Improving Multi-agent Coordination by Learning to Estimate Contention

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

Abstract

We present a multi-agent learning algorithm, ALMA-Learning, for efficient and fair allocations in *large-scale* systems. We circumvent 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, observes only own action/reward pairs, requires no inter-agent communication, and achieves near-optimal ($< 5\%$ loss) and fair coordination in a variety of synthetic scenarios and a real-world meeting scheduling problem. The lightweight nature and fast learning constitute ALMA-Learning ideal for *on-device* deployment.

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, were edge weights correspond to the utility of each alternative. Many multi-agent coordination problems can be formulated as such. Example applications include role allocation (e.g., team formation [Gunn and Anderson, 2013]), task assignment (e.g., smart factories, or taxi-passenger matching [Danassis *et al.*, 2019b; Varakantham *et al.*, 2012]), resource allocation (e.g., parking/charging spaces for autonomous vehicles [Geng and Cassandras, 2013]), etc. What follows is *applicable to any such scenario*, but for concreteness we focus on the *assignment problem* (bipartite matching), one of the most fundamental combinatorial optimization problems [Munkres, 1957].

A significant challenge for any algorithm for the assignment problem emerges from the nature of real-world applications, which is often *distributed* and *information-restrictive*. Sharing plans, utilities, or preferences creates high overhead, and 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.

A recently proposed heuristic (ALMA [Danassis *et al.*, 2019a]) was specifically designed to address the aforementioned challenges. ALMA is *decentralized*, completely uncoupled (agents are only aware of their *own history*), and requires *no communication* between the agents. Instead, 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. As a result, in the realistic case where each agent is interested in a *subset* (of fixed size) of the total resources, ALMA's convergence time is *constant* in the total problem size. This condition holds 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. This *lightweight* nature of ALMA coupled with the *lack of inter-agent communication*, and the *highly efficient allocations* [Danassis *et al.*, 2019b; Danassis *et al.*, 2019a; Danassis *et al.*, 2020], 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.).

Despite ALMA's high performance in a variety of domains, it remains a heuristic; i.e., sub-optimal by nature. In this work, we 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 chances that they will actually obtain the alternative option they consider when backing-off, which helps guide their search.

ALMA-Learning is applicable in *repeated allocation* games (e.g., self organization of intelligent infrastructure, autonomous mobility systems, etc.), but 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. A motivating *real-world* application is presented in Section 3.2, where ALMA-Learning is applied to solve a large-scale meeting scheduling problem.

1.1 Our Contributions

(1) We introduce **ALMA-Learning**, a distributed algorithm for large-scale multi-agent coordination, focusing on *scalability* and *on-device* deployment in real-world applications.

(2) We prove that ALMA-Learning *converges*.

(3) We provide a thorough evaluation in a variety of synthetic benchmarks and a real-world meeting scheduling problem. In all of them ALMA-Learning is able to quickly (as little as 64 training steps) reach allocations of high social welfare (less than 5% loss) and fairness (up to almost 10% lower inequality compared to the best performing baseline).

1.2 Discussion and Related Work

Multi-agent coordination can usually be formulated as a matching problem. Finding a maximum weight matching is one of the best-studied combinatorial optimization problems (see [Su, 2015; Lovász and Plummer, 2009]). There is a plethora of polynomial time algorithms, with the *Hungarian* algorithm [Kuhn, 1955] being the most prominent centralized one for the bipartite variant (i.e., the assignment problem). In real-world problems, 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. Decentralized algorithms (e.g., [Giordani *et al.*, 2010]) solve this problem, yet they 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. (see also [Kuhn *et al.*, 2016; Elkin, 2004] for general results in distributed approximability under only local information/computation).

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 in the realistic case where 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 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. On the other hand, under reasonable assumptions about the preferences of the agents, ALMA's runtime is *constant* in the total problem size, while requiring no message exchange (i.e., no communication network) between the participating agents. The proposed approach, ALMA-Learning, *preserves* 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 a variant of 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 in 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 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

2.1 The Assignment Problem

The assignment problem refers to finding a maximum weight matching in a weighted bipartite graph, $\mathcal{G} = \{\mathcal{N} \cup \mathcal{R}, \mathcal{V}\}$. In the studied scenario, $\mathcal{N} = \{1, \dots, N\}$ agents compete to acquire $\mathcal{R} = \{1, \dots, R\}$ resources. The weight of an edge $(n, r) \in \mathcal{V}$ 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., $\max_{\mathbf{x} \geq 0} \sum_{(n,r) \in \mathcal{V}} u_n(r) x_{n,r}$, where $\mathbf{x} = (x_{1,1}, \dots, x_{N,R})$, subject to $\sum_{r|(n,r) \in \mathcal{V}} x_{n,r} = 1, \forall n \in \mathcal{N}$, and $\sum_{n|(n,r) \in \mathcal{V}} x_{n,r} = 1, \forall r \in \mathcal{R}$.

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. Both ALMA and ALMA-Learning are run *independently and in parallel by all the agents* (to improve readability, we have omitted the subscript $_n$).

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 to inform collisions and detect free resources. It could be achieved by the use of *sensors*, or by a *single bit* (0 / 1) feedback from the resource (note that these messages would be between the requesting agent and the resource, not between the participating agents themselves).

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

(a)

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

(b)

Table 1: Adversarial examples: (1a) *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.
(1b) *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.

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

ALMA: ALtruistic MAtching Heuristic

ALMA converges to a resource through repeated trials. Let $\mathcal{A} = \{Y, A_{r_1}, \ldots, A_{r_{R^n}}\}$ denote the set of actions, where Y refers to yielding, and A_r refers to accessing resource r, and let g denote the agent's strategy. 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 (set $g \leftarrow Y$) with some probability. Otherwise, if $g = Y$, the agent chooses another resource r for monitoring. If the resource is free, he sets $g \leftarrow A_r$.

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 Alg. 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 use $P(loss) = f(loss)^\beta$, where β controls the aggressiveness (willingness to back-off), and

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

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, starting from the most preferred resource (see step 3 of Alg. 1).

Sources of Inefficiency. ALMA is a heuristic, i.e., sub-optimal by nature. It is worth understanding the sources of inefficiency, which in turn motivated ALMA-Learning. To do so, we provide a couple of adversarial examples.

In the original ALMA algorithm, all agents start attempting to claim their most preferred resource, and back-off with probability that depends on 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 1a. 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 (where agents n_1, n_2, n_3 are 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 in subsequent stage games, resulting in an optimal allocation.

In another example (Table 1b, 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 (for a social welfare 2, which is 28.5% worse than the optimal 2.8), thus, in expectation, resource r_1 has utility 0.5. ALMA-Learning solves this 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.

ALMA-Learning

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 the agent will incur by backing-off ($loss[]$). By learning these two values agents take more informed decisions, specifically: (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[1]:

(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 Alg. 2. The array is initialized to the utility of each resource (line 3 of Alg. 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 Alg. 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 Alg. 2.

[1]We have omitted the subscript n from all the variables and arrays, but every agent maintains their own estimates.

Algorithm 1 ALMA: Altruistic Matching Heuristic.

```
Require: Sort resources ($\mathcal{R}^n \subseteq \mathcal{R}$) in decreasing order of utility
    $r_0, \ldots, r_{R^n-1}$ 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 prob. $P(loss[r])$
10:             else
11:                 $converged \leftarrow True$
12:         else ($g = Y$)
13:             $current \leftarrow (current + 1)$ mod $R$
14:             Agent $n$ monitors $r \leftarrow r_{current}$.
15:             if Free($r$) then set $g \leftarrow A_r$
16:     return $r$, such that $g = A_r$
```

(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 Alg. 2). Subsequently, for every stage game, agent n starts by selecting resource r_{start}, and ends up winning resource r_{won}. The loss of r_{start} is then updated according to the following averaging process, where α is the learning rate:

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

Finally, the last condition (lines 17-18 of Alg. 2) ensures that agents who have acquired resources of high preference stop exploring, thus stabilizing the learning process.

2.3 Convergence

Convergence of ALMA-Learning does not translate to a fixed allocation at each stage game. 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 1. *There exists time-step t_{conv} such that $\forall t > t_{conv} : r^n_{start}(t) = r^n_{start}(t_{conv})$, where $r^n_{start}(t)$ denotes the starting resource r_{start} of agent n at the stage game of time-step t.*

Proof. (Sketch; see [Danassis *et al.*, 2021]) Theorem 2.1 of [Danassis *et al.*, 2019a] proves that ALMA (called at line 9 of Alg. 2) converges in polynomial time (in fact, under some assumptions, it converges in constant time, i.e., *each stage game converges in constant time*). In ALMA-Learning 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

Algorithm 2 ALMA-Learning

```
Require: Sort resources ($\mathcal{R}^n \subseteq \mathcal{R}$) in decreasing order of utility
    $r_0, \ldots, r_{R^n-1}$ 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:                 $(1-\alpha)loss[r_{start}] + \alpha\left(u(r_{start}) - u(r_{won})\right)$
16:
17:         if $r_{start} \neq r_{won}$ then
18:             $r_{start} \leftarrow \arg\max_r reward[r]$
```

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. $\square$

3 Evaluation

We evaluate ALMA-Learning in a variety of synthetic benchmarks and a meeting scheduling problem based on *real* data from [Romano and Nunamaker, 2001]. Error bars representing one standard deviation (SD) of uncertainty.

Fairness. 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 2. 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. To capture the fairness of this 'mixed' allocation, we report the average fairness on 32 evaluation time-steps that follow the training period.

To measure fairness, we used the *Gini coefficient* [Gini, 1912]. An allocation $\mathbf{x} = (x_1, \ldots, x_N)^\top$ is fair iff $\mathbb{G}(\mathbf{x}) = 0$, where: $\mathbb{G}(\mathbf{x}) = (\sum_{n=1}^N \sum_{n'=1}^N |x_n - x_{n'}|)/(2N \sum_{n=1}^N x_n)$

3.1 Test Case #1: Synthetic Benchmarks

Setting. We present results on three benchmarks:

(a) *Map*: Consider a Cartesian map on which the agents and resources are randomly distributed. The utility of agent n

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

Table 2: 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.

	Greedy	ALMA	**ALMA-Learning**
(a) Map	$1.51\% - 18.71\%$	$0.00\% - 9.57\%$	$0.00\% - 0.89\%$
(b) Noisy	$8.13\% - 12.86\%$	$2.96\% - 10.58\%$	$1.34\% - 2.26\%$
(c) Binary	$0.10\% - 14.70\%$	$0.00\% - 16.88\%$	$0.00\% - 0.39\%$

Table 3: Range of the average loss (%) in social welfare compared the the (centralized) optimal for the three different benchmarks.

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 $\sim \mathcal{N}(0, \sigma^2)$.

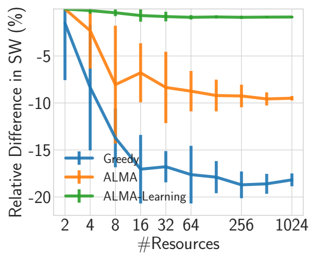
(c) *Binary Utilities*: This corresponds to each agent being indifferent to acquiring any resource amongst his set of desired resources, i.e., $u_n(r)$ is randomly assigned to 0 or 1.

Baselines. We compare against: (i) the *Hungarian algorithm* [Kuhn, 1955], which computes a maximum-weight matching in a bipartite graphs, (ii) *ALMA* [Danassis *et al.*, 2019a], and (iii) the *Greedy* algorithm, which goes through the agents randomly, and assigns them their most preferred, unassigned resource.

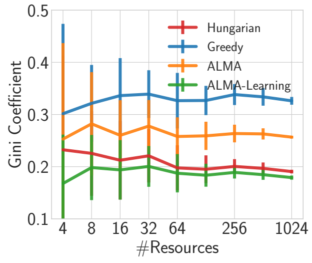
Results. We begin with the loss in social welfare. Figure 1a depicts the results for the Map test-case, while Table 3 aggregates all three test-cases[2]. ALMA-Leaning reaches *near-optimal* allocations (*less than* 2.5% *loss*), in most cases in just $32-512$ training time-steps. The exception is the Noisy Common Utilities test-case, where the training time was slightly higher. Intuitively we believe that this is because ALMA already starts with a near optimal allocation (especially for $R > 256$), and given the high similarity on the agent's utility tables (especially for $\sigma = 0.1$), it requires a lot of fine-tuning to improve the result.

Moving on to fairness, ALMA-Leaning achieves the *most fair allocations in all of the test-cases*. As an example, Figure 1b depicts the Gini coefficient for the Map test-case. ALMA-Learning's Gini coefficient is -18% to -90% lower on average (across problem sizes) than ALMA's, -24% to -93% lower than Greedy's, and -0.2% to -7% lower than Hungarian's.

[2]For the the Noisy Common Utilities test-case, we report results for $\sigma = 0.1$; which is the worst performing scenario for ALMA-Learning. Similar results were obtained for $\sigma = 0.2$ and $\sigma = 0.4$.



(a) Relative Difference in SW



(b) Gini Index (lower is better)

Figure 1: Map test-case. Results for increasing number of resources ($[2, 1024]$, x-axis in log scale), and $N = R$. ALMA-Learning was trained for 512 time-steps.

3.2 Test Case #2: Meeting Scheduling

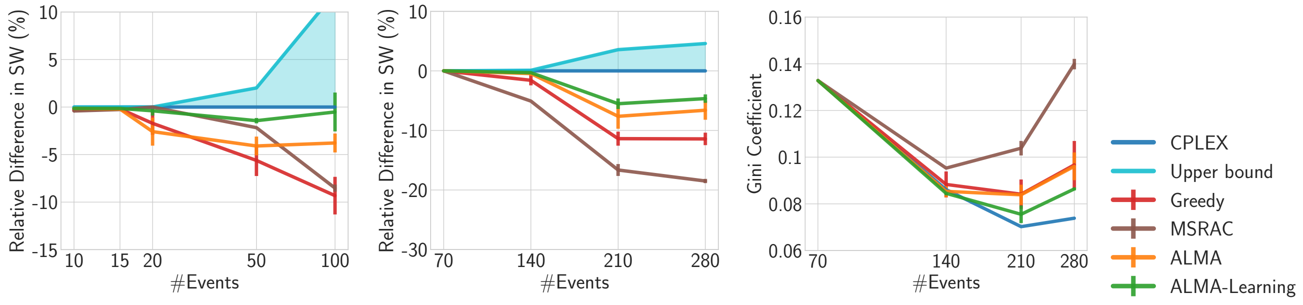
Motivation. The problem of scheduling a large number of meetings between multiple participants is ubiquitous in everyday life [Nigam and Srivastava, 2020; Ottens *et al.*, 2017; Zunino and Campo, 2009; Maheswaran *et al.*, 2004; BenHassine and Ho, 2007; Hassine *et al.*, 2004; Crawford and Veloso, 2005; Franzin *et al.*, 2002]. The advent of social media brought forth the need to schedule large-scale events, while the era of globalization and the shift to working-from-home require business meetings to account for participants with diverse preferences (e.g., different timezones).

Meeting scheduling is an inherently decentralized problem. Traditional approaches (e.g., distributed constraint optimization [Ottens *et al.*, 2017; Maheswaran *et al.*, 2004]) can only handle a bounded, small number of meetings. Interdependences between meetings' participants can drastically increase the complexity. While there are many commercially available electronic calendars (e.g., Doodle, Google calendar, Microsoft Outlook, Apple's Calendar, etc.), none of these products is capable of autonomously scheduling meetings, taking into consideration user preferences and availability.

While the problem is inherently online, meetings can be aggregated and scheduled in batches, similarly to the approach for tackling matchings in ridesharing platforms [Danassis *et al.*, 2019b]. In this test-case, we map meeting scheduling to an allocation problem and solve it using ALMA and ALMA-Learning. This showcases an application were ALMA-Learning can be used as a negotiation protocol.

Modeling. Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ denote the set of events and $\mathcal{P} = \{P_1, \ldots, P_m\}$ the set of participants. To formulate the participation, let $\text{part} : \mathcal{E} \to 2^{\mathcal{P}}$, where $2^{\mathcal{P}}$ denotes the power set of $\mathcal{P}$. We further define the variables $days$ and $slots$ to denote the number of days and time slots per day of our calendar (e.g., $days = 7, slots = 24$). In order to add length to each event, we define an additional function $\text{len} : \mathcal{E} \to \mathbb{N}$. Participants' utilities are given by:
$\text{pref} : \mathcal{E} \times \text{part}(\mathcal{E}) \times \{1, \ldots, days\} \times \{1, \ldots, slots\} \to [0, 1]$.

Mapping the above to the assignment problem of Section 2.1, we would have the set of $(day, slot)$ tuples to correspond to $\mathcal{R}$, while each event is represented by one event agent (that aggregates the participant preferences), and the set of which would correspond to $\mathcal{N}$.



(a) Relative Difference in SW (b) Relative Difference in SW (c) Gini Coefficient (lower is better)

Figure 2: Meeting Scheduling. Results for 100 participants ($\mathcal{P}$) and increasing number of events (x-axis in log scale). ALMA-Learning was trained for 512 time-steps.

Baselines. We compare against four baselines: (a) We used the IBM ILOG CP optimizer [Laborie *et al.*, 2018] to formulate and solve the problem as a CSP[3]. An additional benefit of this solver is that it provides an upper bound for the optimal solution (which is infeasible to compute). (b) A modified version of the MSRAC algorithm [BenHassine and Ho, 2007], and finally, (c) the greedy and (d) ALMA, as before.

Designing Large Test-Cases. As the problem size grows, CPLEX's estimate on the upper bound of the optimal solution becomes too loose (see Figure 2a). To get a more accurate estimate on the loss in social welfare for larger test-cases, we designed a large-instance by combining smaller problem instances, making it easier for CPLEX to solve which in turn allowed for tighter upper bounds as well (see Figure 2b).

We begin by solving two smaller problem instances with a low number of events. We then combine the two in a calendar of twice the length by duplicating the preferences, resulting in an instance of twice the number of events (agents) and calendar slots (resources). Specifically, in this case we generated seven one-day long sub-instances (with 10, 20, 30 and 40 events each), and combined then into a one-week long instance with 70, 140, 210 and 280 events, respectively. The fact that preferences repeat periodically, corresponds to participants being indifferent on the day (yet still have a preference on time).

These instances are depicted in Figure 2b and in the last line of Table 4.

Results. Figures 2a and 2b depict the relative difference in social welfare compared to CPLEX for 100 participants ($|\mathcal{P}| = 100$) and increasing number of events for the regular ($|\mathcal{E}| \in [10, 100]$) and larger test-cases ($|\mathcal{E}|$ up to 280), respectively. Table 4 aggregates the results for various values of $\mathcal{P}$. ALMA-Learning is able to achieve less than 5% loss compared to CPLEX, and this difference diminishes as the problem instance increases (less than 1.5% loss for $|\mathcal{P}| = 100$). Finally, for the largest hand-crafted instance ($|\mathcal{P}| = 100, |\mathcal{E}| = 280$, last line of Table 4 and Figure 2b), ALMA-Learning losses less than 9% compared to the *possible upper bound* of the optimal solution.

Moving on to fairness, Figure 2c depicts the Gini coefficient for the large, hand-crafted instances ($|\mathcal{P}| = 100, |\mathcal{E}|$ up to 280). ALMA-Learning exhibits low inequality, up to -9.5% lower than ALMA in certain cases. It is worth noting, though, that the fairness improvement is not as pronounced as in Section 3.1. In the meeting scheduling problem, all of the employed algorithms exhibit high fairness, due to the nature of the problem. Every participant has multiple meetings to schedule (contrary to only being matched to a single resource), all of which are drawn from the same distribution. Thus, as you increase the number of meetings to be scheduled, the fairness naturally improves.

	Greedy	MSRAC	ALMA	**ALMA-Learning**
$\lvert\mathcal{P}\rvert = 20$	$6.16\% - 18.35\%$	$0.00\% - 8.12\%$	$0.59\% - 8.69\%$	$0.16\% - 4.84\%$
$\lvert\mathcal{P}\rvert = 30$	$1.72\% - 14.92\%$	$1.47\% - 10.81\%$	$0.50\% - 8.40\%$	$0.47\% - 1.94\%$
$\lvert\mathcal{P}\rvert = 50$	$3.29\% - 12.52\%$	$0.00\% - 15.74\%$	$0.07\% - 7.34\%$	$0.05\% - 1.68\%$
$\lvert\mathcal{P}\rvert = 100$	$0.19\% - 9.32\%$	$0.00\% - 8.52\%$	$0.15\% - 4.10\%$	$0.14\% - 1.43\%$
$\lvert\mathcal{E}\rvert = 280$	$0.00\% - 15.31\%$	$0.00\% - 22.07\%$	$0.00\% - 10.81\%$	$0.00\% - 8.84\%$

Table 4: Range of the average loss ($\%$) in social welfare compared to the IBM ILOG CP optimizer for increasing number of participants, $\mathcal{P}$ ($|\mathcal{E}| \in [10, 100]$). The final line corresponds to the loss compared to the upper bound for the optimal solution for the large test-case with $|\mathcal{P}| = 100, |\mathcal{E}| = 280$ (Figure 2b).

Supplementary Material. We refer the reader to [Danassis *et al.*, 2021] for a detailed model of the meeting scheduling problem, implementation details and hyper-parameters, and additional results for both Sections 3.1, 3.2.

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 synthetic scenarios and a real-world meeting scheduling problem. ALMA-Learning is able to quickly (in as little as 64 training steps) reach allocations of high social welfare (less than 5% loss) and fairness.

[3]Computation time limit 20 minutes.

References

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

[Bellman, 2013] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[BenHassine and Ho, 2007] Ahlem BenHassine and Tu Bao Ho. An agent-based approach to solve dynamic meeting scheduling problems with preferences. *Engineering Applications of Artificial Intelligence*, 20(6):857–873, 2007.

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

[Busoniu *et al.*, 2008] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Trans. Sys. Man Cyber Part C*, 38(2):156–172, March 2008.

[Crawford and Veloso, 2005] Elisabeth Crawford and Manuela Veloso. Learning dynamic preferences in multi-agent meeting scheduling. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 487–490. IEEE, 2005.

[Danassis *et al.*, 2019a] Panayiotis Danassis, Aris Filos-Ratsikas, and Boi Faltings. Anytime heuristic for weighted matching through altruism-inspired behavior. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 215–222, 2019.

[Danassis *et al.*, 2019b] Panayiotis Danassis, Marija Sakota, Aris Filos-Ratsikas, and Boi Faltings. Putting ridesharing to the test: Efficient and scalable solutions and the power of dynamic vehicle relocation. *ArXiv: 1912.08066*, 2019.

[Danassis *et al.*, 2020] Panayiotis Danassis, Aleksei Triastcyn, and Boi Faltings. Differential privacy meets maximum-weight matching, 2020.

[Danassis *et al.*, 2021] Panayiotis Danassis, Florian Wiedemair, and Boi Faltings. Improving multi-agent coordination by learning to estimate contention. *http://arxiv.org/abs/2105.04027*, 2021.

[Elkin, 2004] Michael Elkin. Distributed approximation: A survey. *SIGACT News*, 35(4):40–57, December 2004.

[Franzin *et al.*, 2002] Maria Sole Franzin, EC Freuder, F Rossi, and R Wallace. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In *Proceedings of the AAAI Workshop on Preference in AI and CP*, 2002.

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

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

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

[Gunn and Anderson, 2013] Tyler Gunn and John Anderson. 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).

[Hassine *et al.*, 2004] Ahlem Ben Hassine, Xavier Defago, and Tu Bao Ho. Agent-based approach to dynamic meeting scheduling problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, 2004.

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

[Kuhn *et al.*, 2016] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2):17:1–17:44, March 2016.

[Kuhn, 1955] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*, 1955.

[Laborie *et al.*, 2018] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.

[Lovász and Plummer, 2009] László Lovász and Michael D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[Maheswaran *et al.*, 2004] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, 2004.

[Munkres, 1957] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 1957.

[Nigam and Srivastava, 2020] Archana Nigam and Sanjay Srivastava. Oddms: Online distributed dynamic meeting scheduler. In Simon Fong, Nilanjan Dey, and Amit Joshi, editors, *ICT Analysis and Applications*, Singapore, 2020. Springer Singapore.

[Ottens *et al.*, 2017] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. Duct: An upper confidence bound approach to distributed constraint optimization problems. *ACM Trans. Intell. Syst. Technol.*, 8(5), July 2017.

[Romano and Nunamaker, 2001] Nicholas C Romano and Jay F Nunamaker. Meeting analysis: Findings from research and practice. In *Proceedings of the 34th annual Hawaii international conference on system sciences*, pages 13–pp. IEEE, 2001.

[Stone *et al.*, 2010] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*, 2010.

[Su, 2015] Hsin-Hao Su. *Algorithms for Fundamental Problems in Computer Networks.* PhD thesis, University of Michigan, 2015.

[Varakantham *et al.*, 2012] Pradeep Varakantham, Shih-Fen Cheng, Geoff Gordon, and Asrar Ahmed. Decision support for agent populations in uncertain and congested environments. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12. AAAI Press, 2012.

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

[Zunino and Campo, 2009] Alejandro Zunino and Marcelo Campo. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 2009.