Teams to exploit spatial locality among agents

James Parker and Maria Gini
Department of Computer Science and Engineering, University of Minnesota
Email: [jparker,gini]@cs.umn.edu

***Abstract*—In many situations, task allocation is highly dependent on the spatial locality of the agents and the tasks. In this paper we explore task allocation in complex spatial environments, where the cost of completing a task can vary over time. This work focuses on a subset of cost functions for tasks that grow over time and presents the real-time Latest Finishing First (RT-LFF) algorithm, which strikes a balance between the optimal zero travel time solution and minimizing the amount of time agents spend transferring to tasks in different locations. We also present how spatial knowledge can be used to infer missing knowledge in partially observable spaces.**

I. Introduction

Multi-agent systems offer robustness, flexibility, and efficiency over single-agent systems. The fields of wide-area surveillance, exploration and mapping, transportation, and search & rescue are all being enriched by incorporating multiple agents. In all these examples, not only must agents travel through the spatial topology of the environment to accomplish their goals, but the topology itself can effect how the agent's goals can change over time. However, the benefits of multi-agent systems increases the burden on coordination.

In dynamic environments where the set of agents might change over time, as agents fail or new agents are added to the system, and where new tasks can appear and old tasks can expire, task allocation has to be done in real-time to handle the uncertainty as to location, completion requirements and size of tasks. Also agents travel through unknown and possibly unsafe areas to reach the tasks. Unlike most task allocation problems [1], we focus on domains where the amount of resources that a task requires changes over time.

We present the *Latest Finishing First (LFF)* algorithm for task allocation in domains where the cost of tasks increases with time (Section IV). These types of problems 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 later. We then extend our LFF algorithm to a real-time heuristic version for partially observable dynamic environments where new tasks can appear as time progresses (Section V). Finally, we propose a novel way of incorporating partial information in RoboCup Rescue to accurately predict a model for clustered tasks (Section VI), and we evaluate experimentally our work against other methods (Section VII).

II. Related Work

Multi agent task allocation is well known to be an NP hard problem, giving rise to many different techniques for finding an approximate solution. One example is swarm robotics, where agents often use threshold based methods to 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 allocates itself to the task. If not, the agent passes the information to other agents. An example is [2], which uses distributed constraint optimization (DCOP) as a basis for task allocation. A comparison between DCOP and other swarm techniques is provided in [3]. In comparison, market inspired auction methods typically require more communication and are more centralized. Zhang et al. [4] 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 swarm and centralization, each agent is directed to an area by a central authority, but upon reaching the destination, agents act on their own individual abilities.

Other approaches have been developed, such as modeling task allocation as a potential game [6]. Sandholm et al. [7] present a generalized coalition formation algorithm which produces solutions within a bound from the optimal via pruning a subset of the search tree. Dang et al. [8] improve on it by further pruning, but the search time is still exponential. The work in [9] focuses on tasks that require more than one agent to do them, while simultaneously trying to use efficiently the agent's resources and time. Our approach also assumes multiple agents are required, but we also allow the requirements of tasks to change over time.

Our work specifically applies to urban search & rescue using the RoboCup Search and Rescue Simulator [10], providing simulations based off street and building maps of real cities. Emergency situations are very time critical and often lacking in information, as demonstrated by [11]. Most notably, when the emergency occurs agents are spatially spread out in a disorganized fashion and must quickly coordinate and form teams to accomplish tasks.

III. Problem description

Our problem is task allocation for multi-agent systems, where agents must travel on designated pathways. The agents are scattered at random initially and need to converge on tasks. These tasks require more work to be completed as time progresses, which necessitates multiple agents being allocated to them. Thus, simply going to the nearest task can cause overallocation and other tasks might grow out of control.

We denote the set of homogeneous agents by $A = \{a_1, \ldots, a_{|A|}\}$ and the set of tasks by $B = \{b_1, \ldots, b_{|B|}\}$. The set of assignments of agents to a task 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 assigned to task b_i at time unit t. An agent $a_i \in A$ can only work on one task, b_j, at a time and at most one assignment $n_j(t) \in N$ can contain a_i. All agents and tasks

have a spatial location in the environment and the travel time between two locations is assumed to be computable.

Each agent provides the amount of work w per time unit towards the task it is assigned. Every task $b_i \in B$ has a cost function $f_i^t : (f_i^{t-1}, |n_i(t-1)|) \to \mathbb{R}$ with the relationship:

$$f_i^{t+1}(f_i^t, |n_i(t)|) = f_i^t(f_i^{t-1}, |n_i(t-1)|) + \Delta f_i^t \quad (1)$$

where Δf_i^t has the form:

$$\Delta f_i^t = g_i(f_i^t) - w \times |n_i(t)| \quad (2)$$

and $g_i : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ is a monotonically increasing function with initial value f_i^0. For simplicity, we will treat f_i^t as a value rather than a function if the values of f_i^{t-1} and $|n_i(t-1)|$ are clear in the context. This means f_i^t is strictly monotonically increasing when $g_i(f_i^t) > w \times |n_i(t)|$ and strictly monotonically decreasing when $g_i(f_i^t) < w \times |n_i(t)|$. When the cost function f_i^t reaches or passes zero at some time $t_i(0)$, the task is considered complete and is removed from the problem. In our problem the cost of completing a task increases over time, therefore the goal of task allocation is to minimize the time the last task is completed.

IV. Latest Finishing First

With perfect knowledge about how a task grows, we can predict what allocation of agents would be ideal. This ideal solution is not reachable due to travel time being required to reach tasks, but even in this ideal setting task allocation is not as simple as it seems. Even neglecting the spatial limitations, when tasks grow over time some allocations of agents are better than others. Taking into account the spatial limitations, we derive the Latest Finishing First (LFF) algorithm, which we present after first describing properties of the dynamic cost functions we will use. Throughout this section, we assume that time t is sufficiently small to enable us to treat t as continuous and we assume a solution exists. $TT(x, y)$ is defined to be the travel time between location x and location y.

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

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

Note that $\sum_{b_i \in B} f_i^0$ 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 overall percent of time the agents are working. We can then rewrite (3) as:

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

This means that $\sum_{b_i \in B} \sum_{t<t_s} g_i(f_i^t)$ 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 $\Delta R_t = R_t - R_{t-1} = \sum_{b_i \in B} g_i(f_i^t)$.

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 $\hat{R}_{t_{\hat{s}}} = \sum_{b_i \in B} \sum_{t<t_{\hat{s}}} g_i(\hat{f}_i^t)$ where $\hat{S} < S$ with $\hat{R}_{t_{\hat{s}}} > R_{t_s}$. 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 $\hat{R}_{t_{\hat{s}}}$ in terms of t_s:

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

Using our two assumptions, we can then write:

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

which implies that R_{t_s} satisfied (3) at time $t_{\hat{s}}$. This is a contradiction. ∎

Theorem 2: Minimizing ΔR_{t_s} for every time unit t also minimizes the global regret, R_{t_s}, when 1. $TT(x, y) = 0 \ \forall x, y$, 2. $g_i = g_j \ \forall i, j$, 3. $\lim_{x \to 0^+} g(x) = 0$, and 4. $\frac{\partial^2}{\partial t^2} g_i(f_i^t) \geq 0$.

Proof: Since $g_i = g_j \ \forall i, j$, we will denote g_i with g to simplify notation. Assume there exists a better solution $\hat{F}_i = \{\hat{f}_i^1, \hat{f}_i^2, \ldots\}$ which differs from the greedy minimization $F_i = \{f_i^1, f_i^2, \ldots\}$. This means that at some time t_d the better solution must assign agents differently than the greedy solution. By definition the greedy minimization is a minimum ΔR_{t_d} at time t_d, thus $\Delta R_{t_d} \leq \Delta \hat{R}_{t_d}$, where $\Delta \hat{R}_t = \sum_{b_i \in B} g(\hat{f}_i^t)$. After time t_d, the greedy minimization will allocate agents exactly in the same way as the better solution 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_i 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 can then compute for f_i^{t+1} as:

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

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

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

If we use the monotonicity of g, then we can see $g(f_i + c_i) = g(f_i) + \delta_i \times c_i$ for some $\delta_i > 0$. This means $\hat{f}_i^{t+1} - f_i^{t+1} = c_i + \delta_i \times c_i$ or rearranging: $f_i^{t+1} + c_i + \delta_i \times c_i = \hat{f}_i^{t+1}$. Since $\frac{\partial^2}{\partial t^2} g_i(f_i^t) \geq 0$ we know $\delta_i \geq \delta_j$ when $c_i > c_j \ \forall i, j$. Thus, $\sum_{b_i \in B} \delta_i \times c_i \geq 0$ since $\sum_{b_i \in B} c_i \geq 0$. 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 $\hat{R}_{t_s} = \sum_{b_i \in B} \sum_{t<t_s} g(\hat{f}_i^t)$. 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 $\lim_{x\to 0^+} g(x) = 0$, so when a task is completed it simply disappears from the equations. ■

Unfortunately, the assumption that $TT(x, y) = 0 \forall x, y$ is rather strong. When agents require time to move between different task clusters, every time an agent is allocated to a different task, $\bar{p}$ decreases in (4). To address this issue, we introduce Latest Finishing First (LFF) in Algorithm 1. The inspiration behind LFF is to create a stable assignment that will try to maintain $\bar{p}$ close to 1 to maximize the overall output of agents in the system. To do this, each task is prioritized and the closest agent to the highest priority task is assigned to that task. After an agent has been assigned, the priority of the highest priority task is recomputed and this process is repeated.

The priority of a task is the expected time to complete that task, so tasks that take longer have a higher priority. Since g_i increases work needed to complete a task, all tasks initially start out never completing (or $t_i(0) = \infty,\ \forall i$). In case of ties, the task which has the largest current cost is allocated the closest agent. This causes the completion times of all the tasks to be as balanced as possible, which in turn reduces the amount of travel needed for agents.

Algorithm 1 Latest Finishing First

while $\exists a_i \in A$ and $\nexists j$ such that $a_i \in n_j(0)$ **do**
  Find b_i for $\underset{b_i \in B}{\text{argmax}}\ t_i(0)$ {Task b_i ends last}
  if $\exists b_i, b_j$ where $t_i(0) = \infty$ and $t_j(0) = \infty$ **then**
    Find b_k for $\underset{b_k \in B}{\text{argmax}}\ f_k^0$ with $t_k(0) = \infty$
    Assign $\underset{a_j \in A \text{ and } a_j \text{ unassigned}}{\text{argmin}}\ TT(a_j, b_k)$
  else
    Assign $\underset{a_j \in A \text{ and } a_j \text{ unassigned}}{\text{argmin}}\ TT(a_j, b_i)$
  end if
end while

V. Real-Time Latest Finishing First

This section extends LFF outlined in Algorithm 1 to a decentralized real-time solution for dynamic partially observable environments in Algorithm 2. The LFF algorithm assumed all the tasks were known initially, but without this assumption it becomes impractical to try and minimize agent movement. When new tasks are discovered, it is crucial that agents are reallocated to the new tasks. The Real-Time Latest Finishing First (RT-LFF) algorithm assumes that new tasks are identified from an oracle, and when a new task is identified agents are reallocated to try finish all tasks at the same time.

When the task allocation first starts, the normal LFF is used to compute the assignments of agents. While time is progressing, if a new task, b_j, is identified then RT-LFF uses a greedy heuristic to reallocate agents to b_j. Tasks are ordered based on spatial proximity to b_j, and tasks that are closest to the new task attempt to reallocate agents first. Let $t_i^-(0)$ be the time for task b_i to complete with one fewer agent and $t_j^+(0)$ be the time for task b_j to complete with one additional agent including the delay from the agent traveling. An agent is only reallocated from b_i to b_j if $t_i^-(0) < t_j^+(0)$, namely a greedy heuristic which only transfers agents if the original task will still finish first even after reallocating an agent.

If the oracle informs all agents of the new task, each agent can compute this algorithm and agree on which agents should transfer without any communication. In the case where agents have limited computational power, a single more powerful agent at or nearby each task could direct this algorithm. If the oracle only informs the agents closest to the newly identified task, these agents would run the RT-LFF algorithm, since they are the highest priority, and after any assignments pass on updated information to the next closest agents.

Theorem 3: RT-LFF has at most $|B| - 1$ misallocated agents from LFF at any time assuming zero travel time.

Proof: We show that using RT-LFF a task b_1 will never be assigned more than one agent when using LFF by breaking it down into two cases. First, assume that we have two tasks b_1 and b_2 and let $t_j^{++}(0)$ be the time task b_j is completed with two additional agents assigned to it and similarly $t_i^{--}(0)$ be the time task b_i is completed with two less agents assigned to it. Without loss of generality assume $t_2(0) < t_1(0)$. If RT-LFF did not reallocate an agent from b_2 to b_1, then $t_2^-(0) > t_1^+(0)$. We notice that reallocating two agents from b_2 to b_1 means $t_2^{--}(0) > t_2^-(0)$ and $t_1^{++}(0) < t_1^+(0)$, which is a contradiction to the RT-LFF algorithm since b_1 would allocate an agent back to b_2 since $t_2^-(0) > t_1^+(0)$.

Next we show that a task b_1 would never receive a single agent from more than one cluster. The argument is similar to the first part, only it now involves task b_3. If b_2 and b_3 did not reallocate an agent to b_1 under RT-LFF, then $t_1^+(0) < t_2^-(0)$ and $t_1^+(0) < t_3^-(0)$. Suppose LFF did have both b_2 and b_3 reallocate an agent to b_1, then without a loss of generality assume $t_2^-(0) < t_3^-(0)$. Then $t_1^{++}(0) < t_1^+(0) < t_2^-(0) < t_3^-(0)$. This is a contradiction to how LFF works. Currently $t_3^-(0)$ is the last finishing and $t_1^{++}(0)$ is the fastest finishing, and from the equation above if b_1 gave back an agent to b_3 then $t_1^+(0) < t_2^-(0)$. This means the transfer back has reduced the time of latest finishing task, thus under RT-LFF b_1 will be not allocated an agent from more than one cluster compared to LFF. This implies a task in RT-LFF cannot have more than one agent under the allocation of LFF for each time step. The worst case is when $|B| - 1$ tasks have one agent fewer than LFF's allocation with the last task over allocated. ■

Algorithm 2 Real-Time Latest Finishing First

Use LFF for initial allocation
while $t < t_s$ **do**
  for $b_i \in B$ **do**
    if $\exists b_i \neq b_j$ where $t_i^-(0) < t_j^+(0)$ with $\underset{b_i}{\text{argmin}}\ TT(b_i, b_j)$ **then**
      Reassign $a_k \in N_i(t)$ to b_j with $\underset{a_k}{\text{argmin}}\ TT(a_k, b_j)$
    end if
  end for
end while

VI. Modeling Cost of Task Completion for Extinguishing Fires

In this section we focus only on the subproblem of dealing with fires in RoboCup Rescue and present a way of modeling how fire spreads and assigning agents to fires. The RoboCup Rescue simulator is designed for urban search & rescue after an earthquake, which fires to start in buildings. The environment is large (see Figure 1) 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 simulate the oracle and constantly search for new fires, and relay 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 heuristics.

A. 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, g, 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 exponential growth function given by (5b). A constant C is introduced by integration to satisfy the initial conditions. Eq. (5a) can be modified to incorporate fire agents extinguishing effect on a fire. If there are n fire agents assigned to a fire cluster that each extinguish at a rate w, then the rate of growth of a fire of a fire cluster 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. Matters are further complicated since the intensity of the fire has an effect on the amount of time it takes to extinguish, thus, w is biased 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 agent's capabilities as shown in Section VII. Since both n and w are constants, this still is a linear differential equation which simplifies to a slightly modified exponential growth function shown in (7). This satisfies all the conditions in Theorem 2 except the zero travel-time assumption.

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

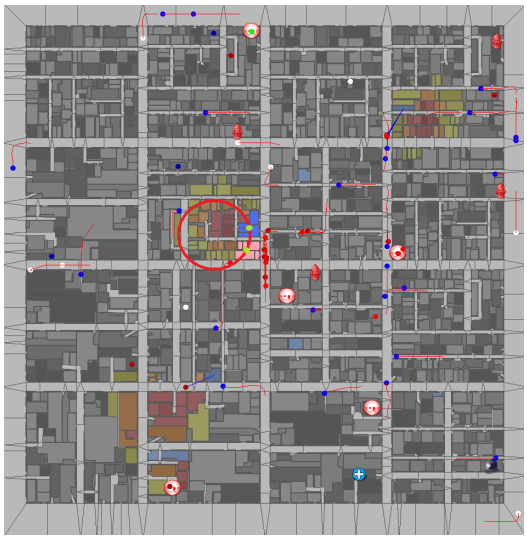

Fig. 1. Pink and blue buildings' average position determine the two points on the red fire cluster circle.

B. Estimating the Size of a Cluster

Clustering is commonly done to group similar objects into a single abstract object to reduce the complexity and to enable a macro-level analysis. 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. Furthermore, the model will change based on which fires agents are assigned to extinguish, so ideally the model should be recomputed after every assignment. 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.

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 using (6) with current known information would be a good estimate. 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. To do this we need to make one assumption: on average fire propagates equally in all directions.

From this assumption, a circle is a good estimate of the area on fire. This circle is identified by two points on its edge in the following manner: First, the most recently seen building and the 9 closest burning buildings (or all known nearby burning buildings if fewer than 9 are on fire) are included 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 to fit the circle upon, shown in Figure 1.

The method for finding the radius can be extended from the exponential model given in Section VI-A. 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 (5a) yielding:

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

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 to choose from. The circle with the highest ratio of burning buildings to total buildings is the one chosen as the fire cluster.

Since this radius is the worst case estimate, we should incorporate current information to refine the estimate. For all buildings in the circle, we check their status at the last time viewed. If a building has never been seen, we neglect it. 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 and this time we choose the circle whose center was inside the old overestimated circle. This process is repeated until there is no conflicting information.

C. Assignment to Fire Clusters

The goal of assigning agents to clusters is to extinguish fires as quickly as possible. Eq. (7) can be solved for t when $x = 0$ as shown in (9). If the constant C is nonnegative, then the fire is growing faster than the n agents can extinguish it. Thus the fire will grow indefinitely and $t(0)$ is set to ∞. When C is negative, (9) will be computable and will give the time when the fire will be fully extinguished.

$$t(0) = (\ln \frac{n \times w}{g \times -C})/g \quad (9)$$

Algorithm 3 shows the implementation of the RT-LFF algorithm for the case when $g(x) = x$ in RoboCup Rescue. 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. When a new fire cluster is found, multiple agents may need to be transfered from the old clusters. This is why any time an agent was transfered, we should run the algorithm again until no more useful transfers exist.

The order in which the fire clusters are labeled is important, since every cluster first attempts to transfer an agent to b_1. When a new fire is discovered, it will initially have no agents assigned and will need to receive agents from other clusters. Since this fire cluster is in the greatest need of agents, it is assigned b_1. All other clusters are then reordered based on their distance from cluster b_1, the newly discovered fire. Thus b_2 will be the closest fire cluster to b_1, b_3 the second closest and so forth. This helps reduce the travel time $TT(i, j)$ in order to more efficiently use fire trucks.

Algorithm 3 RT-LFF for g(x) = x

Require: $C_i, C_j, n_i, n_j, w, g, t$ {C_i is the integration constant and $n_i(t)$ is the number of agents on task b_i, C_j and n_j correspond to similar things on task b_j, w is the task completion rate of agents, g is the growth rate of fires and t is the current time.}

$change \leftarrow$ **true**
while $change$ **do**
  $change \leftarrow$ **false**
  for $i = 1$ to k **do**
    $t_i^-(0) \leftarrow (\ln \frac{(n_i(t)-1) \times w}{g \times -C_i})/g$ {Compute the time to extinguish fire i with one less agent than it currently is assigned.}
    for $j = 1$ to k **do**
      $\hat{x_j} \leftarrow \frac{n_j(t) \times w}{g} + C_j \times e^{g \times (t + TT(i,j))}$ {Before the agent from fire cluster i arrives to cluster j, compute the effect of the agents.}
      $\hat{C_j} \leftarrow (\hat{x_j} - \frac{(n_j(t)+1) \times w}{g})/(e^{g \times (t + TT(i,j))})$ {Once the agent from cluster i arrives, we need to recompute C_j.}
      $t_j^+(0) \leftarrow (\ln \frac{(n_j(t)+1) \times w}{g \times -\hat{C_j}})/g$ {Finally compute when fire j is fully extinguished.}
      if $t_i^-(0) < t_j^+(0)$ **then**
        Transfer an agent from i to j
        $n_i \leftarrow n_i - 1$
        $n_j \leftarrow n_j + 1$
        $change \leftarrow$ **true**
      end if
    end for
  end for
end while

VII. 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 a random assignment method and a closest distance greedy heuristic.

A. Model Fitting

A single fire was tracked over 5 simulations on two maps (Virtual City and Berlin) and the actual number of fires is compared against the exponential function best fit in Figure 2. The exponential function slightly under estimates the fire cluster between $t = 40$ and $t = 60$ but is otherwise fairly accurate.

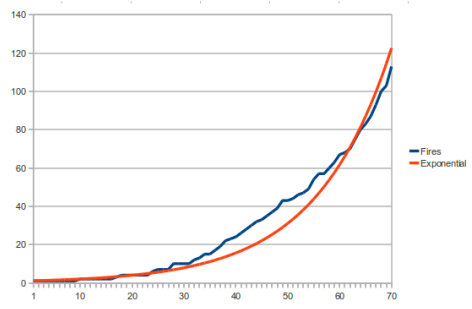

Fig. 2. Average fire spread estimated by best exponential fit to data.

The estimate of fire cluster size described in Section VI-B

was tested by assigning a static amount of fire trucks 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 a larger number of buildings in the cluster 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 normalized error is shown in Figure 3. The error in estimation had a mean 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.

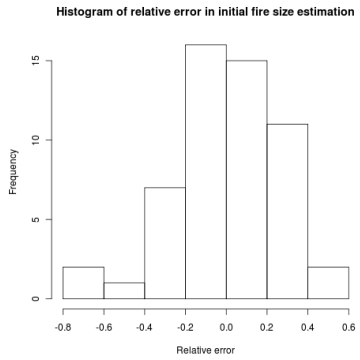

Fig. 3. Relative error of extinguish time when a cluster is first discovered.

B. Performance Comparison of Different Allocation Strategies

Each map was run with 5 simulations and the time the last fire was extinguished is reported in Figure 4. In each case, the oracle is simulated by a fixed number of agents randomly searching the environment for new tasks and broadcasting the location of the tasks when they are discovered. The fixed number of agents searching the environment do not interact with any of the tasks directly. RT-LFF method does much better in Virtual City (8.3% better than the closest heuristic approach) than in Berlin (4.0% better). This is probably due to Virtual City being an artificial map with similar building sizes and a compact building arrangement. Berlin has open spaces where there is a river or park that throws off the accuracy of the circle estimating method. There is also a larger discrepancy in building size in Berlin which can again effect the circle estimate. If a large building is chosen for one of the two clusters to estimate the two points on the circle, the buildings in this cluster will be more spread out and would give an inaccurate point estimate since the center of the large building is far away from the others. The random method performs poorly on both maps, due to lack of cooperation.

VIII. Conclusions and Future Work

This paper addresses task allocation with multiple agents, where each task has a cost that changes over time. This adds a substantial amount of complexity and requires more coordination between agents. We limited the investigation of dynamic cost tasks to a general family of functions in order to reduce the complexity. We presented the Latest Finishing

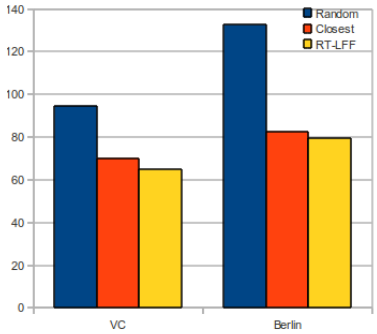

Fig. 4. Average finish time (no fires left) per map for each configuration.

First algorithm, which attempts to minimize the time when the last task finishes. We documented properties exhibited by dynamic tasks and went on to present a real-time solution, which approximates LFF for partially observable spaces. The algorithms currently work only for homogeneous agents with identical abilities. We hope to expand the current model to include other agent types. The choice of metric for evaluating an algorithm for dynamic cost functions is also an open question, especially when no solution exists. Completing fewer tasks at the expense of ignoring others might be beneficial more than keeping the current total cost low.

References

[1] 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*, 2010, pp. 1181–1188.

[2] 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*, 2005, pp. 727–734.

[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*, vol. 20, no. 3, pp. 421–443, 2010.

[4] 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.

[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*, 2010, pp. 473–484.

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

[7] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 1–2, pp. 209–238, 1999.

[8] V. D. Dang and N. R. Jennings, "Generating coalition structures with finite bound from the optimal guarantees," in *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2004, pp. 564–571.

[9] 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*, 2008, pp. 559–566.

[10] H. Kitano and S. Tadokoro, "RoboCup rescue: A grand challenge for multiagent and intelligent systems," *AI Magazine*, vol. 22, no. 1, pp. 39–52, 2001.

[11] 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*, vol. 38, no. 2, pp. 1255 – 1267, 2011.