Throwing in the Towel: Faithless Bounty Hunters as a Task Allocation Mechanism

Drew Wicke **Ermo Wei** **Sean Luke**
Department of Computer Science, George Mason University
4400 University Drive MSN 4A5
Fairfax VA 22314 USA
{dwicke, ewei, sean}@cs.gmu.edu

Abstract

Bounty hunting has been shown to solve the multiagent task allocation problem robustly in noisy and dynamic scenarios with multiple other agents of unknown quality. This bounty hunting model does not require task exclusivity, but does require task commitment. We examine eliminating this second requirement, thus allowing bounty hunters to commit to tasks but abandon them later (to *jump ship*) for more promising tasks, without telling other agents of their intent. Bounty hunting agents must use an adaptive valuation mechanism in order to avoid too much redundancy in working on tasks. We examine how one might revise this mechanism in the face of task abandonment. We compare jumping ship favorably against other bounty hunting methods and against methods which are more "auction-like" in that they permit exclusivity, and we do so under both static environments and ones in which agents, and task scenarios, change dynamically.

Introduction

In *dynamic multiagent task allocation*, some *N* agents are tasked with collectively deciding how to divvy up tasks that appear dynamically. There is no central task assignment system, and the agents act independently and under certain communication constraints. Creating a globally efficient mechanism arising from the task assignment decisions of individual agents can be difficult. Many distributed methods have been *auctions*, where each agent bids for tasks they would prefer to do. When a task is available, an auctioneer offers it to the agents, who bid according to their own individual valuations of the task. After a clearing time has passed, the winning agent is offered the task, and it is normally exclusive to him.

Dynamic multiagent task allocation was formally described in Lerman *et al.* [2006]. The area of multiagent task allocation has seen numerous surveys, which provide detailed taxonomies of the task allocation literature [Gerkey, 2004; Khamis *et al.*, 2015; Korsah *et al.*, 2013]. Some early approaches using markets have been explored as a means to solve the task allocation problem [Botelho and Alami, 1999; Pustowka and Caicedo, 2012; Walsh and Wellman, 1998] as well as multiagent exploration [Simmons *et al.*, 2000; Zlot *et al.*, 2002]. Current solution methods include centralized approaches using integer programming and combinatorial optimization; and decentralized approaches like auctions, markets, task swapping and bounty hunting [Lagoudakis *et al.*, 2004; Liu *et al.*, 2014; Liu and Shell, 2012; Nanjanath and Gini, 2006; Wicke *et al.*, 2015]. Auction methods such as MURDOCH [Gerkey and Matarić, 2002], CoMuTaR [Shiroma and Campos, 2009], traderBots [Dias, 2004; Jones *et al.*, 2006], and repeated auctions [Nanjanath and Gini, 2010] are frequently studied in task allocation literature. In some cases adaptation has been explored from the auctioneer side, [Pippin and Christensen, 2012].

As argued in Wicke *et al.* [2015], auctions are a strange fit for this problem for a variety of reasons, but we repeat two here: because dynamic task allocation requirements usually resort to outfitting agents with multi-task queues or clearing times; and because the task exclusivity of an auction makes it a poor performer in environments with significant noise. Instead, they proposed using *bounty hunting* as a more robust mechanism for dynamic task allocation. This model is loosely based on the tradition of bounty hunters (the agents) competing to complete tasks in return for gradually increasing rewards posted (bounty) by a bail bondsman.

In a bounty hunting model, as tasks become available, they are assigned a gradually-increasing *bounty* which an agent will win if he completes the task before any other agent does. Normally there is no exclusivity; multiple agents may compete for the same task. There is however, the notion of *commitment*: an agent must commit to a task before he can begin work on it, an agent may commit to no more than one task at a time, and once an agent has committed to a task, he may not abandon that task until either he, or some other agent, has completed it. If an agent completes a task, he receives a reward equal to the bounty posted when he originally committed to the task. Note that this doesn't cause the lock-in problems that exclusivity does: if an agent has committed to a task but cannot complete it, eventually the bounty on the task will rise to the point that some other agent will jump in and do the task instead.

All this allows agents to work in environments with other agents of *unknown capability, heterogeneity, number,* or even *existence.* But because agents may work on the same task, a bounty hunting model can be inefficient unless the agents can adapt to one another so as to pursue tasks that each is likely to win. Thus the previous bounty hunting work concerned itself with adaptive valuation methods to do this. They showed that with such a valuation method, bounty hunting is competitive with auction-like methods in static and dynamic environments, and significantly outperforms them in noisy scenarios such as unreliable collaborators or unexpectedly bad tasks. Another possible way to improve efficiency is to allow agents to abandon a task (to *jump ship*, so to speak). This might reasonably happen when a task turns out to be unexpectedly difficult for an agent to pursue, or because another, more capable agent has committed to compete on the task. Unfortunately, our previous experimental work in jumping ship used a very simple decision model, and the results were not promising.

In this paper we revisit the notion of abandoning tasks. We ask: under what circumstances is it actually more efficient to allow agents to give up on a task rather than wait for another agent to steal it from them? To do this, we first review the bounty hunting model, as well as issues such as valuation, exclusivity, and so on. We then propose a much more sophisticated model to learn when to jump ship. We then recreate the experimental setup of the original paper, and compare our jump-ship method to previously studied methods using the original five experiments. We also examine these methods in two new scenarios: one where the bounty rate is non-homogenous among the tasks, and one where certain task classes do not appear until well after learning has converged.

Our empirical results are surprising. Abandoning tasks is effective in our two new scenarios, but they were devised to demonstrate the obvious corner cases where it *ought* to work well. However we have found that abandoning tasks *also* performs as well as, and usually better than, all other methods in six out of seven experiments, including our pseudo-auction method. This suggests that jumping ship can be very effective in combination with adaptive valuation.

Model Description

We will use roughly the same model and notation as Wicke *et al.* [2015]. There is a single bondsman[1] and a set of one or more bounty hunter agents $A : \{a_1, a_2, ..., \}$. There exists a set of *task classes* $S : \{S_1, S_2, ...\}$, and for each such task class S_i, there exists a set of possible tasks $\{I_{i,1}, I_{i,2}, ...\}$ of that class which might appear. We will presume that at most one task of a given class will be posted at any time. At a given timestep t there is a set $Q^{(t)}$ of uncompleted tasks available for the bounty hunters to pursue. These tasks will be known by the integers $1, 2, ..., i, ...$. Each such uncompleted task $i \in Q^{(t)}$ has a *bounty* $b_i \in \mathbb{R}$, which unless otherwise specified (as in Experiment 6) rises at a fixed rate $r_i = 1.0$. When an agent commits to a task at time t, it establishes a relationship between the agent, the task it committed to, and the bounty it will receive, which is fixed to the task's current bounty at time t. If the agent has not committed, we denote its current "task" as $\square$, whose bounty is unimportant. This relationship is defined as the mapping $M_t : A \times \{Q^{(t)} \cup \{\square\}\} \to \mathbb{R}$.

A bounty hunter may *commit* to at most one task at a time. Agents could refuse to commit to any task for some period of time, but in our model this is generally irrational to do (and by design, agents do not do this). A bounty hunter may not work on a task until he has committed to it; when he commits, this fact is announced to the other agents. Committing does not imply exclusivity: agents may and can commit to the same task. In some methods discussed later, an agent must complete his committed task, or be beaten to its completion by some other agent, before he can work on a new one. In other methods, he may abandon unfinished tasks and start working on new ones immediately. When a bounty hunter successfully completes a task, he receives a payment from the bondsman.

Exclusivity and the Bounty "Auction" Method In Wicke *et al.* [2015] it was argued that auction methods from the literature can be difficult to adapt to dynamic multiagent allocation problems such as their test problem, but they managed to test against pseudo-auction-like techniques by modifying their bounty model to permit exclusivity. We have selected one such technique for comparison in later experiments here.

In this method, called a "bounty auction" (or simply *auction*), when an agent commits to a task, he has exclusive ownership of it (no other agents may commit to it). Committing uses an abstract auction-like approach as follows: at a given time t, when one or more agents wish to commit, *all* agents are queried for their valuations of *all* the uncommitted tasks in $Q^{(t)}$. The agent with the highest valuation of a task is paired with that task, breaking ties randomly, and both the task and agent are removed from consideration. From the remainder, we once again pair the highest valuation agent-task pair and remove them from consideration, and so on until all agents have been paired to some task. Then any agent who wants to commit to a task is assigned the task to which he was paired. This mechanism by its nature incorporates a kind of single-task queue of sorts: if an agent is working on a task it can still claim a highly valued task in $Q^{(t)}$.

Adaptive Valuation

Previous work defined two approaches to adaptive valuation, denoted *Simple* and *Complex*, which vary in what information they consider when making historical valuations of tasks. Both methods do well in different circumstances. They also used *Bounty Auctions* and a *Greedy* agent for comparisons. Here is a brief summary.

Simple In the *Simple* method, the agent does not consider the other agents in his historical valuation of tasks. Given the bounty b_i for an open task of class i and the agent's historical time to completion T_i of tasks in this class, an agent would value the task as the "bounty per timestep" of the task $\frac{b_i}{T_i}$, times the historical probability P_i of completing tasks in this class. When choosing a task to commit to, the agent would with some ε probability select a random task, else select the task $I^* \leftarrow \text{argmax}_{i \in Q^{(t)}} \frac{b_i}{T_i} P_i$.

The values T_i and P_i start at 1 each, and are updated as follows. When an agent completes a task to which he has committed, they are updated as $T_i \leftarrow (1-\alpha)T_i + \alpha t$ (where t was the time taken to complete the task) and $P_i \leftarrow (1-\beta)P_i + \beta$ respectively. If the agent was instead beaten out by some other agent, then we only update $P_i \leftarrow (1-\beta)P_i$. Finally, after an agent has completed a task or has been beaten out of one, in

[1] The bondsman is fake: several bondsman could post tasks if you liked, or tasks could post themselves and award their own bounties.

order to make the agent a bit more optimistic about task classes in general we update all task classes as $\forall i : P_i \leftarrow (1-\gamma)P_i + \gamma$.

As in the previous paper, we fixed $\alpha = 0.1, \beta = 0.2$. The choice of γ and ε, our exploration parameters, varied depending on the version of *Simple* being tried. We tested ordinary (but good performing) *Simple* ($\varepsilon = 0, \gamma = 0$), and a new version of called *SimplePR* ($\varepsilon = 0.002, \gamma = 0.001$). The reasoning behind *SimplePR* was that it would have exploration properties similar to *SimpleJump* (discussed later), and so could serve as a control better than *Simple* would.

Complex In the *Complex* method, the agent considers which agents beat him out to a task when valuing task performance. To do this, the *Complex* method defines $P_{i,a}$ to be the historical probability of completion of tasks of class i when agent a has also committed to the same task. The naive estimate of P_i is the product, over all agents a presently committed to i, of $P_{i,a}$. Thus the agent now chooses a random task with some ε probability, else it selects the task $I^* \leftarrow \text{argmax}_{i \in Q^{(t)}} \frac{b_i}{T_i} \prod_{a \text{ presently committed to } i} P_{i,a}$.

T_i again starts at 1 and is updated exactly as it was in *Simple*. $P_{i,a}$ starts at 1 and is updated as follows. When the agent completes a task, $\forall a$ presently committed to $i : P_{i,a} \leftarrow (1-\beta)P_{i,a} + \beta$. When the agent is beaten out in its task by a specific agent a^*, we only update $P_{i,a^*} \leftarrow (1-\beta)P_{i,a^*}$. Regardless, whenever an agent completes a task or is beaten out on one, we update all $P_{i,a}$ values to add optimistic exploration, as $\forall i, a : P_{i,a} \leftarrow (1-\gamma)P_{i,a} + \gamma$.

Again, for *Complex*, $\alpha = 0.1, \beta = 0.2$. We selected the best-performing *Complex* method from Wicke *et al.* [2015], called *ComplexP*, where $\varepsilon = 0, \gamma = 0.001$.

Bounty Auctions and Greedy Agents The *Auction* method is exclusive, and so no other agents may be working on a given task. Thus its valuation works like *Simple*, except that since P now serves no purpose, we fix $(\forall i)\ P_i = 1,\ \beta = 0,\ \gamma = 0$.

We also included the *Greedy* method from previous work to serve as a rough upper bound. Here, an agent knows the *true* expected time $E(T_i)$ to complete a task of class i, and just commits to the task $I^* \leftarrow \text{argmax}_{i \in \text{Uncommitted Tasks}} \frac{b_i}{E(T_i)}$. Only one agent will commit to a task.

Jumping Ship

The task-abandoning method we consider is an extension of the *Simple* method, called *SimpleJump*. A fundamental difference in our approach from that in previous work is that, upon task completion an agent receives the bounty posted when he had committed to the task, but in our jump-ship method the agent receives the bounty posted when the task is completed. We will assume that the agent is altruistic in the sense that he will not drag out the task in order to receive more bounty (at any rate, this might not be a good strategy as it would risk another agent beating him out).[2]

We had also considered a jump-ship version of the *Complex* method, but our preliminary work suggested that it was considerably worse than even plain (and poor-performing) *Complex*. We believe this is because the other agents' frequent task abandonment makes it difficult for the agent to build-up sufficient information about them to determine whether to jump ship. As a result, we focus on *SimpleJump* here:

In addition to T_i and P_i, agents also maintain R_i, an estimation of the bounty growth rate for tasks of class i. For most problems, this is simply set the fixed bounty rate, that is, $R_i = r_i\ (= 1.0)$. In the Experiment 6, we vary the bounty rate, and so every single timestep t, R_i is adapted as $R_i \leftarrow (1-\rho)R_i + \rho r_i^{(t)}$, where $r_i^{(t)}$ is the current bounty rate. ρ is a new parameter, set like α to 0.1.

SimpleJump agents may abandon tasks at any time: but if one revisits a task, he must start from scratch. Thus an agent determines at every timestep which task he presently wants to work on. Specifically, each time step t the agent chooses a random open task with probability ε, else he chooses a task as:

$$I^* \leftarrow \underset{i \in Q^{(t)}}{\text{argmax}} \frac{b_i + R_i T_i}{T_i} P_i = \underset{i \in Q^{(t)}}{\text{argmax}} \frac{b_i}{T_i} P_i + R_i P_i$$

That is, instead of using the current bounty b_i as the estimated reward for task completion, the agent is using the expected bounty received on task completion, $b_i + R_i T_i$. If I^* is different from the current task being worked on, the agent will abandon the old task and commit to the new one.

While an agent is working on a task, three things could happen. The agent might complete the task, be beaten out by some other agent, or decide to abandon the task. In the first two cases, we follow the rules of *Simple* to update T and P. When the agent has abandoned a task, we only update P as:

$$P_i \leftarrow (1-\beta)P_i + \beta j$$

where j denotes a small incentive for switching tasks, which helps exploration. We set $j = 0.25$ ($j = 0.0$ performs poorly).

Exploration Strategy Because it relies so heavily on ε to make decisions about task abandonment, *SimpleJump* doesn't use a fixed value of ε, but instead adapts ε to compensate for poor T estimates. We implement the *Value-Difference Based Exploration* (or VDBE) strategy as described in Tokic [2010] to revise ε. Every timestep, the agent has either just updated T_i to a new value T_i', or has not changed it (so we define $T_i' = T_i$). We then update ε as:

$$\varepsilon \leftarrow \frac{1}{|S|} \times \frac{1 - e^{\frac{-|T_i' - T_i|}{\sigma}}}{1 + e^{\frac{-|T_i' - T_i|}{\sigma}}} + \left(1 - \frac{1}{|S|}\right) \times \varepsilon$$

Recall that $|S|$ is the size of S, the set of task classes. We set $\sigma = 0.85$ so as to not be overly sensitive to changes in the time to complete tasks. Initially $\varepsilon = 0.002$ in *SimpleJump*.

Experiments

To test the effectiveness of *SimpleJump*, we chose to use the same simulated environment (retrieving balls) as described in Wicke *et al.* [2015]. We used four agents, placed in each of the four corners of the environment. For the first five experiments the size of the environment was set to 60×40, and an initial bounty of 100 was posted for new tasks. For the last two experiments the environment was set to 600×400: Experiment 6 used an initial bounty of 50,000, and Experiment 7 used 1000.

[2] As reward is just a fiction for purposes of valuation, we are not giving any "unfair advantage" to this method by changing the reward.

Except for Experiment 6, the bounty rate r_i was fixed to 1.0. We defined twenty task classes, each with a *mean location* uniformly randomly chosen somewhere within the environment between the agents' corners. The simulation would start with twenty balls (the tasks, one task per class), which appeared on the field at random locations using a gaussian distribution centered at their task class means and with a standard deviation $\sigma = 5$ in each dimension.

As in the previous bounty hunting paper, the agents could move one grid square every timestep in any of the four cardinal directions, and upon completion of a task, any agent working on the task was teleported back to their home base (teleportation had little impact on the results and doubled the simulation speed). Since we now permitted SimpleJump to abandon tasks, if an agent decided to jump ship, it would be automatically teleported back to its home base to prevent it from taking advantage of its current position.

We tested six methods *Simple, SimplePR, ComplexP, Greedy, Auction*, and *SimpleJump*, in seven scenarios. The first five scenarios are drawn from Wicke *et al.* [2015]: a static environment, dynamic collaborators, dynamic tasks, unreliable collaborators, and unexpectedly bad tasks. The remaining two are new domains meant to highlight scenarios where *SimpleJump* would clearly be preferred. The first such experiment (Experiment 6), varies bounty rate for different classes. The second spawns new tasks in the environment, with larger bounties than have been seen before, after valuation has converged.

The first five experiments ran for 200,000 steps, and last two for 400,000, as they had a larger environment. Each method/scenario combination had 200 independent trials. The metric used was the total bounty $h(t)$ at time t. We verified statistical significance with an ANOVA and Tukey at $p = 0.05$.

First Experiment: A Static Environment

In the first experiment, we implemented the static scenario, where a ball of a given task class would appear (at new random locations) some $p \sim \text{uniform}(0, 19)$ timesteps after a previous one had been retrieved by an agent. If two agents completed the same task simultaneously, the winner would be the one with the lowest agent ID (1, 2, 3, or 4). This was used as a baseline test to confirm that *SimpleJump* was able to adapt in the simplest case. The results are shown in Table 1.

Results The results showed that the *SimpleJump* method in the static case had a lower mean than all other methods (see Table 1). Also, the *SimpleJump* method outperformed *Greedy* as seen in Table 1. The main feature of the *Greedy* method was its knowledge of the exact location of the tasks, so it did not have to learn this. However, it did not know the order of tasks or when they would appear, and like the *Auction* method, the *Greedy* agents were committed to their task and the tasks were exclusive. This experiment reaffirmed previous findings and illustrated the performance of *SimpleJump*.

Second Experiment: Dynamic Agents

The second experiment tested a method's ability to adapt scenarios where agents were prone to failure. This time Agent 1 was removed from the game every 30,000 timesteps, and Agent 2 was removed every 60,000 timesteps. An agent was reinserted 20,000 timesteps after being removed.

Equivalence Classes				Method	γ	ε	Mean
			+	ComplexP	0.001	0	3525.45
		+	+	SimplePR	0.001	0.002	3513.05
	+	+		Simple	0	0	3449.96
	+			Greedy	-	-	3417.09
	+			Auction	-	-	3400.97
+				SimpleJump	0.001	0.002	3315.54

Table 1: Experiment 1 results, Static Environment, at time = 200,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

Equivalence Classes		Method	γ	ε	Mean
	+	SimplePR	0.001	0.002	6531.08
+	+	Simple	0	0	6484.09
+	+	ComplexP	0.001	0	6471.57
+	+	SimpleJump	0.001	0.002	6394.3
+	+	Auction	-	-	6383.78
+		Greedy	-	-	6354.95

Table 2: Experiment 2 results, Dynamic Agents, at time = 200,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

Results Table 2 is a bit muddied: most methods fell in the same statistical significance class, including *SimpleJump*. The only significant difference was between *SimplePR* and *Greedy*. However, this result might be a consequence of the particular cutoff at 2,000 timesteps; *SimpleJump* would generally converge to a better value when all agents were again present.

Third Experiment: Dynamic Tasks

Here the distribution of the tasks could change suddenly by rotating the home bases of the agents. A single rotation (agent 1's corner became agent 2's corner, and so on) would occur every 25,000 timesteps, and a double-rotation would occur every 50,000 timesteps: this invalidated the previous knowledge learned by the agents, requiring rapid adaptation.

Results As shown in Table 3, this was the hardest domain for *SimpleJump*; it placed in the last equivalence class, along with *Simple*. We also noticed that although *SimpleJump* was in the last tier of all the methods, it didn't spike as high as *Auction* or *ComplexP* (as can be seen in Figure 1), which would suggest that abandoning tasks might help deal with instant changes.

Fourth Experiment: Unreliable Collaborators

This is the first of two experiments originally meant to show the advantages of non-exclusivity. Here, two additional unreliable collaborators were added to the system (sharing home bases with agents in the top two corners). These unreliable collaborators committed to tasks at random and moved 10x slower than other agents. Obviously, if these agents had exclusive ownership they would slow things down significantly.

Results The results in Table 4 show that *SimpleJump* again performed best, and confirm the previous result in Wicke *et al.* [2015]: namely, that *ComplexP* also significantly outperforms *Auction*. We also note that in this setup *Simple* was in the same equivalency class as *ComplexP*. However in previous

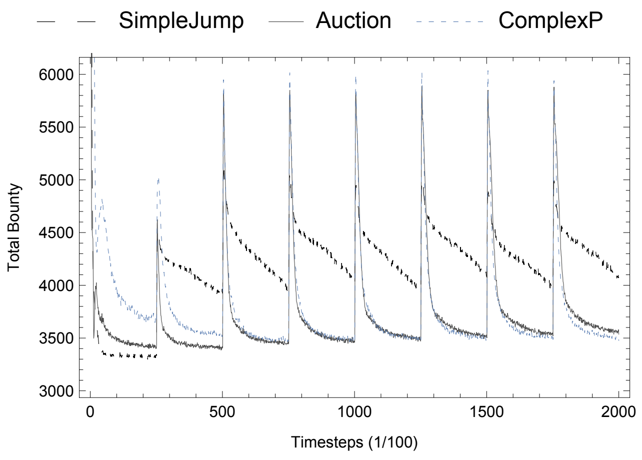

Figure 1: Experiment 3, Dynamic Tasks (Selected Results), 200,000 steps. Lower values are better. Legends are read from left to right corresponding to the plots top to bottom.

Equivalence Classes				Method	γ	ε	Mean
			+	Simple	0	0	4161.35
			+	SimpleJump	0.001	0.002	4072.62
		+		SimplePR	0.001	0.002	3573.63
	+	+		Auction	-	-	3535.79
+	+			ComplexP	0.001	0	3476.64
+				Greedy	-	-	3402.37

Table 3: Experiment 3 results, Dynamic Tasks, at time = 200,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

work *Simple* was not included in the experiment. We also found that *Auction* performed worst. This is important as it shows that the ability to abandon tasks does not hinder the ability to handle unreliable collaborators, but in fact helps.

One interesting result is how fast *SimpleJump* converges (similar to Figure 1). It seems likely that the exploration method used is the reason, but future analysis is still needed.

Fifth Experiment: Unexpectedly Bad Tasks

This experiment, which was also originally designed to demonstrate the usefulness of non-exclusivity, produced tasks which with 10% probability were unexpectedly "bad" for a randomly selected agent. By "bad" we meant that the agent would be 10x slower to perform that task than normal.

Results Again, *SimpleJump* performed the best, as seen in Table 5. The *ComplexP* method and *SimplePR* methods performed equivalently, and *Greedy* and *Auction* performed worst.

Sixth Experiment: Variable Bounty Rate

The final two experiments were designed to isolate and make evident the need for task abandonment, and so show the effectiveness of *SimpleJump*. Thus, in these experiments we increased the environment size to 600 $\times$ 400 to make tasks longer and thus give *SimpleJump* more opportunity to give up.

Experiment 6 was motivated by the fact that the bounty rate can increase by more than 1.0. This might happen if

Equivalence Classes				Method	γ	ε	Mean
			+	Auction	-	-	3779.99
			+	Greedy	-	-	3742.71
		+		SimplePR	0.001	0.002	3532.85
	+	+		ComplexP	0.001	0	3472.55
	+			Simple	0	0	3438.33
+				SimpleJump	0.001	0.002	3281.78

Table 4: Experiment 4 results, Unreliable Collaborators, at time = 200,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

Equivalence Classes				Method	γ	ε	Mean
			+	Greedy	-	-	7293.69
			+	Auction	-	-	7014.43
		+		Simple	0	0	6300.30
	+			SimplePR	0.001	0.002	5264.39
	+			ComplexP	0.001	0	5010.31
+				SimpleJump	0.001	0.002	4609.13

Table 5: Experiment 5 results, Unexpectedly Bad Tasks, at time = 200,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

the particular task suddenly becomes more urgent than others. For this experiment we set the default bounty to 50,000, and each task class's bounty increment rate was chosen uniformly randomly over the interval $[1, 50]$, keeping the conservative rule established in Wicke *et al.* [2015], which relates bounty rates with initial bounty values.

Results The results for this experiment were intriguing and are listed in Table 6. Note that the means are higher due to the higher initial bounty. The *Auction* method is in the same equivalency class as *Simple*, but is significantly better than *ComplexP*. This appears to be a scenario where the *Auction* method does relatively well. Even so, *SimpleJump* still outperforms it and all other approaches. We think that part of the reason that *SimpleJump* is good in this domain is that it can rapidly identify fast-growing bounties and shift to their tasks.

We note that in Figure 2 *SimpleJump* does not have as large of an initial spike in the total bounty as compared to either the *ComplexP* method or the *Auction* method. Furthermore, *SimpleJump* in Figure 2 lacks a second, smaller spike which is present in the other methods. We hypothesize that incorporating the learned bounty rate into the decision function *SimpleJump* avoids the second spike. The longer runlength (400,000 timesteps), reveals *Auction*'s gradual decline.

Seventh Experiment: Emergent Tasks

In our final experiment, in addition to the standard 20 task classes, we added four additional *emergent task* classes. A task from an emergent class appears only after 20,000 timesteps, then iteratively reappears 20,000 timesteps after it has been completed. Unlike the regular task classes, the emergent task classes have an initial bounty of 2,000. These type of emergent tasks are meant to simulate a scenario where some very important but infrequent tasks must be done. This would benefit from agents who abandon their existing tasks to handle emergent ones as soon as possible.

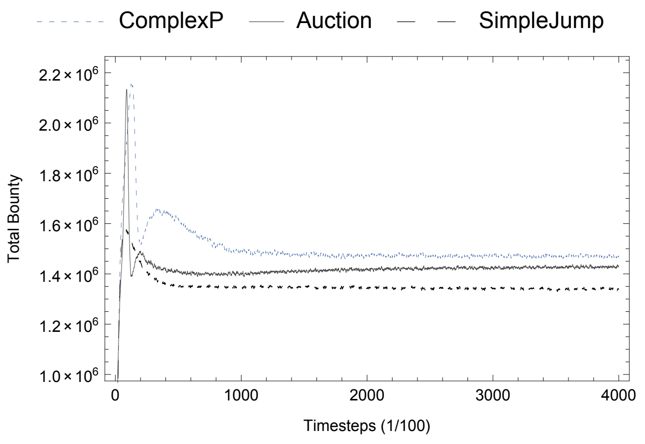

Figure 2: Experiment 6, Variable Bounty Rate (Selected Results), 400,000 steps. Lower values are better. Legends are read from left to right corresponding to the plots top to bottom.

Equivalence Classes				Method	γ	ε	Mean
			+	SimplePR	0.001	0.002	1.49×10^6
		+	+	ComplexP	0.001	0	1.47×10^6
	+	+		Simple	0	0	1.44×10^6
	+	+		Greedy	-	-	1.43×10^6
	+			Auction	-	-	1.42×10^6
+				SimpleJump	0.001	0.002	1.34×10^6

Table 6: Experiment 6 results, Variable Bounty Rate, at time = 400,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

Results This experiment again demonstrates the value of *SimpleJump*. As shown in Table 7 *SimpleJump* does significantly better than all other methods, and again *Auction* does better than *ComplexP* and *SimplePR*.

Figure 3 illustrates that *SimpleJump* can easily handle *emergent tasks* than the other methods, as the spikes in the graph are not as pronounced. Also, notice that although the methods spike when the emergent tasks are present, as time increases the spikes decrease in height. This indicates that the other methods are taking a much longer time to learn how to handle the emergent tasks, since they show less frequently compared to other tasks. However, *SimpleJump* can make the right decision solely based on the bounty value of these tasks, which suggests why the spike height of the bounty method is shorter.

Conclusions and Future Work

We have shown that bounty hunting can be improved by eliminating task commitment. We compared this approach, called *SimpleJump*, against an auction-like mechanism and various commitment-based bounty methods, and demonstrated that it performed statistically significantly best in five of seven experiments, and fell in the top significance tier in one experiment. Only in one scenario did it perform suboptimally. The five strong-performing experiments included not only methods meant to highlight the advantages of *SimpleJump*, but also several for which we did not expect it to fare well.

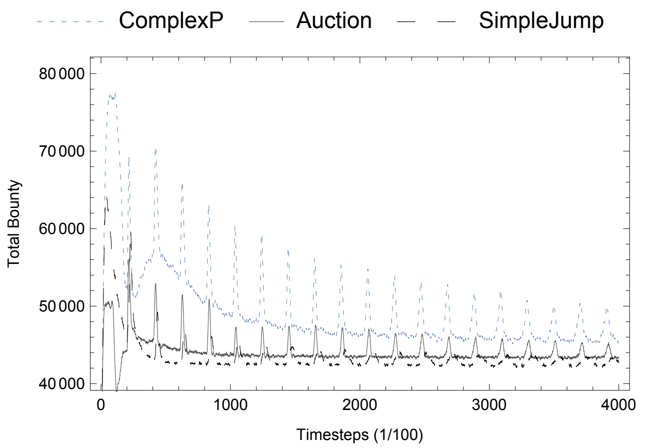

Figure 3: Experiment 7, Emergent Tasks (Selected Results), 400,000 steps. Lower values are better. Legends are read from left to right corresponding to the plots top to bottom.

Equivalence Classes			Method	γ	ε	Mean
		+	ComplexP	0.001	0	46274.6
		+	SimplePR	0.001	0.002	45777.7
	+		Simple	0	0	43878.2
	+		Greedy	-	-	43675.7
	+		Auction	-	-	43515.9
+			SimpleJump	0.001	0.002	42442.3

Table 7: Experiment 7 results, Emergent Tasks, at time = 400,000. Lower means are better. Equivalence Classes show statistically insignificant differences between methods.

We think that *SimpleJump* does not do well in Experiment 3 because its exploration strategy has failed. In Figure 3 the *SimpleJump* method does not spike as high, but the method also does not converge back to the level of *ComplexP* or *Auction*. This seems to be due to slow exploration, which also occurs with *Simple*, as was illustrated in Figure 3 of Experiment 3 in Wicke *et al.* [2015]. Still, we conclude that jumping ship often outperforms exclusive (pseudo-auction) methods. Never did an exclusive method outperform all other bounty techniques.

Our present methods assume no collaboration among the agents. Since bounty hunting allows multiple agents to work on the same task, it seems natural to examine coalitions as a straightforward extension for future work. Along the same lines we are also interested in *bail bondsman hierarchies*, whereby a bail bondsman may act as a broker for agents (or other bondsman) in his sub-group, filtering useful tasks for them to work on. We may also look into more promising exploration strategies. *SimpleJump* relies heavily on its various exploration strategies to perform well, and updates them very rapidly. We think that it is critical to understand how exploration effects the convergence of the system and how effective it can be when system dynamics have suddenly changed.

Acknowledgments

The work presented in this paper is supported by NSF NRI Grant 1317813.

References

[Botelho and Alami, 1999] S. C. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1234–1239, 1999.

[Dias, 2004] M. Bernardine Dias. *Traderbots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2004.

[Gerkey and Matarić, 2002] B. P. Gerkey and M. J. Matarić. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

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

[Jones *et al.*, 2006] E. Gil Jones, B. Browning, M. Bernardine Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *IEEE International Conference on Robotics and Automation*, pages 570–575, 2006.

[Khamis *et al.*, 2015] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks 2015*, pages 31–51. Springer, 2015.

[Korsah *et al.*, 2013] G. Ayorkor Korsah, A. Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[Lagoudakis *et al.*, 2004] M. G. Lagoudakis, M. Berhault, S. Koenigt, P. Keskinocak, and A. J. Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 698–705, 2004.

[Lerman *et al.*, 2006] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J Matarić. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, 25(3):225–241, 2006.

[Liu and Shell, 2012] Lantao Liu and Dylan Shell. A distributable and computation-flexible assignment algorithm: From local task swapping to global optimality. In *Proceedings of Robotics: Science and Systems*, 2012.

[Liu *et al.*, 2014] Lantao Liu, Nathan Michael, and Dylan Shell. Fully decentralized task swaps with optimized local searching. In *Proceedings of Robotics: Science and Systems*, 2014.

[Nanjanath and Gini, 2006] M. Nanjanath and M. Gini. Dynamic task allocation for robots via auctions. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2781–2786, 2006.

[Nanjanath and Gini, 2010] Maitreyi Nanjanath and Maria Gini. Repeated auctions for robust task execution by a robot team. *Robotics and Autonomous Systems*, 58(7):900–909, 2010.

[Pippin and Christensen, 2012] Charles E. Pippin and Henrik Christensen. Learning task performance in market-based task allocation. In *Proceedings of the 12th International Conference on Intelligent Autonomous Systems*, volume 2, pages 613–621, 2012.

[Pustowka and Caicedo, 2012] A Pustowka and E.F. Caicedo. Market-based task allocation in a multi-robot surveillance system. In *Robotics Symposium and Latin American Robotics Symposium*, pages 185–189, 2012.

[Shiroma and Campos, 2009] P. M. Shiroma and M. F M Campos. Comutar: A framework for multi-robot coordination and task allocation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4817–4824, 2009.

[Simmons *et al.*, 2000] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2000.

[Tokic, 2010] Michel Tokic. Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences. In *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence*, pages 203–210, 2010.

[Walsh and Wellman, 1998] William E Walsh and Michael E Wellman. A market protocol for decentralized task allocation. In *International Conference on Multi Agent Systems*, pages 325–332. IEEE, 1998.

[Wicke *et al.*, 2015] Drew Wicke, David Freelan, and Sean Luke. Bounty hunters and multiagent task allocation. In *International Conference on Autonomous Agents and Multiagent Systems*, 2015.

[Zlot *et al.*, 2002] Robert Zlot, A. Stentz, M. Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. In *IEEE International Conference on Robotics and Automation*, 2002.