, namely $|n_i^t| = 0$ when $t > ct_i$. When $h_i(f_i^t) > w \times |n_i^t|$ this means f_i^t is strictly monotonically increasing, which means the task is growing faster than the assigned agents can reduce it. If this is true, the task will never be completed and we say $ct_i = \infty$.

A solution is found at time t_s when the last task is completed, if and only if all $b_i \in B$ have $f_i^{t_s} = 0$. Using (2), we can write this solution as $f_i^0 + \sum_{t<t_s} \Delta f_i^t = 0$.[1] Since this is true for every $b_i \in B$ a solution is reached if and only if:

$$\sum_{b_i \in B} \left(f_i^0 + \sum_{t<t_s} \left(h_i(f_i^t) - w \times |n_i^t| \right) \right) = 0 \tag{3}$$

[1]Technically, this solution will be reached when f_i^t goes from positive to negative in f_i^{t+1} since time is discrete. For theoretical ease we assume t is divisible enough that it is possible to actually reach zero.

4. OPTIMAL SOLUTION

In our problem the cost of completing a task increases over time, therefore the goal is to minimize the time the last task is completed, specifically minimizing t_s. Finding the general optimal solution is very difficult, but we prove the optimal solution can be found by making the following assumptions:

1. there is a positive correlation between f_i^t and $h_i(f_i^t)$. Namely, larger tasks must grow faster than smaller tasks. This is reasonable for domains that have a chain-reaction effect, such as a stampede. If one animal gets frightened and starts running, other animals also get frightened and start running. As more animals start running, f_i^t, the rate that additional animals start to run, $h_i(f_i^t)$, also increases. Fires are another example.
2. as a task gets closer to completing, its growth approaches zero. This is needed for continuity since after a task is complete, we assume it cannot grow and thus $h_i(x) = 0$ for non-positive x.
3. the task growth functions, $h_i(f_i^t)$, are the same for all tasks. Tasks can have different initial costs, and thus different initial growth rates.
4. the travel time between any task locations is zero. In some applications that do not require physical movement this assumption would be correct, but this is rarely the case for physical agents. Even with zero travel time the growth of the tasks makes assignment of agents non-trivial.

For example, consider one agent and two tasks b_1 and b_2 such that $f_1^0 << f_2^0$ but $h_1(f_1^t) >> h_2(f_2^t)$. This means one task starts with a small initial cost but grows very rapidly and the other task has a large initial cost but grows very slowly. The best solution is to quickly finish b_1 to prevent it from growing rapidly and then work at b_2. It takes some time to complete b_1 but the change in b_2 is not very significant, so there is not much of a difference between completing b_2 first or second. However, if one tries to first complete b_2, then b_1 will have grown considerably by the time b_2 is completed and will take much longer to complete b_1 second rather than first.

For ease of the proof, we will define some additional notation. Note that in (3), $\sum_{b_i \in B} f_i^0$ is a constant and $\sum_{b_i \in B} \sum_{t<t_s} w \times |n_i^t| = \bar{p} \times t_s \times w \times |A|$, where $\bar{p}$ is the average percent of time the agents are working. We can then rewrite (3) as:

$$\sum_{b_i \in B} f_i^0 + \sum_{b_i \in B} \sum_{t<t_s} h_i(f_i^t) - \bar{p} \times t_s \times w \times |A| = 0 \tag{4}$$

This means that $\sum_{b_i \in B} \sum_{t<t_s} h_i(f_i^t)$ is the amount of work added to the system from time $t = 0$, which we call R_{t_s} or the global regret. We define $\Delta R_t = R_t - R_{t-1} = \sum_{b_i \in B} h_i(f_i^t)$.

The proofs presented below have the following outline. First we show that minimizing R_{t_s} is an optimal solution with the zero travel time assumption, because we can assume all agents will always be assigned and working on some task. Then the only term left in (4) that depends on the assignments is the middle term, R_{t_s}, namely how the cost function changes based on the assigned agents. From here it is fairly straightforward to see that by reducing R_{t_s}, t_s on the right in (4) can be reduced hence reaching a solution faster. Then we show this middle term, R_{t_s}, can be greedily minimized at each time step. Thus in order to minimize the total growth accumulated, it is sufficient to assign agents to minimize the cost growth of all tasks at every time step.

THEOREM 1. *Minimizing R_{t_s} is an optimal solution when $TT(x, y) = 0 \, \forall x, y$.*

PROOF. When $TT(x, y) = 0 \, \forall x, y$ we can assume $\bar{p} = 1$, since agents can instantly move between tasks. Suppose a better solution $\hat{f}_i^t$ exists, then $\hat{R}_{t_{\hat{s}}} = \sum_{b_i \in B} \sum_{t < \hat{t_s}} h_i(\hat{f}_i^t)$ where $\hat{t_s} < t_s$ with $\hat{R}_{\hat{t_s}} > R_{t_s}$. Rewriting (4) for R_{t_s} yields:

$$R_{t_s} = w \times |A| \times t_s - \sum_{b_i \in B} f_i^0$$

Solving (4) for $\hat{R}_{\hat{t_s}}$ in terms of t_s:

$$\hat{R}_{\hat{t_s}} + w \times |A| \times (t_s - \hat{t_s}) = w \times |A| \times t_s - \sum_{b_i \in B} f_i^0$$

Using our two assumptions, we can then write:

$$R_{t_s} + w \times |A| \times (t_s - \hat{t_s}) < \hat{R}_{\hat{t_s}} + w \times |A| \times (t_s - \hat{t_s})$$

which implies that R_{t_s} satisfied (3) at time $\hat{t_s}$, a contradiction. □

THEOREM 2. *Minimizing ΔR_t for every time unit t also minimizes the global regret, R_{t_s}, when (i) $TT(x,y) = 0 \; \forall x, y$, (ii) $h_i(x) = h_j(x) \; \forall b_i, b_j \in B$, (iii) $\lim_{x \to 0^+} h_i(x) = 0$, and (iv) $\frac{\partial^2}{\partial t^2} h_i(f_i^t) \geq 0$.*

PROOF. Due to assumption (ii), we will denote $h_i(x)$ with $h(x)$ to simplify notation. Assume there exists a better solution $\hat{F}$, which differs from the greedy minimization F. This means that at some time t_d the better solution must assign agents differently than the greedy solution. By definition the greedy minimization, F, is a minimum ΔR_{t_d} at time t_d, thus $\Delta R_{t_d} \leq \Delta \hat{R}_{t_d}$, where $\Delta \hat{R}_t = \sum_{b_i \in B} h(\hat{f}_i^t)$. After time t_d, F will copy the agent assignments of $\hat{F}$ and because $TT(x, y) = 0 \; \forall x, y$ this is possible from any configuration. Next we prove by induction that $\sum_{b_i \in B} f_i^t \leq \sum_{b_i \in B} \hat{f}_i^t$. At time t_d, $\sum_{b_i \in B} f_i^{t_d} = \sum_{b_i \in B} \hat{f}_i^{t_d}$ combined with the fact that $f_i^x = f_i^0 + \sum_{t<x} \Delta f_i^x$ along with F is a greedy choice implies $\sum_{b_i \in B} f_i^{t_d+1} \leq \sum_{b_i \in B} \hat{f}_i^{t_d+1}$, which is the base case in the induction. If we write $\hat{f}_i^t = f_i^t + c_i$, then we can conclude $\sum_{b_i \in B} c_i \geq 0$. We then compute f_i^{t+1} as:

$$f_i^{t+1} = f_i^t + h(f_i) - w \times |n_i^t|$$

and $\hat{f}_i^{t+1}$ as (n_i^t is the same since assignments are copied):

$$\hat{f}_i^{t+1} = (f_i^t + c_i) + h(f_i + c_i) - w \times |n_i^t|$$

If we use the monotonicity of h, then we can see $h(f_i + c_i) = h(f_i) + \delta_i \times c_i$ for some $\delta_i > 0$, basically δ_i is the slope between f_i and $f_i + c_i$. This means:

$$\hat{f}_i^{t+1} - f_i^{t+1} = c_i + \delta_i \times c_i$$

Since $\frac{\partial^2}{\partial t^2} h_i(f_i^t) \geq 0$ we know $\delta_i \geq \delta_j$ when $c_i > c_j \; \forall i, j$ as shown in Figure 1. Thus, $\sum_{b_i \in B} \delta_i \times c_i \geq 0$ since $\sum_{b_i \in B} c_i \geq 0$ as positive c_i get a larger δ_i than negative values and we can conclude that $\sum_{b_i \in B} f_i^{t+1} \leq \sum_{b_i \in B} \hat{f}_i^{t+1}$, the inductive step. Using a similar logic to extracting the definition of ΔR_t from (3), we drop $\sum_{b_i \in B} f_i^0$ and $\sum_{t<t_s} \sum_{b_i \in B} w \times |n_i^t|$ from both sides and get:

$$R_{t_s} \leq \hat{R}_{t_s}$$

where $\hat{R}_{t_s} = \sum_{b_i \in B} \sum_{t<t_s} h(\hat{f}_i^t)$. This is a contradiction to the fact that $\hat{F}$ is a better solution, therefore minimizing ΔR_t at every time t also minimizes R_{t_s}.

Before concluding the proof, we must consider the cases when F completes a task that $\hat{F}$ did not and vice versa. If F completes a task that $\hat{F}$ did not, then the greedy minimization will assign agents from an already completed task to a random unfinished tasks. This does not invalidate any of the inequalities above. When $\hat{F}$ completes a task that F has not, this task will never be completed by direct mimicry from the greedy minimization solution, instead this will be finished by the random assignment described above. There is no discontinuity in the sums since we require $\lim_{x \to 0^+} h(x) = 0$, so when a task is completed it simply disappears from the equations. □

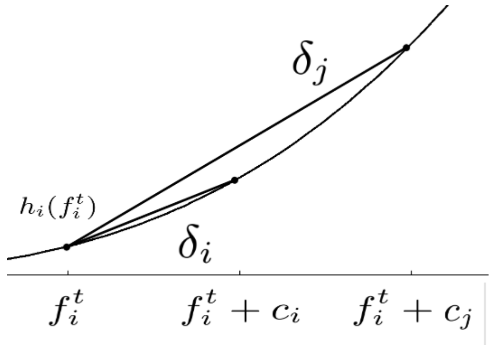

Figure 1: Since the function $h(f_i^t)$ is always accelerating, the slope, δ_i, will be larger when c_i is larger.

5. LATEST FINISHING FIRST

With perfect knowledge about how a task grows, we showed what assignment is optimal. Taking into account the spatial limitations, we derive the Latest Finishing First (LFF) algorithm, which uses a heuristic to maintain a stable assignment.

The optimal solution sometimes frequently reassigns agents because $TT(x, y) = 0$. When agents require time to move between different tasks, every time an agent is reassigned to a different task, $\bar{p}$ decreases in (4). To address this issue, we introduce Latest Finishing First (LFF). The inspiration behind LFF is to create an initial stable assignment that will try to maximize $\bar{p}$ to 1, thus maximizing the overall output of agents in the system. Although this maximizes the right side of (4), there is no guarantee of the effect on the middle term, R_{t_s}. In other words, we will fully utilize all the agents but may inefficiently assign them.

The LFF algorithm is based on the heuristic of iteratively assigning agents to the task which finishes last, or specifically has the largest completion time, ct_i. This causes LFF to have two main phases, the first when few agents are assigned and multiple tasks have $ct_i = \infty$ since the growth rate of the tasks is greater than the reduction from assigned agents. At this stage there is often a tie for tasks with the largest ct_i, so in order to break the tie an agent is assigned to whichever task has a larger initial cost, namely f_i^0.[2] The agent that is actually assigned to this task is selected greedily, simply the closest unassigned agent to that task. Once this agent is assigned, ct_i is updated for this task and this process is repeated until there are no more tasks with $ct_i = \infty$. If at some point there are no more unassigned agents and there is still a task with $ct_i = \infty$, then the algorithm is done but some tasks are not completed.

The second phase, if reached, starts when all tasks have a finite completion time, specifically $ct_i < \infty$ for all tasks. Ties for a maximum ct_i are rare if the time scale of t is sufficiently small, unless there are two identical tasks with similar unassigned agent

[2]If there is a tie for tasks with initial costs, then assign an agent to minimize that agent's travel time to reach either of these two tasks.

locations. If there is a tie, the initial cost can be used to determine where to assign the agent, similar to the first phase. When the task is determined, the closest unassigned agent is assigned to that task and the task's ct_i is updated. This process is repeated until there are no more unassigned agents left and the LFF algorithm finishes.

During this first phase the choice of using the initial cost as a tie breaker is semi-arbitrary, since no assumptions are made about $h_i(f_i^t)$. However, if there is some correlation between the task growth functions, then a task with a higher initial cost will take longer to complete than a smaller task if the number of agents assigned is the same. This means tasks with a large f_i^0 will likely need more agents assigned, thus this tie breaking attempts to reduce overall agent travel time.

Algorithm LFF (Latest Finishing First)

```
1: while $\exists a_i \in A$ and $\nexists j$ such that $a_i \in n_j^{TT(a_i,b_j)}$ do {Loop
   while there are unassigned agents.}
2:   Find $b_i$ for $\underset{b_i \in B}{\operatorname{argmax}}\ ct_i$ {Task $b_i$ ends last}
3:   if $\exists b_j$ where $ct_i = ct_j$ then {This is mostly phase one when
     there are multiple tasks that never finish.}
4:     Find $b_k$ for $\underset{b_k \in B}{\operatorname{argmax}}\ f_k^0$ with $ct_k = ct_i$ {In this case, we
       find the task with the largest initial cost.}
5:     Assign $\underset{a_j \in A \text{ and } a_j \text{ unassigned}}{\operatorname{argmin}} TT(a_j, b_k)$ to $b_k$
6:   else
7:     Assign $\underset{a_j \in A \text{ and } a_j \text{ unassigned}}{\operatorname{argmin}} TT(a_j, b_i)$ to $b_i$ {If there are
       no ties simply assign the closest agent to $b_i$.}
8:   end if
9: end while
```

LFF only provides an initial assignment because the algorithm ends when all agents are assigned, thus it is well suited when reassignment is difficult due to high travel cost or lack of communication. For example, if forest firefighters deploy deep into the wilderness via parachute, they will be unable to move very effectively after their initial deployment. In the case of one shot deployment with a uniform initial assignment cost, LFF provides the optimal solution. In the next section we consider the case when LFF is rerun when $t \neq 0$, which is simply accomplished by replacing n_i^0 and f_i^0 with n_i^t and f_i^t respectively.

6. REAL-TIME LATEST FINISHING FIRST

This section extends the LFF algorithm to a real-time solution applicable to a wider range of environments. The LFF algorithm only does the initial assignment, but after agents are assigned and the cost of a task is discovered to be different from the initial estimate, it can be advantageous to reassign some agents. LFF assumes perfect information of all the tasks but the initial task cost can be incorrect or the function can be an imperfect approximation. Also, when new tasks are discovered, it is crucial that agents are reassigned to the new tasks. The Real-Time Latest Finishing First (RT-LFF) algorithm can adjust to all these situations while attempting to maintain the same property as the original LFF algorithm, namely by trying to assign agents to tasks with high completion times.

For the initial agent assignment RT-LFF simply uses LFF. Every time period after the initial, RT-LFF attempts to find if there is any pair of tasks that could reassign an agent from one task to another to reduce the latest completion time of all tasks. An agent is only transfered from an original task to a task that finishes later if the original task will still complete faster, even after transferring the agent. Let ct_i^- be the time for task b_i to complete with one fewer agent and ct_j^+ be the time for task b_j to complete with one additional agent including the delay from the agent traveling. An agent is only reassigned from b_i to b_j if $ct_i^- < ct_j^+$. Note that this assignment strategy is greedy since the original task will not give up an agent unless it is still better off than the receiving task. Here we provide a formal definition of the RT-LFF algorithm.

Algorithm RT-LFF (Real-Time Latest Finishing First)

```
1: Use $LFF$ for initial assignments
2: while $t < t_s$ do
3:   for $b_i \in B$ do
4:     if $\exists b_j \neq b_i$ where $ct_i^- < ct_j^+$ with $\underset{b_i}{\operatorname{argmin}}\ TT(b_i, b_j)$
       then
5:       Reassign $a_k \in N_i^t$ to $b_j$ with $\underset{a_k}{\operatorname{argmin}}\ TT(a_k, b_j)$
6:     end if
7:   end for
8: end while
```

Another transfer criterion worth considering is when $\max(ct_i, ct_j) \geq \max(ct_i^-, ct_j^+)$. This causes RT-LFF to have a very similar assignment to LFF, if it was rerun at that time step, and can cause assignment thrashing, especially when there is noise or an error in the growth function. For this reason LFF is used for initial allocations only. The task b_j would receive benefits from the extra agent, but the task b_i is now behind and might possibly need an agent transferred back in the future. When the heuristic is greedy, namely only transfer if $ct_i^- \leq ct_j^+$, then after a transfer the completion times of both tasks will be approximately equal, resulting in less likely future transfers between these two tasks.

Whenever possible, all pairs of tasks are considered and the order that the pairs are considered can have a significant effect. If one task is first paired with all other tasks, it will have priority to receive (or send) agents. Thus, we put newly observed tasks first, then order the rest of the remaining tasks based on how close they are to the new task. This ordering causes the new task and closest task to potentially transfer agents first, then the new task and second closest task and so forth. This is desirable because it reduces the travel time of agents assigned to the new task, and since the new task has no agents assigned, it will likely receive multiple. If the domain has a large number of agents or fairly indivisible time steps, it might be useful to transfer multiple agents per pairing or check pairs multiple times. If a domain has many tasks that are almost collinear, it might be best to check all pairs of direct neighbors first before checking pairs that have other tasks between them.

We could rerun LFF at every time step instead of using RT-LFF, but one disadvantage is when there are many more agents than tasks, attempting to reassign every agent can be more expensive than checking pairs of tasks. While the greedy criterion of RT-LFF can cause some inefficiency, we show that the difference between agent assignment in RT-LFF and rerunning LFF at that time step is bounded if travel time is zero. Since LFF attempts to maximize the amount of work each agent provides, this proof shows that agents in RT-LFF must also be near the maximum amount of work.

THEOREM 3. *RT-LFF has at most $|B| - 1$ agents differently assigned from the LFF algorithm rerun at any time assuming zero travel time.*

PROOF. We show that RT-LFF can at most have one fewer agent assigned to each task compared to LFF. This is done by showing it is impossible for RT-LFF to assign two more agents to any task and reduce the overall finishing time, t_s, which LFF minimizes. Assume that we have two tasks b_1 and b_2 and let ct_j^{++} be the time task b_j is completed with two additional agents assigned to it and similarly ct_i^{--} for two fewer agents. Without loss of generality assume $ct_2 < ct_1$. If RT-LFF did not reassign an agent from b_2 to b_1, then $ct_2^- > ct_1^+$. We notice that reassigning two agents from b_2 to b_1 means $ct_2^{--} > ct_2^-$ and $ct_1^{++} < ct_1^+$, but $\max(ct_2^{--}, ct_1^{++}) > \max(ct_2^-, ct_1^+)$ because $ct_2^{--} > ct_2^-$ and $ct_2^- > ct_1^+$. This means t_s is increased by this reassignment, which means this is not a possible LFF assignment.

Next we show that when using RT-LFF a task b_1 would never receive a single agent from more than one other task without increasing t_s. The argument is similar to the first case, only it now involves task b_3. If b_2 and b_3 did not reassign an agent to b_1 under RT-LFF, then $ct_1^+ < ct_2^-$ and $ct_1^+ < ct_3^-$. Suppose LFF did have both b_2 and b_3 reassign an agent to b_1, and without a loss of generality assume $ct_2^- < ct_3^-$. Then $ct_1^{++} < ct_1^+ < ct_2^- < ct_3^-$. This is a contradiction to how LFF works. Currently ct_3^- is the last finishing and ct_1^{++} is the fastest finishing, and from the equation above if b_1 gave back an agent to b_3 then $ct_1^+ < ct_2^-$. This means the transfer back has reduced the time of latest finishing task, thus under RT-LFF b_1 will be not assigned an agent from more than one other task compared to LFF. This implies a task in RT-LFF cannot have more than one agent under the assignment of LFF for each time step. The worst case is when $|B| - 1$ tasks have one agent fewer than LFF's assignment and the last task has $|B| - 1$ too many agents. □

7. COMPARISON IN SIMPLE SIMULATION

To empirically test our algorithms, a simplified simulation was used that has task b_1 on one end of a line segment task b_2 on the other. For all the simulations the number of agents remained constant, so we will abbreviate $|N^t|$ by $|N|$. In addition to the optimal solution, we compared LFF and RT-LFF to a "UNIFORM" and "ALLONONE" baselines. The UNIFORM strategy simply assigns $|N|/2$ agents to b_1 and the other half of the agents to b_2 at $t = 0$ for the whole simulation. The ALLONONE strategy assigns N to task b_1 until it is completed, then assigns all the agents to b_2.

Table 1 compares the completion time of the last task for three different task cost growth functions with no travel time and 10 agents. The $h(x)$ column shows the growth function without coefficients for b_1 first then b_2. For example, the first row is when $h_1(x) = h_2(x) = 0.000016 \times x^3$ with each agent doing an amount of $w = 0.015$ work and initial task costs of $f_1^0 = 20$ and $f_2^0 = 10$. Coefficients for the growth functions and work rates are small to shrink the effects of time, thus simulating granular time steps.

$h(x)$	OPTIMAL	RT-LFF	LFF	UNIFORM	ALLONONE
x^3, x^3	310	440	∞	∞	343
x^2, x^2	260	270	273	626	∞
x^3, x^2	94[3]	118	128	128	∞

Table 1: Completion times of the last task to finish with no travel times and $|N| = 10$.

Figure 2 shows the assignments when solving with the same parameters given above, except $w = 0.0015$, and there are 100 agents. When an algorithm completes all tasks, the marks in the figure stop, except the UNIFORM strategy which never completes task b_1. The optimal strategy is to first assign all agents to b_1

[3]This is not guaranteed to be optimal since $h(x)$ is not identical for all tasks, but does still give the best solution we found.

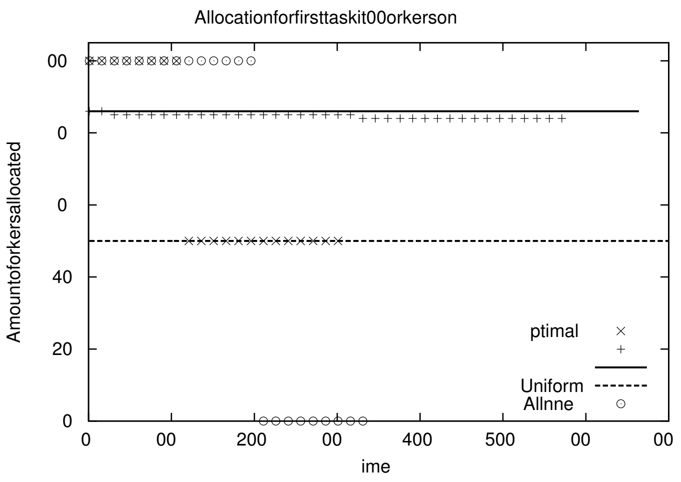

Figure 2: Assignments when $|N| = 100$ for task b_1 with no travel time and b_2 has any left over unassigned agents.

since it has a larger initial cost and thus larger growth. Around $t = 120$ task b_1 has decreased from 20 to about 12.5 while task b_2 has grown from 10 to 12.5. At this point the optimal solution splits the agents 50/50 between both tasks. The ALLONONE strategy performs very well in this case, because the initial assignment strategy happens to be identical to the optimal. LFF initially assigns 86 and 14 to b_1 and b_2 respectively, which gives completion times of $ct_1 = 387$ and $ct_2 = 664$. RT-LFF then balances this assignment to 85/15 at time $t = 15$, yielding new completion times of $ct_1 = 573$ and $ct_2 = 600$, and at $t = 320$ finishes balancing the assignment to 84/16 with $ct_1 = ct_2 = 580$. The growth functions are picked to be barely possible for LFF, causing a large difference in the initial completion times and RT-LFF's reassignments to have such a large impact. This case is shown to demonstrate the maximum effect of the greedy selection criteria.

RT-LFF is extremely resistant to variations in travel time compared to ALLONONE. When agents spend 20 time steps to travel between tasks, the completion time increased for RT-LFF by at most 4 while ALLONONE increased by at least 27. Both LFF and RT-LFF attempt to make the completion time of all tasks approximately equal. This means when $|N| >> |B|$ LFF and RT-LFF have approximately equal performance, but as $|N|$ approaches $|B|$ the improvements for RT-LFF can become quite significant. The UNIFORM and ALLONONE strategies can perform well if they happen to have initial allocations similar to the optimal strategy, otherwise they are prone to being unable to complete all the tasks.

RT-LFF can also adapt to inaccuracies in the estimation model by tracking the estimated completion times. When the growth model is perfect, the predicted completion times are static throughout the simulation. However, when the model is an approximation the completion time estimates will change even when no agent is reassigned. An example is when there is no travel time and $h_1(x) = h_2(x) = 0.000012$. If the model thinks $h_1(x) = h_2(x) = 0.00001$ and underestimates the rate of growth, completion times will increase as simulation progresses and agents will be reassigned when completion times become too unbalanced. With the real growth function 20% higher than estimated, RT-LFF was able to complete in 286 steps where it would have only taken 282 steps if the model was perfect. When RT-LFF overestimates the growth, the completion times will decrease without agent reassignment and agents will be reassigned to balance the completion times.

8. FIRES IN ROBOCUP

In this section we focus on the problem of dealing with fires in RoboCup Rescue and present a way of modeling how fire spreads and applying RT-LFF. The RoboCup Rescue simulator is designed for urban search and rescue after an earthquake, where buildings collapse and fires start in buildings. The environment is very large (see Figure 3) with upward of 100 agents. The full simulator uses heterogeneous agents, but for this work we focus only on the agents that can extinguish fires, i.e. firetrucks. The other agents constantly search for new fires, and broadcast the location and estimated size when a new fire is found.

Fires are the most dangerous hazard in RoboCup Rescue. While a single building on fire can be dealt with quickly, if too many buildings ignite the fire becomes very difficult to tackle both due to its size and re-ignitions of buildings. We present two novel contributions. First, fire clusters are modeled as single tasks that have a cost which increases as time passes. Second, we present a method to estimate the number of buildings on fire in a cluster when only a few buildings from that cluster have been observed. The RT-LFF algorithm is then shown to out-perform more naïve algorithms.

8.1 Growth of Fire Clusters

Each building on fire individually has a chance to ignite nearby buildings based primarily on distance. This means the rate of growth, $h(f_i^t)$, is proportional to the number of current buildings on fire, x:

$$\text{(a)}\quad \frac{\delta x}{\delta t} = g \times x \qquad \text{(b)}\quad x = C \times e^{g \times t} \tag{5}$$

Ten simulations with a single fire starting in various locations were used to empirically evaluate g to be roughly 0.0687. This is a first order linear differential equation that can be explicitly solved by separation of variables to yield the well known equation in (5b). A constant C is introduced by integration to satisfy the initial conditions. Eq. (5a) can be modified to incorporate the fire agents extinguishing effect on a fire. If there are n fire agents working on a fire cluster each extinguishing at a rate w, then the rate of growth of a fire will be reduced by $n \times w$:

$$\frac{\delta x}{\delta t} = g \times x - n \times w \tag{6}$$

The constant w was also empirically calculated to be about 0.184 by running ten simulations with a single fire agent and tracking the total number of fires extinguished after 100 cycles. Complications arise since the intensity of the fire has an effect on the effort required to extinguish it, which biases w to be higher than the real value. Agents can extinguish small fires much more quickly than larger fires, which often causes the agent to repeatedly put out small fires as they are reignited from nearby larger fires. Nevertheless, w gives a reasonable estimate for the agents' capabilities as shown in Section 9. Since g, n and w are constants, this still is a linear differential equation which simplifies to the slightly modified exponential growth function shown in (7).

$$x = \frac{n \times w}{g} + C \times e^{g \times t} \tag{7}$$

8.2 Estimating the Size of a Cluster

Running a probabilistic model for every single building to predict the fire spread would require a large amount of computation and have a low probability for every possible state. Ideally this model would be recomputed after every agent reassignment. This method will not scale and is too complex for a scenario in which a large amount of information is already missing. For that reason fires are abstracted into clusters and the macro-level behaviors of these clusters are analyzed instead.

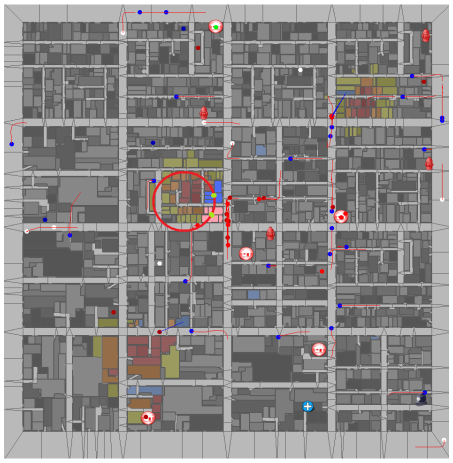

Figure 3: Pink and blue buildings' average position determine the two points on the red fire cluster circle.

The lack of full information in RoboCup Rescue makes clustering difficult and requires some assumptions to be made. If a large number of agents were available to circle around the fire cluster and monitor its growth, then direct clustering could accurately estimate the number of buildings on fire, namely x. However, normally agents are not able to dedicate this much time to information gathering, so it is necessary to come up with a way of estimating the size of a fire cluster while only being able to see a few buildings.

If we assume fires spread equally in all direction, a circle is a good estimate of the area on fire. This circle is found by identifying two points on its edge by first putting the most recently seen building and the 9 closest burning buildings (or all known nearby burning buildings if fewer than 9 exist) in a set. Then k-means clustering with $k = 2$ is run to find two subsets. The average position of each subset is then used as the two points of the circle, as shown in Figure 3.

The method for finding the radius can be extended from the exponential model given in Section 8.1. If x is the number of buildings on fire, then we can estimate $\pi \times r^2 = A \times x$, where r is the circle radius and A is the average building area (estimated over all buildings on the map). This can be rewritten as: $x = \frac{\pi \times r^2}{A}$ which can then be substituted into (5) yielding:

$$\frac{\delta r}{\delta t} = \frac{g}{2} \times r, \text{ and } r(t) = D \times e^{\frac{g}{2} \times t} \tag{8}$$

To overestimate the radius of the circle, we initially assume the fire started at the beginning of the simulation. For example if a fire is found at time $t = 50$, we would assume this fire started as a single burning building, $D = 1$, at time $t = 0$. Thus the radius would simply be $r(50)$. With a radius and two points on a circle, there are two possible circles and we choose the one with the highest ratio of burning buildings to total buildings.

This radius is the worst case possibility, so we incorporate new information to refine the estimate. For all buildings in the circle, we check their status at the last time viewed and neglect buildings never seen. If there is a conflict between past information and the assumption that the fire started at time $t = 0$, we assume the center of the circle is still the same and recompute D for a radius that satisfies the information. This new initial condition gives a smaller radius to be fit on the two circle points. This process is repeated until there is no conflicting information.

8.3 Assignment to Fire Clusters

When assigning agents to clusters the goal is to extinguish all fires as quickly as possible. Eq. (7) can be solved for t when $x = 0$, yielding the completion time, ct, as shown in (9). If the constant C is nonnegative, then the fire is growing faster than the n agents can extinguish it, thus $ct = \infty$. When C is negative, (9) is computable and will give the time when the fire will be fully extinguished.

$$ct = (\ln \frac{n \times w}{g \times -C})/g \tag{9}$$

Algorithm RoboCup RT-LFF with $h_i(x) = x$

```
Require: $C_i, C_j, n_i^t, n_j^t, w, g, t$ {For task $b_i$, $C_i$ is the integration constant from Eq. 7 and $n_i^t$ is the number of agents working at time $t$. A similar relationship exists for $b_j$, $C_j$ and $n_j^t$. $w$ is the task completion rate of an agent, $g$ is the growth rate of fires.}
1:  $change \leftarrow$ true
2:  while $change$ do
3:    $change \leftarrow$ false
4:    for $i = 1$ to $|B|$ do
5:      $ct_i^- \leftarrow (\ln \frac{(n_i^t - 1) \times w}{g \times -C_i})/g$ {Compute the time to extinguish fire $i$ with one fewer agent than currently assigned.}
6:      for $j = 1$ to $|B|$ do
7:        $\hat{x_j} \leftarrow n_j^t \times w/g + C_j \times e^{g \times (t + TT(b_i, b_j))}$ {Before the agent from fire cluster $i$ arrives to cluster $j$, compute the effect of the agents already there.}
8:        $\hat{C_j} \leftarrow (\hat{x_j} - (n_j^t + 1) \times w/g)/(e^{g \times (t + TT(b_i, b_j))})$ {Once the agent from cluster $i$ arrives, we need to recompute $C_j$.}
9:        $ct_j^+ \leftarrow (\ln(n_j^t + 1) \times w/(g \times -\hat{C_j}))/g$ {Finally compute when fire $j$ is fully extinguished.}
10:       if $ct_i^- < ct_j^+$ then
11:         Transfer an agent from $i$ to $j$
12:         $n_i^t \leftarrow n_i^t - 1$
13:         $n_j^{t+TT(b_i,b_j)} \leftarrow n_j^{t+TT(b_i,b_j)} + 1$
14:         $change \leftarrow$ true
15:       end if
16:     end for
17:   end for
18: end while
```

In the adaption of RT-LFF to RoboCup Rescue, $h_i(x) = x$ for all tasks. Normally in RoboCup Rescue no fire clusters are known initially, so when the first fire cluster is found all agents are assigned to that cluster. As discussed in Section 6, how many times and in what order pairs of clusters are considered has an effect on the performance of RT-LFF. When a new fire cluster is found, multiple agents may need to be transfered from the old clusters, so the algorithm runs until no more useful transfers exist. Newly found fire clusters are labeled b_1 to have the first chance of receiving agents from other clusters, which are ordered by increasing distance from this new task to reduce the traveling time.

9. RESULTS

In this section, we show the validity of our exponential model and RT-LFF by empirical evaluation in RoboCup Rescue. First, we show how the exponential model accurately fits the real fire growth data. Then the model is used to estimate the time a fire is extinguished and compared against the real time it took for agents to extinguish the fire. Finally RT-LFF is compared against the ALLONONE and UNIFORM strategies and a "CLOSEST" strategy which simply assigns agents to the closest unfinished task.

9.1 Model Fitting

A single fire was tracked over 5 simulations on two maps and the actual number of fires is compared against the exponential function in Figure 4. This approximation is reasonable, except it slightly underestimates the fires between $t = 40$ and $t = 60$.

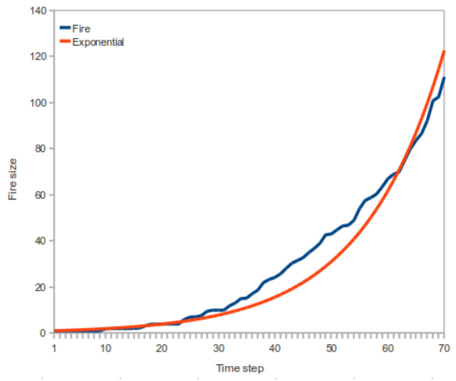

Figure 4: Real fire count compared to exponential.

The estimate of fire cluster size described in Section 8.2 was tested by assigning a static amount of firetrucks to extinguish the fire cluster and comparing the estimated and real extinguish time. The estimated size of the cluster is recomputed every time step and increases in accuracy as more of the environment is observed. For worst case analysis the real extinguish time is compared against the estimated extinguish time when the fire cluster is first discovered. A histogram of 54 fires extinguished with the relative error is shown in Figure 5 with a mean error of 0.0268 and standard deviation of 0.2526. The distribution is close to Gaussian and fairly unbiased at overestimating or underestimating. Some error may be due to imprecise empirically derived constants in the modeling equation or a lack of incorporating the intensity in the model.

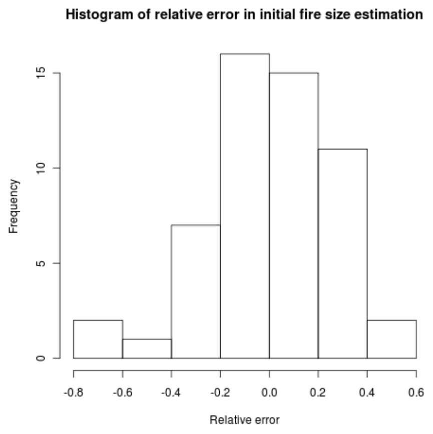

Figure 5: Relative error of extinguish time when a cluster is first discovered.

9.2 Comparison of Assignment Strategies

Each map was run with 5 simulations and the time the last fire was extinguished is reported in Table 2. A fixed number of non-firetruck agents randomly search the environment for new tasks and broadcasts the locations when seen. The agents searching the environment are unable to interact with any of the tasks directly. The

Map	RT-LFF	CLOSEST	UNIFORM	ALLONONE
Berlin	118.2	165	141.4	189
Virtual City	95.8	115	162.0	109

Table 2: Average completion times of the last task for two maps in RoboCup Rescue Simulation.

UNIFORM and ALLONONE strategies described in Section 7 were reimplemented for the RoboCup Rescue Simulator. The UNIFORM strategy does not only assign initially as in Section 7, but will reassign an approximately equal number of agents to all tasks when a new task is discovered. Also a "CLOSEST" strategy is implemented for RoboCup, where agents go to the closest known unfinished task, work until the task is complete and then go to the next closest task.

Berlin is a large map with no fires in the middle of the map, which required both the ALLONONE and CLOSEST strategies to reassign many agents to the other side of the map. When the ALLONONE strategy discovers a new task, it is added to the end of the current task list. This behavior caused agents to cross the map multiple times as new fires were discovered on alternate sides of the map. The UNIFORM strategy spreads agents equally, which reduces the amount of travel for the agents, but unlike LFF cannot estimate the growth of fires to more efficiently assign agents. RT-LFF reassigns agents much more frequently than in Section 7, due to approximations in both the cost growth function and initial size estimates. If the cost growth function is an underestimate for a task, the completion times will become unbalanced and will cause RT-LFF to assign more agents to that task. This behavior lets RT-LFF handle imperfect estimations of $h_i(f_i^t)$ by detecting when a discrepancy arises.

Virtual City as seen in Figure 3 has three fires that are almost collinear. This allows the CLOSEST strategy to have a reasonable distribution of agents, and since the fires near the corners get extinguished first there is only a short distance to move to the next closest task. One of the fire grew more quickly since it was actually started from two individual fires that were close together. When the fire was identified, the initial fires had overlapped and were clustered as a single task. The UNIFORM strategy poorly assigned for this fire, while RT-LFF was able to compensate for the error in the model. ALLONONE did quite well since this critical task was assigned all agents after the first task completed.

On both maps RT-LFF was able to complete all tasks in about 85% of the time next best strategy and was statistically significant from other strategies on both maps. The major random factor for all maps was the difference in time when new tasks were discovered by the exploring agents. The CLOSEST and ALLONONE strategy are both very resistant against changes to the discovery time. ALLONONE would simply add this new task to the end of the list, so the algorithm would perform no differently if the new task was discovered almost immediately or right before the last known task was completed. For the CLOSEST strategy an agent does not reassign until its task is complete, thus the agent acts on this knowledge only slightly faster than AllOnOne. RT-LFF and UNIFORM almost always immediately reassign agents when a new task is discovered, but RT-LFF was able to minimize variance in completion time by properly assigning more agents when a task was discovered later.

10. CONCLUSIONS AND FUTURE WORK

This paper addresses task allocation with multiple agents, where each task has a cost that changes over time. This adds substantial complexity and requires more coordination between agents. We limited the investigation of cost changes to a general family of functions in order to reduce the complexity. We presented the Latest Finishing First (LFF) algorithm, which maximizes the amount of time agents work and attempts to finish all tasks at the same time. We went on to present a real-time solution, RT-LFF, which improves LFF for partially observable spaces and is resistant to noise or errors in the model.

While RT-LFF's heuristic is able to compensate for errors in $h_i(f_i^t)$, it might be possible to start with an assumption about the growth function but refine the coefficients for that particular task as the simulation progresses. Another possibility to explore is to include heterogeneous agents into the model. The choice of metric for evaluating an algorithm when the task cost functions change is also an open question, especially when no solution exists. One metric would be to minimize the total cost over all time, regardless of how many tasks are completed. Perhaps if there is one large task and many smaller tasks, it would be best to maximize the number of tasks completed by finishing all the smaller tasks while completely ignoring the large task.

Acknowledgment: Work supported in part by NSF IIS-1208413 and the Safety, Security, and Rescue Research Center at the University of Minnesota (NSF IIP-0934327).

11. REFERENCES

[1] Álvaro Monares, S. F. Ochoa, J. A. Pino, V. Herskovic, J. Rodriguez-Covili, and A. Neyem. Mobile computing in urban emergency situations: Improving the support to firefighters in the field. *Expert Systems with Applications*, 38(2):1255 – 1267, 2011.

[2] A. Chapman, R. A. Micillo, R. Kota, and N. Jennings. Decentralised dynamic task allocation using overlapping potential games. *The Computer Journal*, 2010.

[3] P. R. Ferreira, Jr., F. dos Santos, A. L. C. Bazzan, D. Epstein, and S. J. Waskow. RoboCup Rescue as multiagent task allocation among teams: experiments with task interdependencies. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(3):421–443, 2010.

[4] H. Kitano and S. Tadokoro. RoboCup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39–52, 2001.

[5] M. Nanjanath, A. Erlandson, S. Andrist, A. Ragipindi, A. Mohammed, A. Sharma, and M. Gini. Decision and coordination strategies for RoboCup rescue agents. In *Proc. SIMPAR*, pages 473–484, 2010.

[6] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings. Coalition formation with spatial and temporal constraints. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1181–1188, 2010.

[7] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1–2):209–238, 1999.

[8] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 727–734, 2005.

[9] Y. Zhang and L. E. Parker. Task allocation with executable coalitions in multirobot tasks. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2012.

[10] X. Zheng and S. Koenig. Reaction functions for task allocation to cooperative agents. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 559–566, 2008.