Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Controlling Growing Tasks with Heterogeneous Agents

James Parker and **Maria Gini**
University of Minnesota
jparker@cs.umn.edu and gini@cs.umn.edu

Abstract

We propose solutions for assignment of physical tasks to heterogeneous agents when the costs of the tasks change over time. We assume tasks have a natural growth rate which is counteracted by the work applied by the agents. As the future cost of a task depends on the agents allocation, reasoning must be both spatial and temporal to effectively minimize the growth so tasks can be completed. We present optimal solutions for two general classes of growth functions and heuristic solutions for other cases. Empirical results are given in RoboCup Rescue for agents with different capabilities.

1 Introduction

Wide area surveillance, search and rescue, transportation, and exploration all benefit from efficient task allocation methods. For example, transportation costs can be minimized by planning distribution routes [Toth and Vigo, 2002]. We extend task allocation to cover problems where the costs for completing tasks change over time. In these situations, the tasks growth is similar to their reduction due to agents work. This adds a strong temporal component that requires a high degree of coordinated effort between agents. Areas of application include containment of forest fires, minimization of damage from an invasive species, resource allocation for fighting epidemics, and search and rescue [Kruijff *et al.*, 2012].

Task growth over time is most interesting when agents need to coordinate their efforts to finish a single task. For this reason, we primarily focus on cases where multiple agents are needed to complete a task. If too few agents are assigned to a task, the task cost will grow towards infinity. If the growth rate of a task surpasses the reduction that all agents combined can provide, then that task can no longer be completed. Our goal is to minimize the total growth of tasks before all tasks are completed.

This paper starts with the problem framework and algorithms in [Parker and Gini, 2014]. The main contributions of this paper are: (1) a novel formulation for heterogeneous agents; (2) proofs of optimal solutions for a relaxed version of the problem with zero travel time for two families of task growth functions; (3) a heterogeneous agent version of the Latest Finishing First algorithm (LFF) for cases when travel times are prohibitively large and agents are not reallocated; (4) a heterogeneous agent version of the Real-Time Latest Finishing First algorithm (RT-LFF) for cases where travel times are non-prohibitive; (5) experimental modeling and results in the RoboCup Rescue Agent Simulator [Kitano and Tadokoro, 2001].

2 Related Work

Multi-agent task allocation typically assumes that each task has a known and fixed cost. A few studies address tasks where the task costs are uncertain (e.g. [Mills-Tettey *et al.*, 2007; Nam and Shell, 2015]), but not where task costs change over time according to some cost function. Other methods for adapting to dynamic environments with localized communication (i.e., [Léauté and Faltings, 2011; Atlas and Decker, 2010]), assume costs change in a stochastic way. In our work, tasks are assumed to have a known trend and we exploit this knowledge to optimize the allocation. The work of [Zhang and Parker, 2013] considers general coalition formation with instantaneous assignment. We consider a less general problem domain, but are able to find the optimal solutions for them. We also provide solutions for non-instantaneous assignments. Robust optimization is another way to frame this problem [Beyer and Sendhoff, 2007], but our problem model often does not have a closed form solution.

Tasks with decreasing rewards over time are considered in [Amador *et al.*, 2014]. Their algorithm encourages agents to arrive to tasks quickly. If tasks are not finished quickly, other tasks might get a higher priority. In our case, if a task is neglected, the growth makes the task impossible to finish. The disparity between an impossible task versus a low utility task makes our problems not directly comparable.

Work in multi-agent task allocation with temporal constraints is also relevant [Nunes *et al.*, 2012; Gombolay *et al.*, 2013]. Problems when tasks can be completed only with multiple agents are addressed by many (i.e., [Zheng and Koenig, 2008; Ramchurn *et al.*, 2010]), but all assume task costs are fixed and known. Hard deadlines, or time windows, are temporal constraints where tasks have to be completed within a specified start and end time. Melvin et al. [2007] describe efficient auction based methods when time windows and rewards for task completion are known, but assume disjoint time windows to create a total ordering of the tasks. We cannot assume disjoint time windows.

3 Problem Definition

Our problem is to assign heterogeneous agents to tasks which have a cost that grows over time. We make no assumptions on the spatial locations of agents or tasks other than an agent must be at a task's location in order to work on that task. Tasks do not require multiple agents, but multiple agents decrease completion time. Since task costs grow over time, if too few agents are allocated to a task the cost will continue to grow and the task might become impossible to complete. We assume the task growth is known or can be estimated.

We denote the set of heterogeneous agents by $A = \{a_1, \ldots, a_{|A|}\}$ and the set of tasks by $B = \{b_1, \ldots, b_{|B|}\}$. 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 t, without counting any of the agents which are in transit at time 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. The travel time for agent a, $TT(a, x, y)$, between two locations, x and y, is assumed to be computable.

Each agent a provides the amount of work w_a per time unit when the agent is at a task location. An agent traveling to a task does not provide any work. Every task $b_i \in B$ has a cost defined with the following recurrence relation [Parker and Gini, 2014]:

$$f_i^{t+1} = f_i^t + h_i(f_i^t) - \sum_{a \in n_i^t} w_a, \tag{1}$$

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. We use Euler's method as the growth rate depends on both the agents and task size, which is similar to solving a differential equation. It has been shown that the global truncation error of the Taylor series is no more than $\frac{M\Delta T}{2 \cdot L}(e^{t_f - t_0} - 1)$, where M is the upper bound on the derivative of Δf_i^t and L is the Lipschitz constant on the interval $[t_0, t_f]$ with a step size of ΔT [Atkinson, 2008]. Here we treat f_i^t as a sequence due to the use of discrete time steps, but it could be treated as a continuous function if that is a better model for the domain (Sec. 4.1).

If at some time t the cost of task b_i, namely f_i^t, reaches or goes below zero, we denote this time as ct_i for the task completion time. Thus, when f_i^t is non-positive, $h_i(f_i^t)$ is set to zero and agents cannot be assigned to this task, namely $|n_i^t| = 0$ when $t > ct_i$. When each task has $f_i^t \leq 0$, the problem is considered solved. When $h_i(f_i^t) > \sum_{a \in n_i^t} w_a$ this means f_i^t is strictly monotonically increasing, which means the task is growing faster than the assigned agents can reduce it. If more agents are not assigned at a later time, the task will never be completed and we say $ct_i = \infty$.

The goal is to complete all tasks with as little growth as possible. This is different from [Parker and Gini, 2014], which only considers the time when the last task is finished. In forest fires, this would correspond to minimizing the number of trees burnt, rather than how quickly the fire is put out. Formally, our goal is to minimize the sum of the accumulated growth cost, R_{ct_i}, across all tasks:

$$\min \sum_{b_i \in B} R_{ct_i}, \text{ where } R_{ct_i} = \sum_{t < ct_i} h(f_i^t). \tag{2}$$

4 Task Growth Functions

We first examine the case when travel time is instantaneous and consider various types of task growth functions, $h_i(x)$ from (1), along with their optimal allocation. For simplicity we assume all tasks have the same growth function $h(x)$, specifically: $\forall i \quad h_i(x) = h(x)$. These relaxations are purely used to develop the theoretical formulation of this problem. The algorithms in this paper do not rely on these assumptions.

The travel time relaxation might seem to trivialize the problem, but the optimal solutions to the relaxed problems provide an intuition on desirable types of allocations. For example, consider one agent and two tasks b_1 and b_2 such that $f_1^0 < f_2^0$, namely b_2 is initially larger than b_1. Since $h(x)$ is defined to be monotonically increasing, b_2 grows faster than b_1. If the agent tries to do b_1 first, b_2 will grow even larger, which will in turn increase the rate of growth, and make it even harder to tackle b_2 later. However, if the agent starts on b_2 then it will take a longer time to finish b_2, and this will give b_1 more time to grow and become harder to complete. As we will show in the next few sections, the key to decide how to allocate agents to tasks is the second derivative of the growth function $h(x)$.

As in [Parker and Gini, 2014], we minimize the regret $\sum_{b_i \in B} R_{ct_i}$ (2). Next we show how $\sum_{b_i \in B} R_{ct_i}$ can be minimized by greedily minimizing $\sum_{b_i \in B} h(f_i^t)$ at each time step. We prove this when $h(x)$ is linear or monotonically decelerating, as the monotonically accelerating case was proven in [Parker and Gini, 2014]. In addition to the proofs, we provide an intuition on the general patterns of optimal solutions if travel time is not zero.

4.1 Linear

When $\frac{\partial^2}{\partial x^2} h(x) = 0$, $h(x)$ is linear. In this case the recurrence relation in (1) has a closed form solution. If we write $f_i^{t+1} = f_i^t + p \cdot f_i^t + q - \sum_{a \in n_i^t} w_a$, where n_i^t is assumed to be constant for the duration of the projection, then this has a closed form solution of $f_i^t = K_i + L_i \cdot (p+1)^t$. Here K_i and L_i are constants determined by the initial conditions of f_i^t and the number of agents assigned. We can solve for the initial conditions and get[1]:

$$f_i^t = -\frac{q - \sum_{a \in n_i^t} w_a}{p} + (f_i^0 + \frac{q - \sum_{a \in n_i^t} w_a}{p}) \cdot (p+1)^t \tag{3}$$

If we let the step size shrink to zero (along with p and w), we get the well known continuous exponential function:

$$f_i^t = \frac{1}{p} \cdot (\sum_{a \in n_i^t} w_a) + D \cdot e^{p \cdot t}, \tag{4}$$

where again D is an integration constant and it will be negative if the agents are completing the task faster than it is growing, namely $D < 0$ if and only if $\sum_{a \in n_i^t} w_a > p \cdot f_i^t$, as $D = f_i^0 - \frac{1}{p} \cdot (\sum_{a \in n_i^t} w_a)$. We let $q = 0$ for continuity of the derivative when a task finishes.

We show that $\sum_{b_i \in B} h(f_i^t)$ is minimized by any assignment of all agents, as it is a constant. Thus $\sum_{b_i \in B} R_{ct_i}$ is

[1] If $p = 0$, then $f_i^t = f_i^0 + (q - \sum_{a \in n_i^t} w_a) \cdot t$.

also a constant. The proof will be direct and can be intuitively thought of as using the linearity to merge all tasks into a single meta-task, which is the direct sum of each individual tasks. Since there is a single task all agents will be assigned to it at every time step. This will allow us to derive an explicit formula for both $\sum_{b_i \in B} R_{ct_i}$ and $\sum_{b_i \in B} h(f_i^t)$.

Theorem 1. *If the travel time is zero, all agents are assigned to an active task, and $h(x)$ is linear, then $\sum_{b_i \in B} R_{ct_i}$ and $\sum_{b_i \in B} h(f_i^t)$ are constants and are directly computable for any assignment.*

Proof. First we will directly show that $\sum_{b_i \in B} h(f_i^t)$ is a constant via direct proof. Since $h(x)$ is linear, $\sum_{b_i \in B} h(f_i^t) = h(\sum_{b_i \in B} f_i^t)$. Summing over i in (3), we get:

$$\sum_{b_i \in B} f_i^t = -Z + (\sum_i f_i^0 + Z) \cdot (c+1)^t,$$

where $Z = \frac{1}{p}(q \cdot |B| - \sum_{a \in A} w_a)$ since $\sum_{b_i \in B} d = d \cdot |B|$ and $\sum_{b_i \in B} \sum_{a \in n_i^t} w_a = \sum_{a \in A} w_a$ as every agents is assigned to a task. As no assumptions were made about the individual assignments n_i^t, we can conclude that $\sum_{b_i \in B} h(f_i^t)$ is a constant at time t for any assignment, namely:

$$-q \cdot (|B|-1) - \sum_{a \in A} w_a + (p \cdot \sum_i f_i^0 + q \cdot |B| - \sum_{a \in A} w_a) \cdot (p+1)^t.$$

Since this is constant, it is also a minimum for all assignments. As $\sum_{b_i \in B} R_{ct_i} = \sum_{t < \max(ct_i)} \sum_{b_i \in B} h(f_i^t)$ and $\sum_{b_i \in B} h(f_i^t)$ is a constant, it follows that $\sum_{b_i \in B} R_{ct_i}$ is also a constant. □

When the task growth function is linear, the benefit for doing a task is no longer determined by the size of that task, but instead by the size of the sum of all tasks. In cases with non-instantaneous travel time, we want to increase the amount of time agents spend working, as it does not matter which task they are working on. The cost of tasks still grows exponentially, so a poor initial allocation of agents is worse than poor allocations near completion. This does not quite reduce the problem to finding a minimum distance solution, but this would be close to optimal.

One can also easily determine whether all tasks can be completed or not with zero travel time. If $h(\sum_{b_i \in B} f_i^t) \geq \sum_{a \in A} w_a$ then the problem is no longer solvable, otherwise all tasks can be completed. With zero travel time, this condition needs to be checked only once to determine the outcome. If there is travel time, the right hand side of the equation is an overestimate, but it can still be evaluated at every time step to determine if the problem is no longer solvable.

4.2 Monotonic Deceleration

The next case we analyze is when $\frac{\partial^2}{\partial x^2} h(x) \leq 0$, and is not necessarily constant. $h(x)$ is still required to be monotonically increasing, so f_i^t is $\Omega(x)$ and $O(e^x)$. For this family of functions, the rate of change in growth is faster for smaller tasks. So while larger tasks still grow faster than smaller tasks, reducing smaller tasks will shrink their future growth more than for larger tasks. This is the opposite of the case when $\frac{\partial^2}{\partial x^2} h(x) > 0$, where task growth has a positive feedback loop.

To prove that minimizing the growth of all functions at every time step is the optimal solution, we examine a pair of tasks and show by contradiction that the optimal solution must assign all agents to the smallest task. Since the pairs of tasks are general, this reasoning can be applied to all pairs.

Theorem 2. *If the travel time is zero, all agents are assigned to an active task and $\frac{\partial^2}{\partial x^2} h(x) \leq 0$, then $\sum_{b_i \in B} R_{ct_i}$ is minimized by greedily reducing $\sum_{b_i \in B} h(f_i^t)$ at every step, which is done by assigning all agents to the smallest task.*

Proof. Suppose we have two tasks b_1 and b_2, where without a loss of generality $f_1^0 < f_2^0$. First we show that minimizing $\sum_{b_i \in B} h(f_i^t)$ is achieved by assigning all agents to b_1. We want to assign an agent to the tasks in order to decrease the overall growth as much as possible, namely

$$\operatorname*{argmin}_{b_i \in B} h(f_i^t - w_a) - h(f_i^t).$$

If we divide by $-w_a$, a constant, we would want to maximize $\frac{h(f_i^t - w_a) - h(f_i^t)}{-w_a}$. This is a rough approximation of the derivative, and since $h(x)$ is decelerating, $\frac{\partial}{\partial x} h(x)$ is maximized for small f_i^t. As each agent is assigned to b_1, f_1^t will shrink more and cause $\frac{\partial}{\partial x} h(x)$ to become even larger. Thus $\sum_{b_i \in B} h(f_i^t)$ is minimized by assigning all agents to b_1. An exception to this is if b_1 completes, then $h(f_1^t) = \frac{\partial}{\partial x} h(f_1^t) = 0$, so remaining agents will go to b_2.

Next we represent $\sum_{b_i \in B} R_{ct_i}$ between b_1 and b_2 as $T_1 + T_2 + \Delta T_1 + \Delta T_2$, where T_i is the growth accumulated for completing b_i with all agents assigned to it initially and ΔT_i is accumulated growth from not assigning all agents. That is $\Delta T_i = 0$ if all agents are allocated to b_i and $T_i = \sum_{t \leq ct_i} h(f_i^t)$ where $|n_i^t| = |A|$ until b_i is completed. By definition, assigning all agents to b_1 until it is completed will eliminate ΔT_1 and increase ΔT_2 as much as possible. Suppose there exists some smaller $\sum_{b_i \in B} R_{ct_i}$ denoted by hats with $\Delta \hat{T}_1 \neq 0$. This implies that $\Delta \hat{T}_1 + \Delta \hat{T}_2 < \Delta T_2$, however this means that agents can reduce the growth of b_2 faster than b_1. This is a contradiction with what we have shown earlier, since $h(x)$ is decelerating and $f_1^t < f_2^t$.

For $|B| > 2$, we can compare all possible pairs of tasks. Since $f_i^0 < f_j^0$ implies all agents should be allocated to b_i over b_j, we can conclude that all agents should be allocated to $\operatorname*{argmin}_{b_i \in B} f_i^0$, until this task finishes. We will then remove the finished task from B and search for the next smallest task and repeat the process. If there is a tie for smallest, either one can be picked and assigned to the first available agent. After the first agent is assigned, that task will be smaller and the tie is broken. □

For this family of functions, the optimal solution is fairly intuitive. Since $h(x)$ is decelerating, smaller tasks have a much larger relative increase in size. Additionally, agents can reduce smaller task more than larger tasks, as $h(x)$ is monotonically increasing. This creates a positive feedback loop of incentives for agents to work on the smaller task. We can find

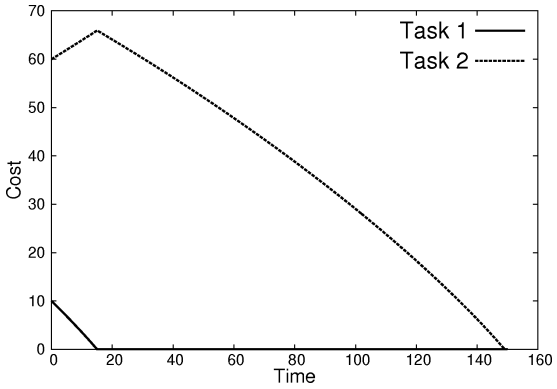

Figure 1: Optimal solution when $TT(a, x, y) = 0, f_1^0 = 10, f_2^0 = 60, |A| = 65, w = 0.01201, h(x) = 0.05 \cdot \sqrt{x}$.

whether all tasks can be completed for a monotonically decelerating function by repeatedly estimating the time to complete the smallest undone task, then projecting the growth of all the remaining tasks. If at some point the projection of the remaining task b_i at time t satisfies $h(f_i^t) > \sum_{a \in A} w_a$, then the problem is not solvable.

Figure 1 shows an example of an optimal zero travel time solution, where all the agents are assigned to task 1 until it is completed. The important aspect is that it took under 20 time steps to complete 10 units of work on task 1 and during this time task 2 only increased about 6 units. Task 2 then took over 132 time steps to do 66 units of cost.

5 Heterogeneous agents

Each heterogeneous agent a has a work amount w_a, that the agent can apply to a task. The size of a work unit can be the greatest common divisor of all the agents' work amounts. We place no restrictions on the travel speed of agents, but this impacts how long an agent is not working on a task, thus is not in any n_i^t. Solutions to the allocation problem depend critically on travel time. We have provided (Section 4) optimal solutions for specific classes of growth functions when travel time is zero. We are now ready to consider cases when travel time is prohibitively large, hence agents are allocated at the beginning and never reallocated, and cases where travel time is non zero but not prohibitively large, hence reallocation of agents might be useful.

First we solve a variation of the Variable-Sized Bin-Packing-Problem [Correia *et al.*, 2008; Wäscher *et al.*, 2007] once to find an initial solution when travel times are prohibitively large. Then we present an incremental heuristic for reassigning agents when travel time is not prohibitive.

5.1 Heterogeneous LFF

We modified the LFF algorithm [Parker and Gini, 2014] to handle heterogeneous agents. LFF is a centralized heuristic algorithm, which produces stable assignments. LFF is a one-shot algorithm which generates an optimal solution when reassigning agents to a new task is prohibitively expensive. An example domain for which these conditions hold is remote forest fires, where a small team of firefighters deploy

input : Agents A and tasks B
output: Real assignments W

for $i \leftarrow 1$ **to** $|B|$ **do**
  set $W_i^* = \infty$
end
$totalUnits \leftarrow \frac{\sum_{a \in A} w_a}{\text{1 work unit amount}}$
for $i \leftarrow 1$ **to** $totalUnits$ **do**
  $chosen \leftarrow \text{argmax}_i \, W_i^*$
  `Assign` 1 work unit to $chosen$
  Recompute ct_{chosen} via simulation.
  Recompute W_{chosen}^* as $R_{ct_{chosen}}$ in (2)
end
Solve for W and I for all tasks by mixed-integer linear program
for *a not assigned in* I **do**
  `Assign` largest a to next largest $W_i^* - W_i$
  Update I and W
end

Algorithm 1: Heterogeneous LFF (HLFF)

via parachute close to the fire. This initial deployment has a fairly uniform cost, as reaching any drop point takes a similar amount of fuel. However, once on ground it is infeasible for the firefighters to traverse the terrain quickly to get to another section of the fire.

LFF works by assigning agents one at a time to the task which currently finishes last. This is repeated until there are no more agents left to assign. In our problem we are concerned with cost and not with finish time, so we prioritize assignments to the task which has the largest accumulated growth, $b^* = \text{argmax}_{b_i} R_{ct_i}$. Instead of assigning a whole agent, we assign a single work unit to b^* and recompute the new value of $R_{ct_{b^*}}$, as detailed in Alg. 1. Often minimizing the finish time also minimizes $R_{ct_{b^*}}$, but we chose the latter to be consistent with the theoretical proofs.

When all work units have been assigned to the ideal assignment W^*, we compute a real work assignment, $I_{i,j}$, of agent a_i to task b_j and the corresponding work amounts w_{a_i} for task W_j. To do this, we solve a mixed integer linear program for the Variable-Sized Bin-Packing-Problem as follows [Correia *et al.*, 2008]:

$$\begin{array}{lll} \max \sum_{b_i \in B} W_i \,, & \text{subject to:} & \\ \text{(a)} \quad \forall b_j \in B: & \sum_{a_i \in A} w_{a_i} \cdot I_{i,j} - W_j = 0 & \\ \text{(b)} \quad \forall a_i \in A: & \sum_{b_j \in B} I_{i,j} \leq 1 & \\ \text{(c)} \quad \forall b_j \in B: & 0 \leq W_j \leq W_j^* & \\ \text{(d)} \quad \forall i, j: & I_{i,j} \in \{0, 1\} & \end{array}$$

where the first line is the optimization objective, i.e. maximize the amount of work units packed into each task. w_a is the work amount of agent a, and $I_{i,j}$ is an indicator variable signifying agent a_i is assigned to task b_j. W_j is the amount of work assigned to task b_j and W_j^* is the ideal amount of work for task b_j. (a) specifies agent assignment to tasks, (b) ensures that agents are only assigned to a single task, and (c) specifies that tasks cannot receive more work than their ideal amount. This mixed integer linear program only needs to be solved once per simulation. It can solve for 300 agents and 2 tasks in under 0.2 seconds.

Since an agent has to be assigned in full to a task, at times some tasks cannot receive the ideal amount of work. This means some agents will not have an assignment (i.e. some a_i might have $\sum_{b_j \in B} I_{i,j} = 0$). After the mixed-integer linear program runs, we assign tasks to agents that are not assigned. This takes $O(|A|)$ time and is done by repeatedly assigning the unassigned agent with the largest number of work units to the task with the largest difference between the ideal and the current work unit amount. If there are insufficient agents to finish all tasks, some tasks will not receive enough work.

5.2 Heterogeneous RT-LFF

While HLFF maximizes an agent's work to travel time ratio, the assignments are not optimal if agents can relocate to different tasks. HLFF is also a one-shot assignment, which can be detrimental if new tasks can appear or the growth model is inaccurate. Parker and Gini [2014] proposed a heuristic algorithm, RT-LFF, to reassign agents as needed. We extend RT-LFF to Heterogeneous RT-LFF (HRT-LFF).

RT-LFF compares each pair of tasks (b_i, b_j). If $ct_i < ct_j$ even after transferring an agent from b_i to b_j, then this transfer happens. This can only decrease the overall finish time across all tasks. This heuristic is conservative. To see why, let the completion times change from ct_i and ct_j to ct_i^* and ct_j^* respectively after the transfer. It is possible that $ct_i < ct_j^* < ct_i^* < ct_j$. This is a better solution as $\max(ct_i, ct_j) > \max(ct_i^*, ct_j^*)$, yet using the heuristic task b_i will not give up an agent.

In the original formulation, the agent to transfer was simply the closest one to the other task. With heterogeneous agents, the order in which agents are checked for transfer can lead to significant differences in the accumulated cost. If two agents have identical work amounts, then the faster one is always the better one to transfer. Thus, there is no reason to consider transferring slow agents unless there are no more faster ones.

Using these observations, we considered the following strategies for the HRT-LFF heuristic: (1) agents with the highest speed, (2) agents with the lowest work amount, and (3) agents with the highest speed to work ratio. A weighted combination of these three strategies might work better for specific problems, but (3) worked the best on average on tested problems.

Empirical results showed that there was not a significant difference between checking just the highest speed to work ratio agent compared to checking all agents in decreasing speed to work ratio order. Thus only a single agent is checked in Alg. 2. We check all pairs of tasks to see if any transfer is possible, but the order in which the pairs are checked has an effect on the outcome. Tasks compared earlier are more likely to get an agent transfered to them than tasks compared later. When a new task appears, it should try to get an agent from all other tasks first. Since our goal is to minimize $\sum_{b_i \in B} R_{ct_i}$, we sort task pairs (b_i, b_j) based on the difference $R_{ct_j} - R_{ct_i}$. The largest differences are tested first for possible transfers, until the difference is non-positive, in which case a transfer will never happen and the loop ends. As we need to keep track of the best agent to pick, this increases the run time from $O(|B^2|)$ in RT-LFF to $O(B^2 \ln A)$ in the heterogeneous case.

input : Agents A, tasks B, current task assignments
output: New assignments

for *every pair of tasks:* $b_i \neq b_j$ **do**
  $a^* \leftarrow \text{argmax}_{a \in n_i^t} \frac{1}{w_a \cdot TT(a, b_i, b_j)}$
  $ct_i^* \leftarrow$ simulate task with $n_i^t - a^*$ active agents
  $\hat{t} \leftarrow t + TT(a^*, b_i, b_j)$
  $ct_j^* \leftarrow$ simulate task with n_j^t active agents from t to $\hat{t}$ and $n_j^t \cup a^*$ agents onwards
  Compute $R_{ct_i^*}$ and $R_{ct_j^*}$
  if $R_{ct_i^*} < R_{ct_j^*}$ **then**
    `Assign` a^* from b_i to b_j
  end
end

Algorithm 2: Heterogeneous RT-LFF (HRT-LFF)

6 Results

For our experiments, we use the RoboCup Rescue simulator, which was designed for urban search and rescue after an earthquake. The environment is very large with upward of 100 agents. The simulator uses different types of agents, but for this work we focus only on agents that can extinguish fires (i.e. firetrucks) through the use of the RMASBench simulator extension [Kleiner *et al.*, 2013].

Fires are the main hazard in RoboCup Rescue. Buildings heat up and catch on fire based on how many other nearby buildings are on fire. This creates a positive feedback loop, which causes mores fires to grow more rapidly. Screenshots of the simulator are in Figure 2 and Figure 3. Red dots represent fire trucks, dark gray polygons are buildings while light gray polygons are roads. Buildings on fire are yellow, orange and red in increasing intensity and temperature. If a building burns too long and is destroyed, it will turn black. When a building is extinguished, it becomes blue or purple.

In the RoboCup Rescue simulator, individual buildings heat up and catch fire. Since buildings eventually burn out or re-ignite, instead of modeling fires in individual buildings we model fires in clusters of buildings. A cluster is considered a single task, where the cost is the number of buildings on fire in the cluster. We do this clustering using bottom-up hierarchical clustering with the Euclidean distance as the metric and the minimum distance between all pairs linkage criteria. If the distance between the closest pair of clusters is over 50 meters in the simulation, then the clustering would cease and the clusters left would be the tasks.

An exponential function has been shown to be a good estimate for the number of buildings on fire in this domain [Parker and Gini, 2014], thus we approximate task costs by (4). To estimate p, we allowed buildings to burn unhindered for 100 simulation steps in 20 different tests. and we used exponential regression to find the best fit with the data.

The work rate, w_a, was empirically derived for three classes of agent: agents who could use their full hose capacity, agents who could only use only 50% of it, and agents who could only use 10% of it. A fixed small number of fires were repeatedly extinguished in 70 tests for each type of agent. Putting f_i^0, p, n_i^t and ct_i into (4), we can solve for w_a.

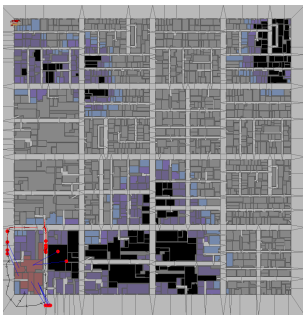

Figure 2: HRT-LFF in VC 3 ending with 69% of buildings intact.

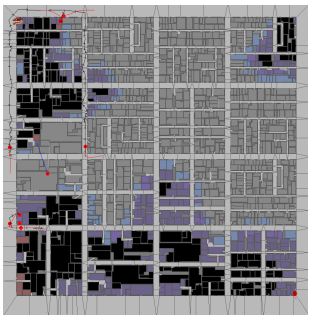

Figure 3: $\widetilde{\text{RT-LFF}}$ in VC 3 ending with 59% of buildings intact.

Table 1: Percent of buildings intact at the end of simulation on different versions of Virtual City and Paris maps

Map	HRT-LFF		$\widetilde{\text{RT-LFF}}$		ALLONEONE		UNIFORM	
	μ	σ	μ	σ	μ	σ	μ	σ
VC 1	**33.34**	**8.97**	31.73	7.77	30.64	9.42	22.77	7.10
VC 2	**81.71**	**0.66**	81.38	0.74	81.58	0.58	81.10	0.83
VC 3	**65.35**	**9.90**	59.24	7.31	40.24	12.12	55.71	9.85
VC 4	75.25	2.50	**75.34**	**2.32**	74.87	1.75	74.74	2.57
VC 5	**33.52**	**4.48**	24.23	3.13	24.20	2.34	23.55	2.20
Paris 1	**88.70**	**3.58**	87.84	3.40	85.28	4.78	85.40	4.02
Paris 2	**45.12**	**8.67**	42.16	7.91	35.54	9.82	26.40	12.00
Paris 3	**68.39**	**3.57**	67.95	4.74	64.23	5.35	64.43	4.52
Paris 4	**91.78**	**0.35**	91.53	0.42	91.70	0.39	91.39	0.45
Paris 5	**85.89**	**3.26**	83.77	3.23	74.43	8.07	79.19	5.06

To compare HRT-LFF against other algorithms, we modified both the original LFF and RT-LFF to consider accumulated growth cost instead of finishing time. We call this modified algorithms $\widetilde{\text{LFF}}$ and $\widetilde{\text{RT-LFF}}$, respectively. HRT-LFF starts with the solution to the mixed integer linear program, while $\widetilde{\text{RT-LFF}}$ starts with an initial assignment produced by $\widetilde{\text{LFF}}$. $\widetilde{\text{RT-LFF}}$ checks pairs of agents in order of their set identification (a_1 before a_2, etc.). The UNIFORM strategy simply calculates the total work by agents as $\sum_{a \in A} w_a$ and solves the Variable-Sized Bin-Packing-Problem to evenly divide the work amount between all not finished tasks. The ALLONEONE strategy assigns all the agents to task b_1 until it is completed, then assigns all the agents to b_2 and so on.

Table 1 shows the percent of intact buildings for 5 variations of the Virtual City (VC) and Paris map, where each configuration was run 5 times with the average being displayed. Note that a higher percent of buildings intact is desirable, which satisfies our goal of a low accumulated cost. Since $h(x)$ and w_a are only approximations, we cannot compute the optimal solution in this case. Each configuration ranged between 2-4 100% capability agents, 5-8 50% capability agents and 10-20 10% capability agents. Each map was seeded with 2-4 initial fires and we let 30 simulation time steps pass before agents could move. This ensured that fires were of a moderate size when agents started.

HRT-LFF can reason about heterogeneous agents, thus it scores better on average than $\widetilde{\text{RT-LFF}}$. Both algorithms try to maintain a balanced stable assignment, thus typically their assignments are similar. UNIFORM and especially ALLONEONE suffer from assignment thrashing in this domain. Even after a building cluster has been extinguished, some buildings are still hot. This can cause small fires to restart a few time steps after the task seems to be completed. Both UNIFORM and ALLONEONE will send more agents than needed for this small task, which can cause a substantial increase in travel time. For this reason, ALLONEONE does especially poor when there are 4 separate tasks, as each fire could start again a few times and cause all the agents to turn around and go back. Even if UNIFORM does not send all the agents back, it still sends more than necessary. HRT-LFF and $\widetilde{\text{RT-LFF}}$ in this case only send one agent back, as they can reason that this will be enough. HRT-LFF can reason with finer granularity and thus sends back one or two 10% capability agents, leaving the rest to work on other tasks.

Some results, such as VC 5, are poor across all algorithms. This indicates more the hardness of the configuration rather than the efficiency of the algorithms. The opposite case is in Paris 4, where the configuration is too easy and all algorithms score identically. Despite these extreme cases, the results show the efficiency of the algorithms in general.

As noted in [Parker *et al.*, 2014], the bimodal distribution of the results makes it difficult to infer statistical significance. For this reason we apply the non-parametric Wilcoxon signed-rank test instead of the normal t-test. HRT-LFF outperforms the other algorithms on all maps except VC 4, which has the smallest difference in score between HRT-LFF and $\widetilde{\text{RT-LFF}}$. Thus HRT-LFF has a p-value of 0.005063 when compared against ALLONEONE or UNIFORM. HRT-LFF compared against $\widetilde{\text{RT-LFF}}$ has a p-value of 0.006911. Thus HRT-LFF is better than the other algorithms by a statistically significant amount.

7 Conclusions and Future Work

We proposed the first solution for assigning heterogeneous agents to tasks whose costs grow over time. Two algorithms were presented, one which assumes prohibitive travel times and does not reallocate agents, the other which accounts for travel time and inaccuracies in the prediction model. The performance of the latter algorithm was analyzed in a complex simulation environment, where the growth function had to be estimated. Optimal solutions were proven in a travel time relaxation of our core problem. In the future we intend to extend our work to a decentralized solution. We are also interested in studying the performance of algorithms when there is noise or errors in the growth function.

Acknowledgments: Partial support for this work is acknowledged from the National Science Foundation under grant NSF IIP-1439728 and the Graduate School of the University of Minnesota.

References

[Amador *et al.*, 2014] Sofia Amador, Steven Okamoto, and Roie Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proc. Nat'l Conf. on Artificial Intelligence*, pages 1384–1390, 2014.

[Atkinson, 2008] Kendall E Atkinson. *An introduction to numerical analysis*. John Wiley & Sons, 2008.

[Atlas and Decker, 2010] James Atlas and Keith Decker. Coordination for uncertain outcomes using distributed neighbor exchange. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1047–1054, 2010.

[Beyer and Sendhoff, 2007] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33):3190–3218, 2007.

[Correia *et al.*, 2008] Isabel Correia, Luís Gouveia, and Francisco Saldanha-da Gama. Solving the variable size bin packing problem with discretized formulations. *Computers & Operations Research*, 35(6):2103–2113, 2008.

[Gombolay *et al.*, 2013] Matthew Gombolay, Ronald Wilcox, and Julie Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Proc. Robotics: Science and Systems (RSS)*, Berlin, Germany, June 2013.

[Kitano and Tadokoro, 2001] Hiroaki Kitano and Satoshi Tadokoro. RoboCup Rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39–52, 2001.

[Kleiner *et al.*, 2013] Alexander Kleiner, Alessandro Farinelli, Sarvapali Ramchurn, Bing Shi, Fabio Maffioletti, and Riccardo Reffato. RMASBench: Benchmarking dynamic multi-agent coordination in urban search and rescue. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1195–1196, 2013.

[Kruijff *et al.*, 2012] G.-J.M. Kruijff, V. Tretyakov, T. Linder, F. Pirri, M. Gianni, P. Papadakis, M. Pizzoli, A. Sinha, E. Pianese, S. Corrao, F. Priori, S. Febrini, and S. Angeletti. Rescue robots at earthquake-hit Mirandola, Italy: A field report. In *IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–8, Nov 2012.

[Léauté and Faltings, 2011] Thomas Léauté and Boi Faltings. Distributed constraint optimization under stochastic uncertainty. In *Proc. Nat'l Conf. on Artificial Intelligence*, pages 68–73, 2011.

[Melvin *et al.*, 2007] J. Melvin, P. Keskinocak, S. Koenig, C. Tovey, and B.Y. Ozkaya. Multi-robot routing with rewards and disjoint time windows. In *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 2332–2337, Oct 2007.

[Mills-Tettey *et al.*, 2007] G. Ayorkor Mills-Tettey, Anthony Stentz, and M. Bernardine Dias. The dynamic hungarian algorithm for the assignment problem with changing costs. Technical Report CMU 7-2007, Carnegie-Mellon University, 2007.

[Nam and Shell, 2015] Changjoo Nam and Dylan A. Shell. When to do your own thing: Analysis of cost uncertainties in multi-robot task allocation at run-time. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2015.

[Nunes *et al.*, 2012] Ernesto Nunes, Maitreyi Nanjanath, and Maria Gini. Auctioning robotic tasks with overlapping time windows. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1211–1212, 2012.

[Parker and Gini, 2014] James Parker and Maria Gini. Tasks with cost growing over time and agent reallocation delays. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 381–388, 2014.

[Parker *et al.*, 2014] James Parker, Julio Godoy, William Groves, and Maria Gini. Issues with methods for scoring competitors in RoboCup Rescue. In *Autonomous Robots and Multirobot Systems at AAMAS*, 2014.

[Ramchurn *et al.*, 2010] Sarvapali Ramchurn, Alessandro Farinelli, Kathryn Macarthur, Mariya Polukarov, and Nick Jennings. Decentralised coordination in RoboCup Rescue. *The Computer Journal*, 53(9):1–15, 2010.

[Toth and Vigo, 2002] P. Toth and D. Vigo, editors. *The vehicle routing problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, PA, 2002.

[Wäscher *et al.*, 2007] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[Zhang and Parker, 2013] Yu Zhang and Lynne E. Parker. Considering inter-task resource constraints in task allocation. *Journal of Autonomous Agents and Multi-Agent Systems*, 26:389–419, 2013.

[Zheng and Koenig, 2008] Xiaoming Zheng and Sven 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.