Learning to Persist or Switch

Efficient and Fair Allocations in Large-scale Multi-agent Systems

Panayiotis Danassis, Boi Faltings
École Polytechnique Fédérale de Lausanne (EPFL)
Artificial Intelligence Laboratory
Lausanne, Switzerland
{panayiotis.danassis,boi.faltings}@epfl.ch

ABSTRACT

We present a multi-agent learning algorithm, ALMA-Learning, for efficient and fair allocations in *large-scale* systems. The proposed approach circumvents the traditional pitfalls of multi-agent learning (e.g., the moving target problem, the curse of dimensionality, or the need for mutually consistent actions) by relying on the ALMA heuristic as a coordination mechanism for each stage game. ALMA-Learning is decentralized and requires no communication between the agents. The latter only observe their own history of action/reward pairs.

Simulation results on a variety of scenarios demonstrate that ALMA-Learning can quickly reach near optimal allocations (less than 2.5% loss) in terms of social welfare, while providing higher fairness (up to almost 10% lower inequality in certain scenarios compared to the best performing baseline, i.e., the centralized, maximum-weight matching solution). The lightweight nature and the fast learning of the proposed approach constitute it ideal for *on-device* deployment in real-world applications.

KEYWORDS

Multi-agent Learning; Coordination and Cooperation; Assignment Problem; Weighted Matching; Resource Allocation; On-device Learning

1 INTRODUCTION

One of the most relevant problems in multi-agent systems is finding an optimal allocation between agents, i.e., computing a maximum-weight matching in a weighted (bipartite or general) graph, were edge weights correspond to the utility of each alternative. This pertains to role allocation (e.g., team formation for autonomous robots [Gunn and Anderson, 2013]), task assignment (e.g., agents in a smart factory, or taxi-passenger matching [Varakantham et al., 2012]), resource allocation (e.g., parking spaces / charging stations for autonomous vehicles [Danassis and Faltings, 2019, Geng and Cassandras, 2013]), etc. What follows is *applicable to any such scenario*, but for concreteness we will refer to the allocation of a set of resources to a set of agents in a bipartite graph, a setting known as the *assignment problem*, one of the most fundamental combinatorial optimization problems [Munkres, 1957].

When designing algorithms for the assignment problem, a significant challenge emerges from the nature of real-world applications, which is often distributed and *information-restrictive*. Sharing plans, utilities, or preferences creates high overhead, and, most importantly, there is often a lack of responsiveness and/or communication between the participants [Stone et al., 2010]. Achieving fast convergence and high efficiency in such information-restrictive settings is extremely challenging.

We recently proposed a heuristic (ALMA – ALtruistic MAtching – Heuristic [Danassis et al., 2019a]) that addresses the aforementioned challenges. ALMA is decentralized, completely uncoupled (agents are only aware of their *own history* of action/reward pairs), and requires *no communication* between the agents. A distinctive characteristic of ALMA is that agents make decisions locally, based on the contest for resources that *they* are interested in, and the agents that are interested in the *same* resources. If each agent is interested in only a *subset* of the total resources, ALMA converges in time polynomial in the maximum size of the subsets; not the total number of resources. Crucially, in the realistic case where the aforementioned quantities are bounded independently of the total number of agents/resources, the convergence time remains *constant* as the total problem size increases. The latter is true by default in many real-world applications (e.g., resource allocation in urban environments), since agents only have a local (partial) knowledge of the world, and there is typically a cost associated with acquiring a resource (e.g., a taxi driver would not be willing to drive to the other end of the city to pick up a passenger). Thus, ALMA utilizes a natural characteristic of the application domain, where other algorithms (e.g., the optimal centralized solution) would require time polynomial in the total number of agents/resources due to interdependencies. This *lightweight* nature of ALMA coupled with the *lack of inter-agent communication*, and the *highly efficient allocations*, make it ideal for an *on-device* solution for large-scale intelligent systems (e.g., IoT devices, smart cities and intelligent infrastructure, industry 4.0, autonomous vehicles, etc.). As an example application, we demonstrated in [Danassis et al., 2019b] that ALMA offers an efficient, end-to-end solution for the Dynamic Ridesharing and Fleet Relocation problem (as compared to 12 different algorithms over 12 metrics).

Despite ALMA's high performance in a variety of domains, it remains a heuristic; i.e., sub-optimal by nature. In this work, we build on the ideas of ALMA, and introduce a learning element (ALMA-Learning) that allows to quickly close the gap in social welfare compared to the optimal solution, while simultaneously increasing the fairness of the allocation. Specifically, in ALMA, while contesting for a resource, each agent will back-off with probability that depends on their *own utility loss* of switching to some alternative. ALMA-Learning improves upon ALMA by allowing agents to learn the expected loss of utility they will incur by backing-off, and the expected reward, which helps guide their search.

ALMA-Learning is applicable in repeated allocation games (e.g., self organization of intelligent infrastructure, autonomous mobility systems, etc.). As an example application, the Swiss Federal Railways recently launched a challenge to enable trains to automatically coordinate their route selection using multi-agent technologies (see https://www.aicrowd.com/challenges/flatland-challenge). Treating alternative routes as resources, with utility proportional to the delay compared to the shortest route, we can apply ALMA-Learning to quickly find an efficient and fair allocation, while easily being able to handle the scale of such domain[1]. Finally, ALMA-Learning can be also applied as a negotiation protocol in one-shot interactions, where agents can simulate the learning process offline, before making their final decision.

1.1 Our Contributions

Our main contributions in this paper are:

(1) We introduce a novel multi-agent (meta-)learning algorithm (**ALMA-Learning**), build on top of the ALMA heuristic of [Danassis et al., 2019a], for maximum-weight matching in both bipartite and general graphs. ALMA-Learning is decentralized, completely uncoupled (agents are only aware of their *own history* of action/reward pairs), and requires *no communication* between the agents. Coupled with its *lightweight*, and *scalable* nature, the above constitute ALMA-Learning ideal for *on-device* deployment in real-world applications.

(2) We prove that ALMA-Learning converges, and provide a thorough empirical evaluation in a variety of scenarios. In all of them ALMA-Learning is able to quickly reach allocations of high social welfare (less than 2.5% loss) and fairness (up to almost 10% lower inequality in certain scenarios compared to the best performing baseline, i.e., the centralized, maximum-weight matching solution).

1.2 Discussion and Related Work

Finding a maximum weight matching in a weighted bipartite graph is one of the best-studied combinatorial optimization problems in the literature. The first polynomial time algorithm (with respect to the total number of nodes, edges) was introduced by Jacobi in the 19th century [Borchardt and Jacobi, 1865, Ollivier, 2009], and was succeeded by many classical algorithms [Bertsekas, 1979, Edmonds and Karp, 1972, Munkres, 1957] with the *Hungarian* algorithm of Kuhn [1955] being the most prominent one (see [Su, 2015] for an overview). The problem can also be solved via linear programming [Dantzig, 1990], as its LP formulation relaxation admits integral optimal solutions [Papadimitriou and Steiglitz, 1982]. The more general *maximum weight matching* problem on general graphs has also been extensively studied (see [Lovász and Plummer, 2009]), with the most prominent algorithm being the *blossom algorithm* of [Edmonds, 1965].

In reality, a centralized coordinator is not always available, and if so, it has to know the utilities of all the participants, which is often not feasible. In the literature of the assignment problem, there also exist several decentralized algorithms (e.g., [Bürger et al., 2012, Giordani et al., 2010, Ismail and Sun, 2017, Zavlanos et al., 2008] which are the decentralized versions of the aforementioned well-known centralized algorithms – see also [Elkin, 2004, Kuhn et al., 2016] for general results in distributed approximability under only local information/computation). These algorithms require polynomial computational time and polynomial number of messages (such as cost matrices [Ismail and Sun, 2017], pricing information [Zavlanos et al., 2008], or a basis of the LP [Bürger et al., 2012], etc.).

While the problem has been 'solved' from an algorithmic perspective – having both centralized and decentralized polynomial algorithms – it is not so from the perspective of multi-agent systems, for two key reasons: (1) complexity, and (2) communication. The proliferation of intelligent systems will give rise to *large-scale*, multi-agent based technologies. Algorithms for maximum-weight matching, whether centralized or distributed, have runtime that increases with the total problem size, even if agents are interested in a small number of resources. Thus, they can only handle problems of some bounded size. Moreover, they require a significant amount of inter-agent communication. As the number and diversity of autonomous agents continue to rise, differences in origin, communication protocols, or the existence of sub-optimal, legacy agents will bring forth the need to collaborate without any form of explicit communication [Stone et al., 2010]. Most importantly though, communication between participants (sharing utility tables, plans, and preferences) creates high overhead. ALMA on the other hand achieves *constant* in the total problem size running time – under reasonable assumptions on the preference domain of the agents – while being, to the best of our knowledge, the only decentralized algorithm that requires no message exchange (i.e., no communication network) between the participating agents [Danassis et al., 2019a]. The proposed approach, ALMA-Learning, maintains the aforementioned two properties of ALMA.

From the perspective of Multi-Agent Learning (MAL), the problem at hand falls under the paradigm of multi-agent reinforcement learning, where for example it can be modeled as a Multi-Armed Bandit (MAB) problem [Auer et al., 2002], or as a Markov Decision Process (MDP) and solved using Q-Learning [Busoniu et al., 2008]. In MAB problems an agent is given a number of arms (resources) and at each time-step has to decide which arm to pull to get the maximum expected reward. In Q-learning agents solve Bellman's optimality equation [Bellman, 2013] using an iterative approximation procedure so as to maximize some notion of expected cumulative reward. Both approaches have arguably been designed to operate in a more challenging setting, thus making them susceptible to many pitfalls inherent in MAL. For example, there is no stationary distribution, in fact, rewards depend on the joint action of the agents and since all agents learn simultaneously, this results to a moving-target problem. Thus, there is an inherent need for coordination in MAL algorithms, stemming from the fact that the effect of an agent's action depends on the actions of the other agents, i.e. actions must be mutually consistent to achieve the desired result. Moreover, the curse of dimensionality makes it difficult to apply such algorithms to large scale problems. ALMA-Learning solves both of the above challenges *by relying on ALMA as a coordination mechanism for each stage of the repeated game.* Another fundamental difference is that the aforementioned algorithms

[1] Today, there are more than 10,000 trains running each day in Switzerland, while due to the growing demand for mobility, the Swiss Federal Railways needs to increase the transportation capacity of the network by approximately 30% in the future (see https://www.aicrowd.com/challenges/flatland-challenge).

are designed to tackle the exploration/exploitation dilemma. A bandit algorithm for example will constantly explore, even if an agent has acquired his most preferred alternative. In matching problems, though, agents know (or have an estimate of) their own utilities. ALMA-Learning in particular, requires the knowledge of personal preference ordering and pairwise differences of utility (which are far easier to estimate than the exact utility table). The latter gives a great advantage to ALMA-Learning, since agents do not need to continue exploring after successfully claiming a resource, which stabilizes the learning process.

2 PROPOSED APPROACH: ALMA-LEARNING

In this section, we define ALMA-Learning and prove its convergence properties. We start by defining the assignment problem.

2.1 The Assignment Problem

The assignment problem consists of finding a maximum weight matching in a weighted bipartite graph, $\mathcal{G} = \{\mathcal{N} \cup \mathcal{R}, \mathcal{E}\}$. In the studied scenario, $\mathcal{N} = \{1, \ldots, N\}$ agents compete to acquire $\mathcal{R} = \{1, \ldots, R\}$ resources. The weight of an edge $(n, r) \in \mathcal{E}$ represents the utility ($u_n(r) \in [0, 1]$) agent n receives by acquiring resource r. Each agent can acquire at most one resource, and each resource can be assigned to at most one agent. The goal is to maximize the social welfare (sum of utilities), i.e., $\max_{\mathbf{x} \geq 0} \sum_{(n,r)\in\mathcal{E}} u_n(r) x_{n,r}$, where $\mathbf{x} = (x_{1,1}, \ldots, x_{N,R})$, subject to $\sum_{r|(n,r)\in\mathcal{E}} x_{n,r} = 1, \forall n \in \mathcal{N}$, and $\sum_{n|(n,r)\in\mathcal{E}} x_{n,r} = 1, \forall r \in \mathcal{R}$.

2.2 Learning Rule

We begin by describing (a slightly modified version of) the ALMA heuristic of [Danassis et al., 2019a], which is used as a subroutine by ALMA-Learning. The pseudo-codes for ALMA and ALMA-Learning are presented in Algorithms 1 and 2, respectively. To improve readability, we have omitted the subscript n from all the variables and arrays in Algorithms 1 and 2. Both ALMA and ALMA-Learning are run independently and in parallel by all the agents.

We make the following two assumptions: First, we assume (possibly noisy) knowledge of personal utilities by each agent. Second, we assume that agents can observe feedback from their environment. This is used to inform collisions and detect free resources. It could be achieved by the use of visual, auditory, olfactory sensors etc., or by any other means of feedback from the resource (e.g., by sending an occupancy message). Note here that these messages would be between the requesting agent and the resource, not between the participating agents themselves, and that it suffices to send only 1 bit of information (e.g., 0, 1 for occupied / free respectively).

For both ALMA, and ALMA-Learning, each agent sorts his available resources (possibly $\mathcal{R}^n \subseteq \mathcal{R}$) in decreasing utility ($r_0, r_1, \ldots, r_i, r_{i+1}, \ldots, r_{R^n-1}$) under his preference ordering $\prec_n$.

2.2.1 ***ALMA: ALtruistic MAtching Heuristic.*** ALMA converges to a resource through repeated trials as follows. The set of available actions is denoted as $\mathcal{A} = \{Y, A_{r_1}, \ldots, A_{r_{R^n}}\}$, where Y refers to yielding, and A_r refers to accessing resource r. Each agent has a strategy,g, that points to a resource. As long as an agent has not acquired a resource yet, at every time-step, there are two possible scenarios. If $g = A_r$ (strategy points to resource r), then agent n attempts to acquire that resource. If there is a collision, the colliding parties back-off with some probability. Otherwise, if $g = Y$, the agent choses a resource r for monitoring. If the resource is free, he sets $g \leftarrow A_r$. Algorithm 1 presents the pseudo-code of ALMA, which is followed by every agent individually.

Algorithm 1 ALMA: Altruistic Matching Heuristic.

Require: Sort resources ($\mathcal{R}^n \subseteq \mathcal{R}$) in decreasing order of utility $r_0, r_1, \ldots, r_i, r_{i+1}, \ldots, r_{R^n-1}$ under $\prec_n$

1: **procedure** ALMA(r_{start}, $loss[R]$)
2: Initialize $g \leftarrow A_{r_{start}}$
3: Initialize $current \leftarrow -1$
4: Initialize $converged \leftarrow False$
5: **while** !$converged$ **do**
6: **if** $g = A_r$ **then**
7: Agent n attempts to acquire r
8: **if** Collision(r) **then**
9: Back-off (set $g \leftarrow Y$) with probability $P(loss[r])$
10: **else**
11: $converged \leftarrow True$
12: **else** ($g = Y$)
13: $current \leftarrow (current + 1) \bmod R$
14: Agent n monitors $r \leftarrow r_{current}$.
15: **if** Free(r) **then** set $g \leftarrow A_r$
16: **return** r, such that $g = A_r$

The back-off probability ($P(\cdot)$) is computed individually and locally based on each agent's expected loss. If more than one agent compete for resource r_i (step 8 of Algorithm 1), each of them will back-off with probability that depends on their expected utility loss. The expected loss array is computed by ALMA-Learning and provided as input to ALMA. The actual back-off probability can be computed with any monotonically decreasing function on $loss$ (see [Danassis et al., 2019a]). In this work we have used $P(loss) = f(loss)^\beta$, where β controls the aggressiveness (willingness to back-off), and $f(\cdot)$ is given by:

$$f(loss) = \begin{cases} 1 - \epsilon, & \text{if } loss \leq \epsilon \\ \epsilon, & \text{if } 1 - loss \leq \epsilon \\ 1 - loss, & \text{otherwise} \end{cases} \quad (1)$$

Using the aforedescribed rule, agents that do not have good alternatives will be less likely to back-off and vice versa. The ones that do back-off select an alternative resource and examine its availability. The resource selection is performed in sequential order, always starting from the most preferred resource (see step 3 of Algorithm 1).

2.2.2 ***Motivating Examples: Sources of Inefficiency.*** Despite ALMA's high performance in a variety of domains (as demonstrated in both [Danassis et al., 2019a] and [Danassis et al., 2019b]), it remains a heuristic; i.e., sub-optimal by nature. In this section we provide some adversarial examples; sources of inefficiency in terms of achieved social welfare, which motivated ALMA-Learning.

In the original ALMA algorithm, all agents start at their most preferred resource, and back-off with probability that depends on

Agents \ Resources	r_1	r_2	r_3
n_1	1	0	0.5
n_2	0	1	0
n_3	1	0.9	0

Table 1: Motivating adversarial example: Inaccurate loss estimate. Agent n_3 backs-off with high probability when contesting for resource r_1 assuming a good alternative, only to find resource r_2 occupied.

Agents \ Resources	r_1	r_2	r_3
n_1	1	0.9	0
n_2	0	1	0.9
n_3	1	0.9	0

Table 2: Motivating adversarial example: Inaccurate reward expectation. Agents n_1 and n_3 always start by attempting to acquire resource r_1, reasoning that it is the most preferred one, yet each of them only wins r_1 half of the times.

their loss of switching to the immediate next best resource. Specifically, in the simplest case, the probability to back-off when contesting resource r_i would be given by $P(loss(i)) = 1 - loss(i)$, where $loss(i) = u_n(r_i) - u_n(r_{i+1})$ and r_{i+1} is the next best resource according to agent n's preferences $\prec_n$.

The first example is given in Table 1. Agent n_3 backs-off with high probability (higher than agent n_1) when contesting for resource r_1 assuming a good alternative, only to find resource r_2 occupied. Thus, n_3 ends up matched with resource r_3. The social welfare of the final allocation is 2, which is 20% worse than the optimal (which is for agents n_1, n_2, n_3 to be matched with resources r_3, r_2, r_1, respectively, achieving a social welfare of 2.5). ALMA-Learning solves this problem by learning an empirical estimate of the loss an agent will incur if he backs-off from a resource. In this case, agent n_3 will learn that his loss is not $1 - 0.9 = 0.1$, but actually $1 - 0 = 1$, and thus will not back-off, resulting in an optimal allocation.

Table 2 presents the second example. Agents n_1 and n_3 always start by attempting to acquire resource r_1, reasoning that it is the most preferred one. Yet, in a repeated game, each of them only wins r_1 half of the times (achieving social welfare 2, 28.5% worse than the optimal 2.8), thus, in expectation, resource r_1 has utility 0.5. ALMA-Learning solves this problem by learning an empirical estimate of the reward of each resource. In this case, after learning, either agent n_1 or n_3 (or both), will start from resource r_2. Agent n_2 will back-off since he has a good alternative, and the result will be the optimal allocation where agents n_1, n_2, n_3 are matched with resources r_2, r_3, r_1 (or r_1, r_3, r_2), respectively.

2.2.3 ***ALMA-Learning: A Multi-Agent (Meta-)Learning Algorithm.*** ALMA-Learning uses ALMA as a sub-routine, specifically as a coordination mechanism for each stage of the repeated game. Over time, ALMA-Learning learns which resource to select first (r_{start}) when running ALMA, and an accurate empirical estimate on the loss it will incur by backing-off ($loss[]$). By learning these two values, agents take more informed decisions, specifically:

Algorithm 2 ALMA-Learning

Require: Sort resources ($\mathcal{R}^n \subseteq \mathcal{R}$) in decreasing order of utility $r_0, r_1, \ldots, r_i, r_{i+1}, \ldots, r_{R^n-1}$ under $\prec_n$
Require: $rewardHistory[R][L]$, $reward[R]$, $loss[R]$
1: **procedure** ALMA-Learning
2: **for all** $r \in \mathcal{R}$ **do** ▷ Initialization
3: $rewardHistory[r]$.add($u(r)$)
4: $reward[r] \leftarrow rewardHistory[r]$.getMean()
5: $loss[r] \leftarrow u(r) - u(r_{next})$
6: $r_{start} \leftarrow \arg\max_r reward[r]$
7:
8: **for** $t \in [1, \ldots, T]$ **do** ▷ T: Time horizon
9: $r_{won} \leftarrow ALMA(r_{start}, loss[])$ ▷ Run ALMA
10:
11: $rewardHistory[r_{start}]$.add($u(r_{won})$)
12: $reward[r_{start}] \leftarrow rewardHistory[r_{start}]$.getMean()
13: **if** $u(r_{start}) - u(r_{won}) > 0$ **then**
14: $loss[r_{start}] \leftarrow$
15: $(1-\alpha)loss[r_{start}] + \alpha\,(u(r_{start}) - u(r_{won}))$
16:
17: **if** $r_{start} != r_{won}$ **then**
18: $r_{start} \leftarrow \arg\max_r reward[r]$

(1) If an agent often loses the contest of his starting resource, the expected reward of that resource will decrease, thus in the future the agent will switch to an alternative starting resource, and (2) if an agent backs-off from contesting resource r expecting low loss, only to find that all his high utility alternatives are already occupied, then his expected loss of resource r ($loss[r]$) will increase, making him more reluctant to back-off in some future stage game. In more detail, ALMA-Learning learns and maintains the following information[2]:

(i) $rewardHistory[R][L]$: A 2D array. For each $r \in \mathcal{R}$ it maintains the L most recent reward values received by agent n, i.e., the L most recent $u_n(r_{won})$, where $r_{won} \leftarrow ALMA(r, loss[])$. See line 11 of Algorithm 2. The array is initialized to the utility of each resource (line 3 of Algorithm 2).

(ii) $reward[R]$: A 1D array. For each $r \in \mathcal{R}$ it maintains an empirical estimate on the expected reward received by starting at resource r and continue playing according to Algorithm 1. It is computed by averaging the reward history of the resource, i.e., $\forall r \in \mathcal{R} : reward[r] \leftarrow rewardHistory[r].getMean()$. See line 12 of Algorithm 2.

(iii) $loss[R]$: A 1D array. For each $r \in \mathcal{R}$ it maintains an empirical estimate on the loss in utility agent n incurs if he backs-off from the contest of resource r. The loss of each resource r is initialized to $loss[r] \leftarrow u_n(r) - u_n(r_{next})$, where r_{next} is the next most preferred resource to r, according to agent n's preferences $\prec_n$ (see line 5 of Algorithm 2). Subsequently, for every stage game, agent n starts by selecting resource r_{start}, and ends up wining resource r_{won}. The loss of r_{start} is then updated according to the following averaging

[2]We remind the reader that to improve readability we have omitted the subscript $_n$ from all the variables and arrays, but every agent maintains their own estimates.

process:

$$loss[r_{start}] \leftarrow (1-\alpha) loss[r_{start}] + \alpha\,(u(r_{start}) - u(r_{won}))$$

where α is the learning rate.

Finally, the last condition in the pseudo-code (lines 17-18 of Algorithm 2) prevent agents with good alternatives to continue exploring. It ensures that an agent only attempts to switch his starting resource if he failed to win said resource in the previous stage game. Note that ties in the calculation of the argument of the maxima (lines 18 and 6 of Algorithm 2) are broken randomly.

2.3 Convergence

In this section we will prove that ALMA-Learning converges. Convergence of ALMA-Learning does not translate to a fixed allocation at each stage game after convergence. The system has converged when agents no longer switch their starting resource, r_{start}. The final allocation of each stage game is controlled by ALMA, which means that even after convergence there can be contest for a resource, i.e., having more than one agent selecting the same starting resource. As we will demonstrate later, this translates to fairer allocations, since agents with similar preferences can alternate between acquiring their most preferred resource.

Theorem 2.1. *There exists time-step $t_{converged}$ such that $\forall t > t_{converged} : r^n_{start}(t) = r^n_{start}(t_{converged})$, where $r^n_{start}(t)$ denotes the starting resource r_{start} of agent n at the stage game of time-step t.*

Proof. Theorem 2.1 of [Danassis et al., 2019a] proves that ALMA (called at line 9 of Algorithm 2) converges in polynomial time.

In fact, under the assumption that each agent is interested in a subset of the total resources (i.e., $\mathcal{R}^n \subset \mathcal{R}$) and thus at each resource there is a bounded number of competing agents ($\mathcal{N}^r \subset \mathcal{N}$) Corollary 2.1.1 of [Danassis et al., 2019a] proves that the expected number of steps any individual agent requires to converge is independent of the total problem size (i.e., N and R). In other words, by bounding these two quantities (i.e., we consider R^n and N^r to be constant functions of N, R), the convergence time of ALMA is *constant* in the total problem size N, R. Thus, under the aforementioned assumptions:

Each stage game converges in constant time.

Now that we have established that the call to the ALMA procedure will return, the key observation to prove convergence for ALMA-Learning is that agents switch their starting resource only when the expected reward for the current starting resource drops below the best alternative one, i.e., for an agent to switch from r_{start} to r'_{start}, it has to be that $reward[r_{start}] < reward[r'_{start}]$. Given that utilities are bounded in $[0, 1]$, there is a maximum, finite number of switches until $reward_n[r] = 0, \forall r \in \mathcal{R}, \forall n \in \mathcal{N}$. In that case, the problem is equivalent to having N balls thrown randomly and independently into N bins (since $R = N$). Since both R, N are finite, the process will result in a distinct allocation in finite steps with probability 1. In more detail, we can make the following arguments:

(i) Let r_{start} be the starting resource for agent n, and $r'_{start} \leftarrow \arg\max_{r \in \mathcal{R}/\{r_{start}\}} reward[r]$. There are two possibilities. Either $reward[r_{start}] > reward[r'_{start}]$ for all time-steps $t > t_{converged}$ – i.e., $reward[r_{start}]$ can oscillate but always stays larger than $reward[r'_{start}]$ – or there exists time-step t when $reward[r_{start}] < reward[r'_{start}]$, and then agent n switches to the starting resource r'_{start}.

(ii) Only the reward of the starting resource r_{start} changes at each stage game. Thus, for the reward of a resource to increase, it has to be the r_{start}. In other words, at each stage game that we select r_{start} as the starting resource, the reward of every other resource remains (1) unchanged and (2) $reward[r] < reward[r_{start}], \forall r \in \mathcal{R}\ \{r_{start}\}$ (except when an agent switches starting resources).

(iii) There is a finite number of times each agent can switch his starting resource r_{start}. This is because $u_n(r) \in [0, 1]$ and $|u_n(r) - u_n(r')| > \delta, \forall n \in \mathcal{N}, r \in \mathcal{R}$, where δ is a small, strictly positive minimum increment value. This means that either the agents will perform the maximum number of switches until $reward_n[r] = 0, \forall r \in \mathcal{R} \forall n \in \mathcal{N}$ (which will happen in finite number of steps), or the process will have converged before that.

(iv) If $reward_n[r] = 0, \forall r \in \mathcal{R}, \forall n \in \mathcal{N}$, the question of convergence is equivalent to having N balls thrown randomly and independently into R bins and asking whether you can have exactly one ball in each bin – or in our case, where $N = R$, have no empty bins. The probability of bin r being empty is $\left(\frac{R-1}{R}\right)^N$, i.e., being occupied is $1 - \left(\frac{R-1}{R}\right)^N$. The probability of all the bins to be occupied is $\left(1 - \left(\frac{R-1}{R}\right)^N\right)^R$. The expected number of trials until this event occurs is $1/\left(1 - \left(\frac{R-1}{R}\right)^N\right)^R$, which is finite, for finite N, R. □

3 EVALUATION

In this section we evaluate ALMA-Learning under various test cases. We focus on (a) the relative difference, i.e., $(achieved - optimal)/optimal$, in social welfare (SW) compared to the optimal – in terms of achieved social welfare – allocation and (b) on fairness. We run each configuration 16 times and report the average values. Since we have randomized algorithms, we also run each problem instance of each configuration 16 times. Error bars represent one standard deviation (SD) of uncertainty. In all of the simulations, ALMA, and ALMA-Learning's parameters were set to: $\alpha = 0.1, \beta = 2, \epsilon = 0.01, L = 20$.

3.0.1 ***Employed Baselines.*** We compare ALMA-Learning to three baselines:

(a) *The Hungarian algorithm*: The most prominent algorithm for computing an maximum-weight matching in a bipartite graph [Kuhn, 1955].

(b) *The ALMA heuristic* of [Danassis et al., 2019a].

(c) *The Greedy algorithm*: It goes through the agents randomly, and assigns them their most preferred, unassigned resource. Greedy approaches are appealing in real-life, large-scale systems, not only due to their low complexity, but also because real-time constraints dictate short planning windows which would potentially diminish the benefit of batch optimization solutions compared to myopic approaches [Widdows et al., 2017].

	Resources		
Agents	r_1	r_2	r_3
n_1	1	0.5	0
n_2	0	1	0
n_3	1	0.75	$\epsilon \rightarrow 0$

Table 3: Motivating adversarial example: Unfair allocation. Both ALMA (with higher probability) and any optimal allocation algorithm will assign the coveted resource r_1 to agent n_1, while n_3 will receive utility 0.

3.0.2 ***Fairness Metrics.*** The usual predicament of efficient allocations is that they assign the resources only to a fixed subset of agents, which leads to an unfair result. Consider the simple example of Table 3. Both ALMA (with higher probability) and any optimal allocation algorithm will assign the coveted resource r_1 to agent n_1, while n_3 will receive utility 0. But, using ALMA-Learning, agents n_1 and n_3 will update their expected loss for resource r_1 to 1, and randomly acquire it between stage games, increasing fairness. Recall that convergence for ALMA-Learning does not translate to a fixed allocation at each stage game after convergence. To capture the fairness of this 'mixed' allocation, we report the average fairness on 32 evaluation time-steps that follow the training period.

Given the broad literature on fairness, we opted to measure two different fairness indices:

(a) *The Jain index* [Jain et al., 1998]: Widely used in network engineering to determine whether users or applications receive a fair share of system resources. It exhibits a lot of desirable properties such as: population size independence, continuity, scale and metric independence, and boundedness. For an allocation of N agents, such that the n^{th} agent is alloted x_n, the Jain index is given by Equation 2. An allocation $\mathbf{x} = (x_1, \ldots, x_N)^\top$ is considered fair, iff $\mathbb{J}(\mathbf{x}) = 1$.

$$\mathbb{J}(\mathbf{x}) = \frac{\left(\sum_{n=1}^{N} x_n\right)^2}{N \sum_{n=1}^{N} x_n^2} \tag{2}$$

(b) *The Gini coefficient* [Gini, 1912]: One of the most commonly used measures of inequality by economists intended to represent the wealth distribution of a population of a nation. For an allocation game of N agents, such that the n^{th} agent is alloted x_n, the Gini coefficient is given by Equation 3. A Gini coefficient of zero expresses perfect equality, i.e., an allocation is fair iff $\mathbb{G}(\mathbf{x}) = 0$.

$$\mathbb{G}(\mathbf{x}) = \frac{\sum_{n=1}^{N} \sum_{n'=1}^{N} |x_n - x_{n'}|}{2N \sum_{n=1}^{N} x_n} \tag{3}$$

3.0.3 ***Evaluation Scenarios.*** We evaluate ALMA-Learning under three test-cases:

(a) *Map*: Consider a Cartesian map on which the agents and resources are randomly distributed. The utility received by agent n for acquiring resource r is proportional to the inverse of their distance, i.e., $u_n(r) = 1/d_{n,r}$. Let $d_{n,r}$ denote the Manhattan distance. We assume a grid length of size $\sqrt{4 \times N}$.

(b) *Noisy Common Utilities*: This pertains to an anti-coordination scenario, i.e., competition between agents with similar preferences. We model the utilities as: $\forall n, n' \in \mathcal{N}, |u_n(r) - u_{n'}(r)| \leq$ noise, where the noise is sampled from a zero-mean Gaussian distribution, i.e., noise $\sim \mathcal{N}(0, \sigma^2)$.

(c) *Binary Utilities*: This corresponds to each agent being indifferent to acquiring any resource amongst his set of desired resources. For each agent n and for each resource r, $u_n(r)$ is randomly assigned to 0 or 1.

3.1 Results: Social Welfare

We begin with the loss in social welfare compared to the optimal solution. Figures 1a, 2a, and 3a present the results for the three test-cases (the Hungarian algorithm computes the optimal solution, thus is omitted from the graphs).

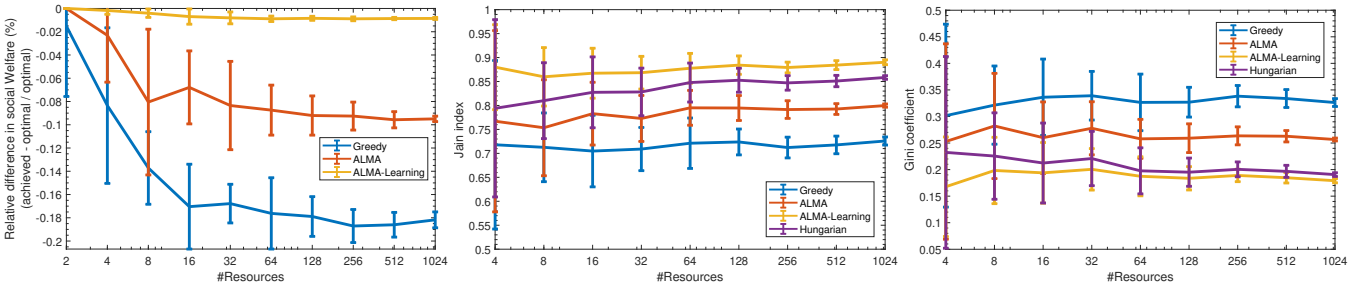
ALMA-Learning loses 0.00% to 0.89% in the test-case (a) Map after 512 training time-steps (0.00% to 1.68% after only 64 training time-steps), 1.34% to 2.26% in the test-case (b) Noisy Common Utilities, $\sigma = 0.1$, and finally 0.00% to 0.39% in the test-case (c) Binary Utilities after 64 training time-steps. ALMA loses 0.00% to 9.57%, 2.96% to 10.58%, and 0.00% to 16.88% in the three test-cases, respectively. Finally, Greedy loses 1.51% to 18.71%, 8.13% to 12.86%, and 0.10% to 14.70%.

We also run test-case (b) Noisy Common Utilities for $\sigma = 0.2$ and $\sigma = 0.4$, but for better visualization we excluded the results from Figure 2a, opting instead to plot the worst performing scenario for ALMA-Learning ($\sigma = 0.1$). ALMA-Learning loses 0.35% to 1.97% for $\sigma = 0.2$, and 0.05% to 2.26% for $\sigma = 0.4$. As a reference, ALMA loses 1.37% to 12.33%, and 0.75% to 10.74%, while Greedy loses 5.40% to 14.11%, and 0.79% to 12.64% for $\sigma = 0.2$, and $\sigma = 0.4$, respectively.

In all of the test-cases, ALMA-Leaning is able to quickly reach *near-optimal* allocations. On par with previous results [Danassis et al., 2019a,b], ALMA loses around 10% to 15%[3]. Is is worth noting that test-cases that are 'harder' for ALMA – specifically test-case (a) Map, where ALMA maintains the same gap on the optimal solution as the number of resources grow, and test-case (c) Binary Utilities, where ALMA exhibits the highest loss for 16 resources[4] – are 'easier' to learn for ALMA-Learning. In the aforementioned two test cases, ALMA-Learning was able to learn near-optimal to optimal allocations in just $64 - 512$ training time-steps (in fact in certain cases it learns near-optimal allocations in as little as 32 time-steps). Contrary to that, in test-case (b) Noisy Common Utilities, ALMA-Learning requires significantly more time to learn (we trained for 8192 time-steps), especially for larger games ($R > 256$). Intuitively we believe that this is because ALMA already starts with a near optimal allocation, and given the high similarity on the agent's utility tables (especially for $\sigma = 0.1$), it requires a lot of fine-tunning to improve the result.

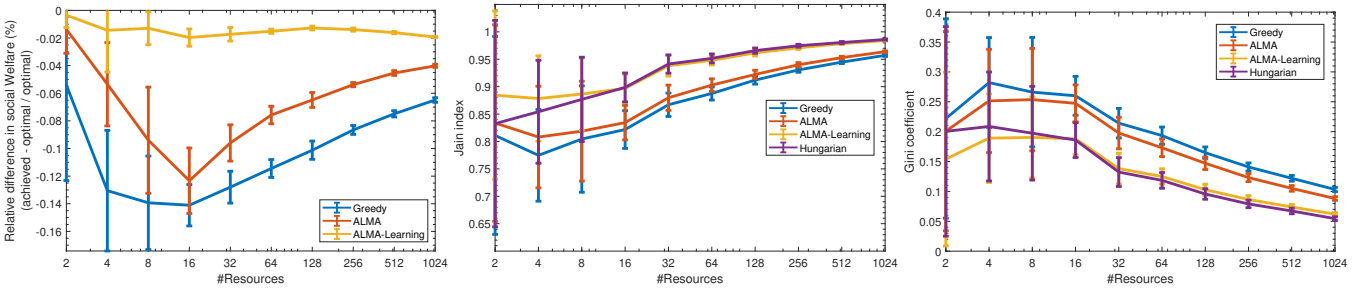
[3]Note that both ALMA and ALMA-Learning use the same function $P(loss) = f(loss)^\beta$ (see Equation 1) to compute the back-off probability, in order to provide a fair common ground for the evaluation.

[4]Binary utilities represent a somewhat adversarial test-case for ALMA, since the agents can not utilize the more sophisticated back-off mechanism based on the loss (loss is either 1, or 0 in this case).



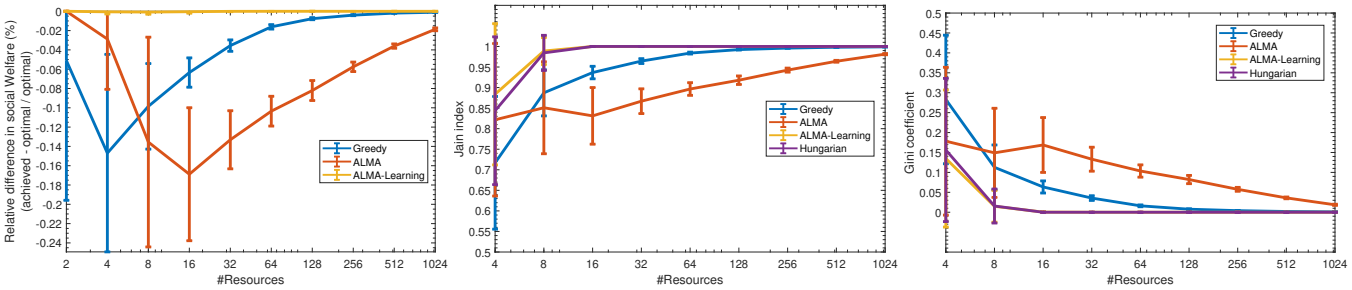
(a) Relative Difference in Social Welfare (%) **(b) Jain Index (higher is better)** **(c) Gini Coefficient (lower is better)**

Figure 1: Test-case (a): Map, we report the relative difference in social welfare, the Jain index, and the Gini coefficient, for increasing number of resources ($[2, 1024]$, x-axis in log scale), and $N = R$. For each problem instance, we trained ALMA-Learning for 512 time-steps.



(a) Relative Difference in Social Welfare (%) **(b) Jain Index (higher is better)** **(c) Gini Coefficient (lower is better)**

Figure 2: Test-case (b): Noisy Common Utilities, $\sigma = 0.1$, we report the relative difference in social welfare, the Jain index, and the Gini coefficient, for increasing number of resources ($[2, 1024]$, x-axis in log scale), and $N = R$. For each problem instance, we trained ALMA-Learning for 8192 time-steps.



(a) Relative Difference in Social Welfare (%) **(b) Jain Index (higher is better)** **(c) Gini Coefficient (lower is better)**

Figure 3: Test-case (c): Binary Utilities, we report the relative difference in social welfare, the Jain index, and the Gini coefficient, for increasing number of resources ($[2, 1024]$, x-axis in log scale), and $N = R$. For each problem instance, we trained ALMA-Learning for 64 time-steps.

3.2 Results: Fairness

Next, we evaluate the fairness of the final allocation. Figures 1b, 2b, and 3b depict the Jain index (higher is better) for the three test-cases, while the Gini coefficient (lower is better) is presented in Figures 1c, 2c, and 3c.

In all of the test-cases and for both indices, ALMA-Leaning achieves the most fair allocations, fairer than the optimal (in terms of social welfare) solution. In particular, ALMA-Learning's Jain index is between 0.86 to 0.89 (11.95%, 22.44%, and 5.03% higher on average than ALMA, Greedy, and Hungarian, respectively) in the test-case (a) Map, 0.85 to 0.93 (5.58%, 7.58%, and 1.81% higher on average than ALMA, Greedy, and Hungarian, respectively) in

the test-case (b) Noisy Common Utilities, $\sigma = 0.1$, and finally 0.88 to 1.00 (10.18%, 5.36%, and 0.58% higher on average than ALMA, Greedy, and Hungarian, respectively) in the test-case (c) Binary Utilities. ALMA's Jain index is between 0.75 to 0.80, 0.79 to 0.89, and 0.82 to 0.98, Greedy's Jain index is between 0.70 to 0.73, 0.77 to 0.88, and 0.72 to 1.00, and finally, Hungarian's Jain index is between 0.79 to 0.86, 0.81 to 0.92, and 0.84 to 1.00 in the three test-cases, respectively.

Moving on to the Gini coefficient, ALMA-Learning achieves between 0.17 to 0.20 (−29.04%, −42.91%, and −9.63% lower on average than ALMA, Greedy, and Hungarian, respectively) in the test-case (a) Map, 0.16 to 0.23 (−18.29%, −23.66%, and −6.52% lower on average than ALMA, Greedy, and Hungarian, respectively) in the test-case (b) Noisy Common Utilities, $\sigma = 0.1$, and finally 0.00 to 0.13 (−90.43%, −92.61%, and −0.18% lower on average than ALMA, Greedy, and Hungarian, respectively) in the test-case (c) Binary Utilities. ALMA's Gini coefficient is between 0.25 to 0.28, 0.19 to 0.28, and 0.02 to 0.18, Greedy's Gini coefficient is between 0.30 to 0.34, 0.21 to 0.29, and 0.00 to 0.28, and finally, Hungarian's Gini coefficient is between 0.19 to 0.23, 0.17 to 0.24, and 0.00 to 0.16 in the three test-cases, respectively.

4 CONCLUSION

The next technological revolution will be interwoven to the proliferation of intelligent systems. To truly allow for scalable solutions, we need to shift from traditional approaches to multi-agent solutions, ideally run *on-device*. In this paper, we present a novel learning algorithm (ALMA-Learning), which exhibits such properties, to tackle a central challenge in multi-agent systems: finding an optimal allocation between agents, i.e., computing a maximum-weight matching. We prove that ALMA-Learning converges, and provide a thorough empirical evaluation in a variety of scenarios. In all of them ALMA-Learning is able to quickly (in as little as 64 training time-steps) reach allocations of high social welfare (less than 2.5% loss) and fairness.

REFERENCES

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002), 48–77.

Richard Bellman. 2013. *Dynamic programming*. Courier Corporation.

Dimitri P. Bertsekas. 1979. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper, MIT* (1979).

Carl W. Borchardt and Carl G.J. Jacobi. 1865. De investigando ordine systematis aequationum differentialium vulgarium cujuscunque. *Journal für die reine und angewandte Mathematik* 64 (1865), 297–320.

Mathias Bürger, Giuseppe Notarstefano, Francesco Bullo, and Frank Allgöwer. 2012. A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. *Automatica* (2012).

L. Busoniu, R. Babuska, and B. De Schutter. 2008. A Comprehensive Survey of Multiagent Reinforcement Learning. *Trans. Sys. Man Cyber Part C* 38, 2 (March 2008), 156–172. https://doi.org/10.1109/TSMCC.2007.913919

Panayiotis Danassis and Boi Faltings. 2019. Courtesy as a Means to Coordinate. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19)*. 9.

Panayiotis Danassis, Aris Filos-Ratsikas, and Boi Faltings. 2019a. Anytime Heuristic for Weighted Matching Through Altruism-Inspired Behavior. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. 215–222. https://doi.org/10.24963/ijcai.2019/31

Panayiotis Danassis, Marija Sakota, Aris Filos-Ratsikas, and Boi Faltings. 2019b. Putting Ridesharing to the Test: Efficient and Scalable Solutions and the Power of Dynamic Vehicle Relocation. *ArXiv: 1912.08066* (2019).

George B. Dantzig. 1990. *Origins of the simplex method*. ACM.

Jack Edmonds. 1965. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B* (1965).

Jack Edmonds and Richard M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* (1972). https://doi.org/10.1145/321694.321699

Michael Elkin. 2004. Distributed Approximation: A Survey. *SIGACT News* 35, 4 (Dec. 2004), 40–57. https://doi.org/10.1145/1054916.1054931

Yanfeng Geng and Christos G. Cassandras. 2013. New "smart parking" system based on resource allocation and reservations. *IEEE Transactions on Intelligent Transportation Systems* (2013).

Corrado Gini. 1912. Variabilità e mutabilità. *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi* (1912).

Stefano Giordani, Marin Lujak, and Francesco Martinelli. 2010. A distributed algorithm for the multi-robot task allocation problem. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer.

Tyler Gunn and John Anderson. 2013. Dynamic Heterogeneous Team Formation for Robotic Urban Search and Rescue. *Procedia Computer Science* (2013). The 4th Int. Conf. on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd Int. Conf. on Sustainable Energy Information Technology (SEIT-2013).

Sarah Ismail and Liang Sun. 2017. Decentralized hungarian-based approach for fast and scalable task allocation. In *2017 Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

Raj Jain, Dah-Ming Chiu, and W. Hawe. 1998. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *CoRR* cs.NI/9809099 (1998). http://arxiv.org/abs/cs.NI/9809099

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2016. Local Computation: Lower and Upper Bounds. *J. ACM* 63, 2, Article 17 (March 2016), 44 pages. https://doi.org/10.1145/2742012

Harold W. Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics* (1955).

László Lovász and Michael D. Plummer. 2009. *Matching theory*. Vol. 367. American Mathematical Soc.

James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* (1957).

François Ollivier. 2009. Looking for the order of a system of arbitrary ordinary differential equations. *Applicable Algebra in Engineering, Communication and Computing* (2009).

Christos H. Papadimitriou and Kenneth Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall.

Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *AAAI*.

Hsin-Hao Su. 2015. Algorithms for Fundamental Problems in Computer Networks. (2015).

Pradeep Varakantham, Shih-Fen Cheng, Geoff Gordon, and Asrar Ahmed. 2012. Decision support for agent populations in uncertain and congested environments. (2012).

Dominic Widdows, Jacob Lucas, Muchen Tang, and Weilun Wu. 2017. GrabShare: The construction of a realtime ridesharing service. In *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*.

Michael M. Zavlanos, Leonid Spesivtsev, and George J Pappas. 2008. A distributed auction algorithm for the assignment problem. In *Decision and Control, 2008*. IEEE.