Learning the Task Allocation Game [*]

Sherief Abdallah[†] and Victor Lesser
University of Massachusetts
Amherst, MA
{shario,lesser}@cs.umass.edu

ABSTRACT

The distributed task allocation problem occurs in domains like web services, the grid, and other distributed systems. In this problem, the system consists of servers and mediators. Servers execute tasks and may differ in their capabilities, e.g. one server may take more time than the other in executing the same task. Mediators act on behalf of users, which can potentially be other mediators, and are responsible for receiving tasks from users and allocating them to servers.

This work introduces a new gradient ascent learning algorithm that outperforms state of the art multiagent learners on this problem. We experimentally show that our algorithm converges faster and is less sensitive to tuning parameters than other algorithms. We also provide an informal proof that WPL has the same convergence guarantee as the best known algorithm, GIGA-WoLF. We also show that our algorithm converges in Jordan's and Shapley's games where many other algorithms fail to converge. Finally, we verify the practicality of our algorithm in the distributed task allocation domain, comparing its performance to an optimal global solution.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Experimentation

[*]This material is based upon work supported by the National Science Foundation Engineering Research Centers Program under NSF Award No. EEC-0313747. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[†]A student author.

Keywords

Reinforcement Learning, Multiagent Systems, Task Allocation, Markov Decision Process

1. INTRODUCTION

The distributed task allocation problem occurs in domains like web services, the grid, and other distributed systems. The problem can be illustrated by Figure 1. There are L servers and N mediators. Servers execute tasks and may differ in their capabilities, e.g. one server may take more time than another server in executing the same task. Mediators receive tasks from users, which can potentially be other mediators. Mediators are responsible for allocating tasks to servers. The goal is to find an optimal allocation that minimizes the average turn around time.[1] Turn Around Time, or TAT, is the time interval between a task arrival and its completion. It includes both the task processing time and the task waiting time at the server. Average TAT is then averaged over tasks to produce TAT.

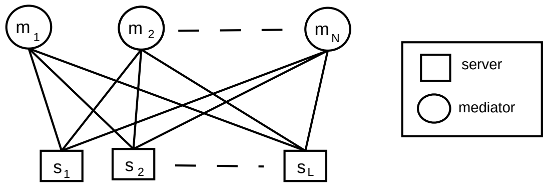


Figure 1: An example of distributed task allocation.

What makes this problem challenging is the *limited observability* of each mediator. In a large open system, a mediator will not be aware of all other mediators, their queues, their actions, or the queue length of each server. Ideally, one would like a learning algorithm that finds the optimal policy for each mediator. We identify certain characteristics that are desirable in such a learning algorithm.

- **Ability to learn a stochastic policy.** This is necessary in order to minimize TAT. For example, consider a system of two servers and three mediators. Any deterministic joint policy will overload one server more than the other, increasing the average TAT.
- **Convergence.** If the mediator policies do not converge, then oscillation between policies will occur making the system unstable.

[1]Other optimal criteria are also possible.

- **Adaptability.** This is directly related to the convergence property: not only do we need the algorithm to converge, we want it to converge quickly. This is important in a dynamic and open system that is constantly changing.

This work develops a gradient ascent learning algorithm that meets all these requirements and is better than the state-of-the-art algorithms. We analyze its performance in two-action games and prove its convergence. We also show that our algorithm converges in Jordan's and Shapley's game where many other algorithms fail to converge. Furthermore, we show that our algorithm converges faster and is less sensitive to tuning parameters when compared to previous approaches. Finally, we verify the practicality of our algorithm in the distributed task allocation domain, comparing its performance to an optimal global solution (computed using queuing theory).

This paper is organized as follows. Section 2 gives necessary definitions of game theory, which provides an elegant methodology for analyzing the performance of multiagent learners. Section 3 describes our algorithm and proves its convergence. Section 4 describes in more detail the state of the art algorithms. Section 5 verifies our claims in both simple games and distributed task allocation. Finally, we conclude in Section 6.

2. GAME THEORY

Game theory provides an elegant framework for analyzing learning in multiagent systems. It was used by previous researchers [4, 3, 2] to analyze the convergence properties of multiagent learning algorithms. This section gives an overview of the game theory framework, illustrating how it can be used to model distributed task allocation. The following section describes our algorithm and analyzes its convergence using game theory.

A game in normal form is defined by the tuple $\langle n, A_1, ..., A_n, R_1, ..., R_n\rangle$, where n is the number of players[2] in the game, A_i is the set of actions available to agent i, and $R_i : A_1 \times ... \times A_n \rightarrow \Re$ is the reward agent i will receive as a function of the joint action executed by all agents. If the game has only two players, then it is convenient to define their reward function as a payoff matrix as shown in Table 1. Each cell (i, j) in the grid represents the payoff received by the row player (Player 1) and the column player (Player 2), respectively, if the row player plays action i and the column player plays action j. The first three games in Table 1 were used in evaluating previous research [3, 1, 2].

The game in Table 1(d) represents the task allocation game at time t for two mediators and two servers. Action s_i corresponds to the row/column mediator assigning the task to server s_i. l_i^t is the expected TAT if the task is allocated to server i at time t. This is a compact representation of a set of games with constant payoff. The game the mediators play at time t depends on the status of the server queues at that time. However, due to the limited observability of each mediator, mediators can not distinguish between these games. In other words, an optimal policy would be the policy that performs best, on average, over time. While we will limit our theoretical analysis to games with payoffs that remain constant over time, Section 5 shows the effectiveness of our algorithm experimentally in the task allocation game with varying payoffs and involving more players and servers.

[2]We use the terms: agent, player, and mediator interchangeably.

	a1	a2
a1	2,1	0,0
a2	0,0	1,2

(a) coordination

	H	T
H	1,-1	-1,1
T	-1,1	1,-1

(b) matching pennies

	a1	a2
a1	0,3	3,2
a2	1,0	2,1

(c) tricky

	s1	s2
s1	$-(l_1^t+2), -(l_1^t+2)$	$-(l_1^t+1), -(l_2^t+1)$
s2	$-(l_2^t+1), -(l_1^t+1)$	$-(l_2^t+2), -(l_2^t+2)$

(d) task allocation

Table 1: 2-action games

A policy (or a strategy) of agent i at time t is $\pi_i^t \in PD(A_i)$, where PD is the set of probability distributions over actions A_i. In case of two-player-two-action games (like the ones in Table 1) we refer to the joint policy of the two players by (p, q), where $\pi_1 = (p, 1-p)$ and $\pi_2 = (q, 1-q)$. Let the vector r_i^t be the actual reward agent i would receive at time t for each of its actions. This is the only feedback signal agent i would receive. The expected payoff agent i would get given its policy is $V_i^t(\pi_i) = \pi_i^t.r_i^t$.

2.1 Learning and Convergence

The primary goal of any rational learner would be to maximize its expected payoff by changing its policy. The problem is that r_i^t is not fixed and depends on other agents' policies. Consider for example the matching pennies game in Table 1(b). Each player can play either head (H) or tail (T). The payoff of Player 1 (row player) is $r_1^t = (2q-1, 1-2q)$, which is dependent on Player 2's policy q. Therefore, an optimal policy for Player 1 depends on the current policy of Player 2 and vice versa.

This dependence between optimal policies may prevent the convergence of a simple policy gradient algorithm. Consider the matching penny example again. Assume that initially the joint policy of the two players is $(p^0 = 1, q^0 = 0)$ (i.e. Player 1 always plays H and Player 2 always plays T), this means $r_1^t = (-1, 1)$ and $r_2^t = (-1, 1)$. Player 1 realizes it can do better by moving its policy toward action T by a small step of size δ. The new joint policy is $(p^1 = 1-\delta, q^1 = 0)$. The process continues, until time t where the joint policy is $(p^t = 0.5-\delta, q^t = 0)$. In this case $r_2^t = (\delta, -\delta)$. Player 1 continues moving its policy toward $p = 0$ while player 2 starts increasing q. This process continues and the two players keeps oscillating around the joint policy $(p = 0.5, q = 0.5)$ as shown in the Figure 2.

The joint policy $(p^* = 0.5, q^* = 0.5)$ that both players are oscillating around is the *Nash Equilibrium* of the matching pennies game. A joint policy is a Nash Equilibrium, NE, iff no agent can get higher expected reward by changing its policy unilaterally. A pure NE is the one where all agents policies are deterministic. Otherwise the NE is mixed. The coordination game has two pure NE, $(1, 1)$ and $(0, 0)$. The other two games have one mixed NE, $(0.5, 0.5)$.

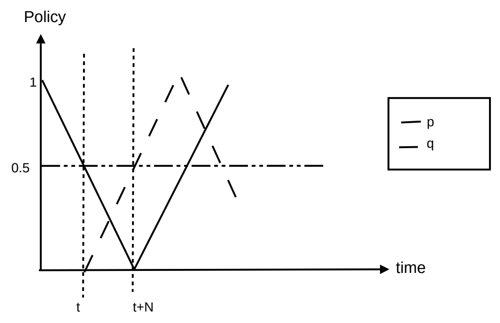


Figure 2: An illustration of policy oscillation.

The oscillation we have just described is common in policy gradient algorithms that change policies depending only on actual reward r_i^t [7, 10]. This is particularly true when the NE is mixed. The main idea behind our algorithm is to slow down learning such that this oscillation dampens until the joint policy gradually stabilizes to a NE.

3. THE WEIGHTED POLICY LEARNER ALGORITHM

Algorithm 1 illustrates our WPL algorithm. Δ is the policy gradient which is used to update the policy π. The idea of our algorithm is to start learning fastest when Δ changes its direction and then to gradually slow down learning if the policy gradient does not change its direction. We detect change in the direction of the gradient using the difference between action rewards (because the policy gradient follows the reward). If the reward of action a is decreasing, then the change in $\pi(a)$ is weighted by $\pi(a)$, otherwise it is weighted by $(1-\pi(a))$. Therefore, the change in $\pi(a)$, $\Delta(a)$, is largest when either $\pi(a)$ is near 1 and $r(a)$ is lower than average reward $\bar{r}$, or $\pi(a)$ is low and $r(a)$ is higher than $\bar{r}$. The process repeats until convergence is reached and we get a damped oscillation as shown in Figure 3. The *limit* function is adopted from [10] with minor modifications to ensure fixed exploration: $limit(x) = argmin_{x':valid(x')}(x - x')$, i.e. $limit(x)$ returns a valid policy that is closest to x. A policy is valid if it sums to 1 and every action is played with miminum probability ϵ (for exploration). The following section discusses the convergence of WPL.

Algorithm 1: WPL: Weighted Policy Learner

```
begin
    Let $r(a)$ be the (expected) reward for executing action $a$
    $\widehat{r} \leftarrow$ total average reward $= \frac{\sum_{a \in A} r(a)}{|A|}$.
    foreach action $a \in A$ do
        $\Delta(a) \leftarrow r(a) - \widehat{r}$
        if $\Delta(a) > 0$ then $\Delta(a) \leftarrow \Delta(a)(1 - \pi(a))$
        else $\Delta(a) \leftarrow \Delta(a)(\pi(a))$
    end
    $\pi \leftarrow limit(\pi + \Delta)$
end
```

3.1 Convergence Proof

This section presents an informal proof that WPL converges for 2x2 games that is verified by experimental results involving 2x2 games as well as other games.[3] In the case of a pure NE, any gradient ascent player that follows the actual reward r^t will converge to the pure NE [7, 10]. The interesting case is when the NE is mixed. This is usually the case in the distributed task allocation problems, where mediators compete for servers.

Because $\Delta_1^t(a1) + \Delta_1^t(a2)$ will always sum to zero in 2x2 games, the only effect of the function *limit* is to constrain each action probability within 0 and 1. This actually helps convergence because it prevents players from moving further away from the NE (since by definition any NE is within the space of valid policies) [7]. Therefore, the limit function will be ignored for the rest of the proof.

LEMMA 1. *WPL learners stop learning iff they reached an NE.*

PROOF. From the WPL description, the change in the row agent policy, or Δ_1^t is zero only if one of the following is true: $p^t = 0, p^t = 1$, or $r_1^t(a) - \widehat{r}^t$. In practice, p is bounded by $[\epsilon, 1-\epsilon]$, where $\epsilon > 0$. This forces agents to keep exploring non-optimal actions in case the environment changes and non-optimal actions becomes optimal.[4] Therefore, Δ^t is zero iff $r_1^t(a1) = r_1^t(a2)$. The same argument goes for player 2, i.e. it stops learning iff $r_2^t(a1) = r_2^t(a2)$. This is only true in a Nash equilibrium.[5] $\square$

LEMMA 2. *A player changes its gradient direction iff the other player crosses its NE strategy.*

PROOF. We say Player 1 *crosses the NE* at time t iff $p^{t-1} < p^* < p^{t+1}$, where p^* is Player 1's NE policy. For example, in Figure 2 Player 1 crosses the NE at time t. The lemma follows from the fact that when a player is at a NE, the values of both actions of the other player are equal. This means that the relative values of each of player 1's actions switch when player 2 crosses its NE strategy. This changes the gradient direction of player 1 because the gradients follow the action with highest expected reward. The lemma is depicted in Figure 3. $\square$

LEMMA 3. *The two agents using WPL will continue to oscillate around the NE unless both converge to the NE.*

PROOF. Both players try to maximize their reward by moving their strategy toward the action that yields more rewards. At any time t, one agent (player 1) will be better off by moving its strategy closer to the NE strategy while the other agent (player 2) is better off by moving its strategy away from the NE strategy (because we are considering only competitive game, i.e. mixed NE). Eventually, Player 1 will cross its NE strategy. This will cause player 2 to switch its policy gradient direction toward the NE strategy, while player 1 is now moving away from the NE. This process continues, unless both players converge to an NE. $\square$

[3]We are currently working on a more formal proof using differential equations.

[4]Even in a static environment agents need time to correctly estimate the rewards r_i^t either because of noise or other agents learning; ϵ then decreases gradually with time t such that $t \overset{lim}{\rightarrow} \infty \, (\epsilon^t) \rightarrow 0$.

[5]This is a property of two-action games (and can be generalized for larger games): if players are at a mixed NE then both actions for each player have the same expected reward.

LEMMA 4. *A WPL learner slows down (i.e. the magnitude of the policy gradient decreases) if its gradient direction does not change.*

PROOF. We will prove the lemma for Player 1 and a similar argument goes for the column player. Let $|\Delta_1^t| = |\Delta_1^t(a1) + \Delta_1^t(a2)| = 2|\Delta_1^t(a1)| = 2(1-p^t)(r_1^t(a1) - r_1^t(a2))$ be the magnitude of gradient change, assuming $r_1^t(a1) > r_1^t(a2)$. A WPL learner slows down its learning iff $|\Delta_1^{t+1}| < |\Delta_1^t|$. If the gradient direction is the same, then $r_1^{t+1}(a1) > r_1^{t+1}(a2)$, and therefore $|\Delta_1^{t+1}| = 2(1-p^{t+1})(r_1^{t+1}(a1) - r_1^{t+1}(a2))$. There are two cases:

- Either $r_1^{t+1}(a1) - r_1^{t+1}(a2) <= r_1^t(a1) - r_1^t(a2)$. In this case, the gradient magnitude decreases; i.e. $|\Delta_1^{t+1}| < |\Delta_1^t|$ because $1-p^{t+1} = 1-p^t - \Delta_1^t(a1) < 1-p^t$. If the gradient magnitude decreases then the learning slows down.

- Or $r_1^{t+1}(a1) - r_1^{t+1}(a2) > r_1^t(a1) - r_1^t(a2)$. We prove by contradiction that this case can not occur. If $r_1^{t+1}(a1) - r_1^{t+1}(a2) > r^t(a1)_1 - r_1^t(a2)$ then the only explanation is that Player 2 moved faster. However as we have just proved, Player 2 will slow down if its gradient Δ_2^{t+1} is moving in the same direction as Δ_2^t. Therefore, Player 2 must have changed its gradient direction. However, if Δ_2^t increases $r^t(a1) - r^t(a2)$ then Δ_2^{t+1} must decrease this difference (because it is in the opposite direction), and therefore $|\Delta_1^t|$ actually decreases further.

□

LEMMA 5. *A WPL learner moves toward its NE strategy faster than it moves away from its NE strategy.*

PROOF. From the previous lemma, an agent slows down as long as it does not change its gradient direction. Therefore, after player 1 crosses its NE strategy (i.e. starts moving its policy away from the NE) player 1 does not change its gradient direction (while player 2 changes its gradient direction) and consequently player 1 moves away from its NE strategy slower than player 1 moves toward its NE strategy. □

Lemma 3 says that both players will continue to oscillate around the NE while Lemma 5 says that each player will slow down as long as it does not change its policy gradient direction. This leads to a damping effect as illustrated by Figure 3 until both players reach the NE. Our experimental results reflect this same behavior as Section 5 will show.

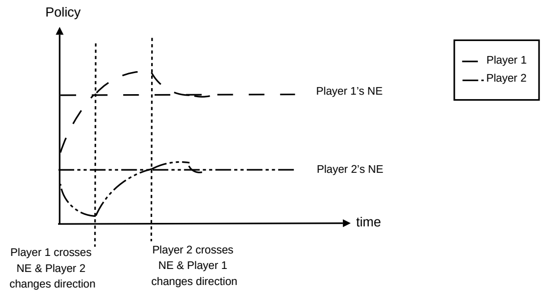

Policy
Player 1
Player 2
Player 1's NE
Player 2's NE
time
Player 1 crosses NE & Player 2 changes direction
Player 2 crosses NE & Player 1 changes direction

Figure 3: An illustration of WPL convergence.

4. RELATED WORK

There are many algorithms that have been proposed for multiagent learning. Unfortunately, all of them either fail to achieve all the three desired characteristics that are listed in Section 1 or assume agents can observe each other's actions. Independent Learners and Joint Learners [4] only learn a deterministic policy using the Q-Learning algorithm[8] and do not have any convergence guarantee. Joint Learners also require the knowledge of other agents' actions, an assumption that is not practical in large open systems.

Equilibrium learners such as Nash-Q [6], OAL [9], AWESOME [5] require that each agent observe other agents' actions. Each agent then computes a stable joint policy (NE). While these learners are guaranteed to converge, similar to joint learners, they are not applicable in open domains.

The gradient ascent learners are the most promising and relevant as they can learn a stochastic policy without observing other agents actions. These algorithms directly follow the expected reward. The Infinitisemal Gradient Ascent algorithm, or IGA [7] and its generalization, Generalized IGA, or GIGA [10] were the first algorithms to provide any convergence guarantees in a multiagent context. They converge to games with pure NE. However both algorithms failed to converge in games with mixed NE, and therefore are not suitable for the distributed task allocation problem.

Several modifications to IGA and GIGA were proposed to avoid divergence in games with mixed NE, including: PHC-WoLF [3], PHC-PDWoLF [1], and GIGA-WoLF [2]. They all used some form of the Win or Learn Fast heuristics [3], whose purpose is to speedup learning if the agent is moving toward its NE policy and slow down learning if the agent is moving away from the NE policy. The main problem with this heuristic is that an agent does not know what policy is an NE policy. Therefore, all the algorithms that use the WoLF heuristic use approximate methods to determine when an agent is moving toward or away from the NE. This is different from our approach, in which an agent slows down its learning as long as its policy gradient has the same direction. Except for GIGA-WoLF, none of the WoLF-based algorithms have any form of convergence guarantee. Therefore, while we will compare our algorithm to all of them, our experiments will focus on comparing our algorithm to GIGA-WoLF.

5. RESULTS

5.1 Practical Issues

5.1.1 *Expected Reward vs Most Recent Reward*

Our algorithm, along with all previous approaches, needs the expected reward of an action a at time t, or $r^t(a)$. However, this is not known a *priori* and needs to be computed. The common approach in reinforcement learning, which we also use, is the equation $r^{t+1}(a) \leftarrow \alpha R^t + (1-\alpha) r^t(a)$, where R^t is the sample reward received at time t and $0 \leq \alpha \leq 1$ is a learning parameter [8]. Tuning α is not an easy task when the expected reward itself is dynamically changing (because other agents are changing their policies concurrently). Smaller α means more samples contribute to computed expected reward and vice versa.

Using more samples to estimate the reward makes the estimate more accurate. However, the time required to get more samples may in fact degrade the accuracy of reward estimate. This is because it will take more time for r^t to follow the dynamically changing expected reward. In other words, a higher α value increases the adaptability of the learning algorithm, but may affect convergence which requires r to reflect the expected reward, not a sample. We investigate this tradeoff in our experiments and show that our approach is less sensitive to the value of α than previous approaches.

5.1.2 *Fixing Learning Parameters*

In our experiments we have not used any decaying rates. The reason is that in an open system where dynamics can continuously change, one would want learning to be continuous as well. We have fixed the exploration rate ϵ to 0.1, the learning rate δ to 0.001. Two values of the learning parameter α have been used to measure the tradeoff between accuracy and responsiveness: 0.1 and 1.

5.2 Benchmark Games

To verify our theoretical result, we tested our algorithm for the three games described in 1. Figure 4 plots $\pi(r1)$ and $\pi(c1)$ against time, when the initial joint policy is $([0.1, 0.9]_r$, $[0.9, 0.1]_c)$ (several initial joint policies have been tested with similar results) and $\alpha = 0.1$. The algorithm converged to an NE in all the games. Note also that the plotted change of agents' policy over time in the two games with mixed equilibrium is almost identical to our theoretical prediction.

The tricky game deserves more attention as it is the most challenging among two-player-two-action games. Figure 5 shows the performance of PHC-WoLF, PHC-PDWoLF, and GIGA-WoLF in this game using the exact same settings of our approach. Only GIGA-WoLF converges but with slower rate than our approach (Figure 4(c)).[6] The performance of PHC-WoLF and GIGA-WoLF conforms to the results reported by their authors [3, 2] (actually PHC-WoLF performs here better than reported results). On the other hand, PHC-PDWoLF diverged in our experiments while it converged in the authors' experiments [1]. A possible explanation is that they used more tuned learning parameters, possibly with decaying rates. Due to the poor performance of both PHC-WoLF and PHC-PDWoLF in the rest of our experiments, the focus will be on GIGA-WoLF thereafter.

Table 2 shows two well known 3-action games used to evaluate GIGA-WoLF [2]. The NE in both games is mixed and equals $([\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]_r, [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]_c)$. GIGA-WoLF is known to converge in the rock-paper-scissors game and diverge in Shapley's game as we verify ourselves.

	c1	c2	c3
r1	0,0	-1,1	1, -1
r2	1,-1	0,0	-1,1
r3	-1,1	1,-1	0,0

(a) rock paper scissors

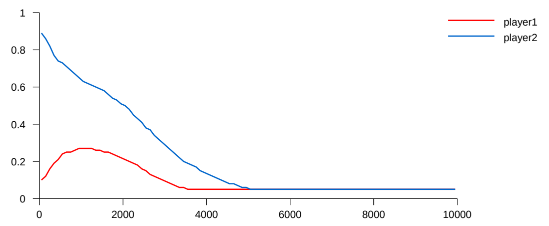
	c1	c2	c3
r1	0,0	1,0	0,1
r2	0,1	0,0	1,0
r3	1,0	0,1	0,0

(b) Shapley's

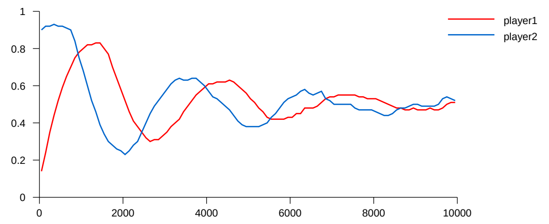
Table 2: 3-action games

Figures 6 and 7 plot the policy of the row player over time

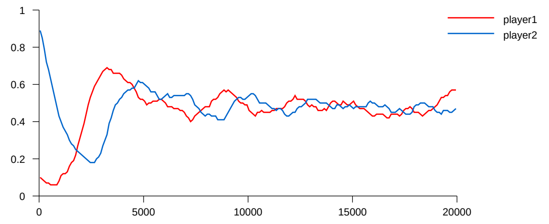
[6]We have tried increasing the learning rate of GIGA-WoLF to 0.01 in order to see if this can speed up its convergence. However, this resulted in GIGA-WoLF diverging.



(a) coordination



(b) matching pennies



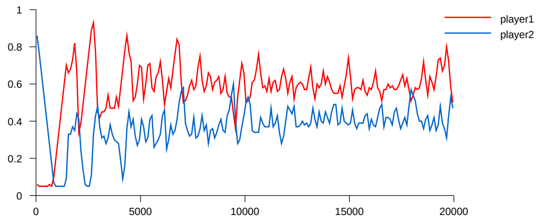
(c) tricky

Figure 4: Convergence of WLP in different two-player-two-action games. The figures plot the probability of playing the first action for each player (vertical axis) against time (horizontal axis).

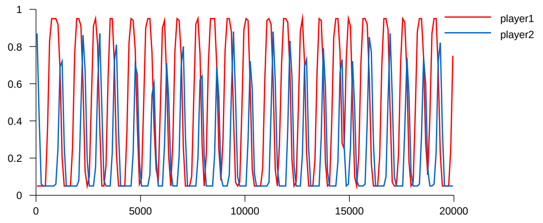
for the two games for both WPL and GIGA-WoLF when the initial joint strategy is $([0.1, 0.8, 0.1]_r, [0.8, 0.1, 0.1]_c)$, $\delta = 0.0003$, and for two values of α: 0.1 and 1. GIGA-WoLF's performance usually gets worse as α increases. GIGA-WoLF also keeps oscillating in Shapley's game regardless of the value of α. WLP on the other hand performs better as α increases and converges in Shapley's game when $\alpha = 1$. We believe the reason is that small α leads to an out of date reward estimate which in turn leads agents to continuously chase the NE without successfully reaching it. One of our future directions is to theoretically analyze this tradeoff between accuracy and recency of the reward and its effect on both convergence and optimality.

5.3 The Task Allocation Game

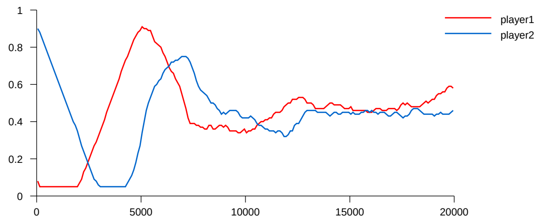
This section evaluates WPL in the distributed task allocation domain. Let N be the number of mediators and L



(a) PHC-WolF



(b) PHC-PdWolF

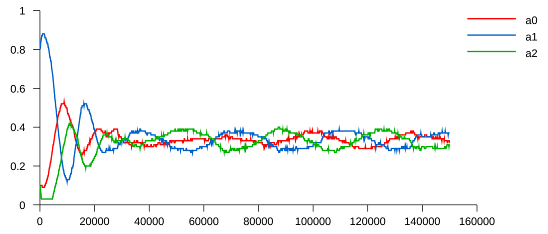


(c) GIGA-WolF

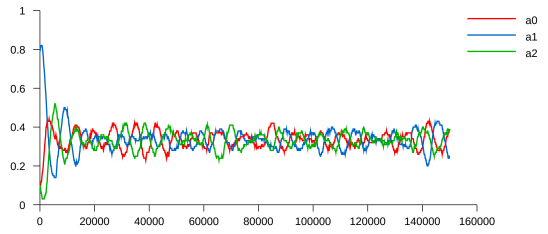
Figure 5: Convergence of the previous approaches in the tricky game. The figures plot the probability of playing the first action for each player (vertical axis) against time (horizontal axis).

be the number of servers. The turn-around-time, or TAT, is the time interval between a task arrival and its completion. An upper bound on the TAT can be computed using queuing theory as follows. Let μ_i be rate by which tasks are serviced at server i. Let λ_j be the rate by which tasks arrive at mediator j. Then the global task arrival rate $\lambda = \sum_j \lambda_j$ and the global service rate $\mu = \sum_i \mu_i$. The global utilization $u = \lambda/\mu$. Then a lower bound on the turn around time is given by the equation $\widehat{TAT} = \frac{1}{\mu}(1 + \frac{u}{1-u})$.

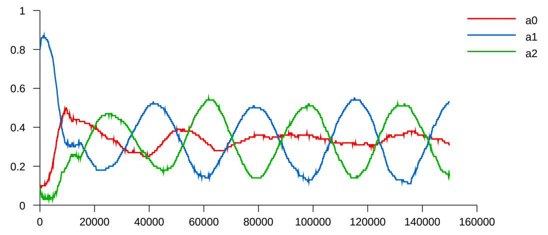
We define $\overline{TAT}$ to be the average TAT seen by each mediator when using WPL. Table 3 shows $\overline{TAT}/\widehat{TAT}$ (lower is better, where 1 is optimal) for different numbers of players N (columns) and different values of load u (rows). This data is collected from a system with 4 servers. Two servers have $\mu_i = 1$ task/time unit and the other two have $\mu = 0.5$ task/time unit. Tasks arrival rate is the same for all agents.



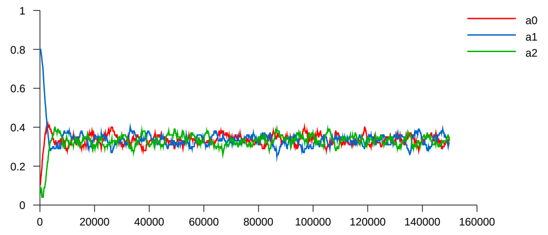
(a) GIGA-WolF, $\alpha = 0.1$



(b) WPL, $\alpha = 0.1$

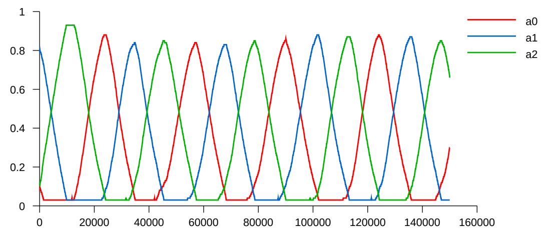


(c) GIGA-WolF, $\alpha = 1$

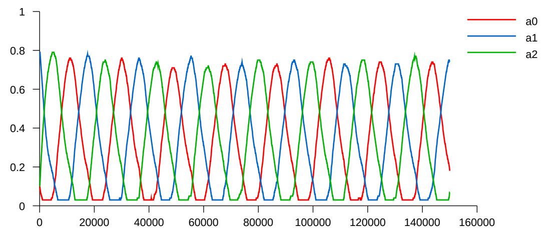


(d) WPL, $\alpha = 1$

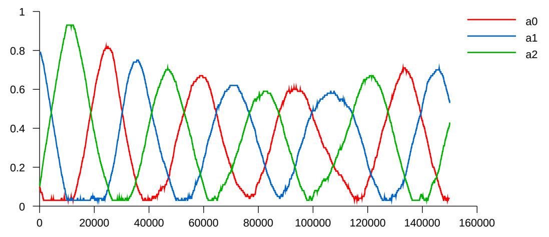
Figure 6: Convergence of GIGA-WoLF and WPL in the rock-paper-scissors game. The figures plot the probability of playing each action of the first player (vertical axis) against time (horizontal axis).



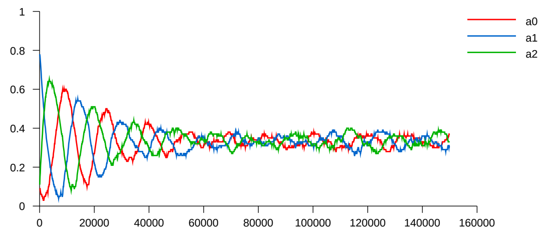
(a) GIGA-WolF, $\alpha = 0.1$



(b) WPL, $\alpha = 0.1$



(c) GIGA-WolF, $\alpha = 1$



(d) WPL, $\alpha = 1$

Figure 7: Convergence of GIGA-WoLF and WPL in Shapley's game. The figures plot the probability of playing each action of the first player (vertical axis) against time (horizontal axis).

The initial policy of all agents is to allocate all tasks to the first server. $\overline{TAT}/\widehat{TAT}$ approaches 1 as u increases. In other words, WPL is closer to optimal in high loads. This is expected because knowing the queue length of all servers is more advantageous in low loads (some servers may have an empty queue while others have nonempty queues). In high loads, all servers have on average constant queue length at all times. We can also notice that $\overline{TAT}/\widehat{TAT}$ does not change much for the same u and different N. This means that WPL scales well with the number of agents.

u / N	3	4	5
0.3	3.13	3.34	3.44
0.6	2.27	2.3	2.3
0.9	1.68	1.875	1.9

Table 3: $\overline{TAT}/\widehat{TAT}$ for different values of N (columns) and u (rows)

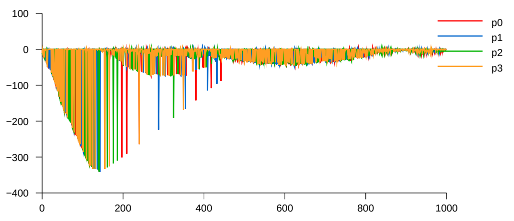
Figure 8 compares the performance of GIGA-WoLF and WPL with 5 mediators and 4 servers. The figure plots the TAT from individual mediators' perspective (TAT is plotted as a negative reward for each mediator). Note that the scale in GIGA-WoLF figure is almost double the scale in the WPL figure. There are two observations from the figures: time of convergence and maximum TAT. By 250 time steps, all mediators using the WLP algorithm witness TAT less than 50 time units. It took GIGA-WoLF more than 450 time units to achieve the same results. The maximum TAT that is ever witnessed by a WPL mediator is under 170 time units. GIGA-WoLF mediators have witnessed TAT of more than 320 time units.

Figure 9 illustrates how the policy of one player converges over time. As expected, for both WPL and GIGA-WoLF, the probability of sending a task to one of the fast servers (a2 and a3) becomes almost double the probability of sending a task to any of the slow servers. However, WPL learns a better policy, where the probabilities of sending a task to any of slower servers are equal (while in GIGA-WoLF there is a consistent difference).

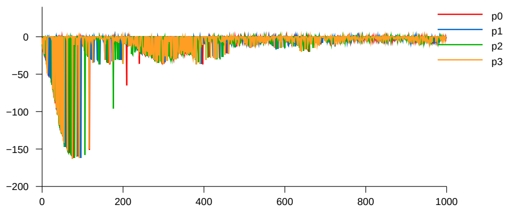
In conclusion, GIGA-WoLF converges slower than WPL and also reaches almost double the maximum TAT that WLP ever reaches. This gives WPL a clear advantage in dynamic environments where agents are continuously adapting to changes in the environment. We believe GIGA-WoLF's slow convergence is due to the way GIGA-WoLF works. GIGA-WoLF relies on learning a slowly moving policy in addition to the actual policy π to determine the direction of the gradient. This requires more time than the WPL algorithm, which changes its gradient direction according to the current expected reward.

6. CONCLUSION AND FUTURE WORK

This work presents WPL, a gradient ascent learning algorithm that outperforms state of the art multiagent learners in the task allocation. We experimentally show that our algorithm converges faster and is less sensitive to tuning parameters than other algorithms. We provide an informal proof of WPL's convergence. We also show that our algorithm converges in Shapley's game where many other algorithms fail to converge. Finally, we verify the practicality

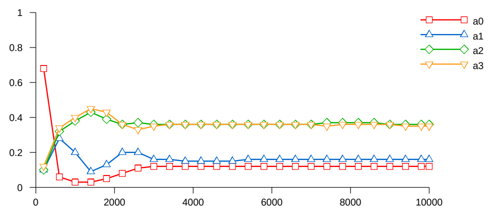


(a) GIGA-WolF

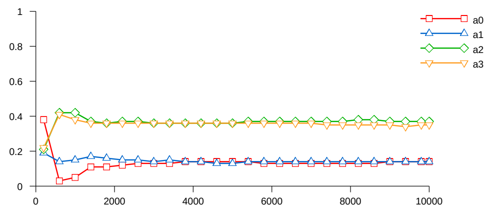


(b) WPL

Figure 8: Convergence of GIGA-WoLF and WPL in distributed task allocation. The figures plot the reward (negative TAT) obtained by each player (vertical axis) against time (horizontal axis).



(a) GIGA-WolF



(b) WPL

Figure 9: Convergence of GIGA-WoLF and WPL in distributed task allocation. The figures plot the policy of a player as the vertical axis (aka the probability of playing each action) against time (horizontal axis).

of our algorithm in the distributed task allocation domain, comparing its performance to an optimal global solution.

In the future, we want to extend our convergence guarantee to more than two players and two actions. We also want to evaluate WPL in more general task allocation settings where mediators may be connected to other mediators. This work also illustrates the importance of computing the expected reward and how it affects convergence. This issue was overlooked in previous work and assumed an implementation issue. Using more samples of the immediate reward may increase the accuracy of the reward expectation but may also make the expectation out of date. Finding the right balance and theoretically analyzing how it affects convergence are interesting research questions that we plan to address.

7. REFERENCES

[1] B. Banerjee and J. Peng. Adaptive policy gradient in multiagent learning. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems.*, 2003.

[2] M. Bowling. Convergence and no-regret in multiagent learning. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 209–216. MIT Press, Cambridge, MA, 2005.

[3] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[4] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

[5] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *International Conference on Machine Learning*, pages 83–90, 2003.

[6] J. Hu and M. P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[7] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *the 16th Conference on Uncertainty in Artificial Intelligence*, pages 541–548, 2000.

[8] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1999.

[9] X. Wang and T. Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In S. T. S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press.

[10] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *International Conference on Machine Learning*, pages 928–936, 2003.