Asynchronous Communication Aware Multi-Agent Task Allocation

Ben Rachmut[1] , **Sofia Amador Nelke**[2] , **Roie Zivan**[1]
[1]Ben Gurion University of the Negev
[2]Holon Institute of Technology
rachmut@post.bgu.ac.il,sofiaa@hit.ac.il,zivanr@bgu.ac.il

Abstract

Multi-agent task allocation in physical environments with spatial and temporal constraints, are hard problems that are relevant in many realistic applications. A task allocation algorithm based on Fisher market clearing (FMC_TA), that can be performed either centrally or distributively, has been shown to produce high quality allocations in comparison to both centralized and distributed state of the art incomplete optimization algorithms. However, the algorithm is synchronous and therefore depends on perfect communication between agents. We propose FMC_ATA, an asynchronous version of FMC_TA, which is robust to message latency and message loss. In contrast to the former version of the algorithm, FMC_ATA allows agents to identify dynamic events and initiate the generation of an updated allocation. Thus, it is more compatible for dynamic environments. We further investigate the conditions in which the distributed version of the algorithm is preferred over the centralized version. Our results indicate that the proposed asynchronous distributed algorithm produces consistent results even when the communication level is extremely poor.

1 Introduction

Task allocation is a major challenge in realistic scenarios, e.g., disaster response, where medical personnel, fire fighters, police, and mechanical entities (e.g., drones and unmanned ground vehicles) need to coordinate their actions in order to save as many victims as possible [Nunes *et al.*, 2017; Tadokoro *et al.*, 2000; Jones *et al.*, 2007]. Such coordination is extremely challenging since in such scenarios the communication among agents is expected to be be severely degraded and unreliable [Macarthur *et al.*, 2011; Carrillo *et al.*, 2021; Otte *et al.*, 2020].

Moreover, such scenarios are highly dynamic due to the appearance of new events or the change of the status of handled events [Wei *et al.*, 2016]. Identification of dynamic events, would most likely be by the agents performing in the environment. Thus, we expect the agents to be able to reinitialize the solving process when necessary [Farinelli *et al.*, 2017; Fioretto *et al.*, 2018].

Fisher Market Clearing Task Allocation (FMC_TA) [Nelke and Zivan, 2017] is an algorithm that was proposed for solving problems where a team of heterogeneous agents need to cooperate in an environment that includes multiple tasks, which require ad-hoc coalitions of agents with different skills in order to properly handle them. The algorithm is composed of two phases. In the first, the problem is reduced to a Fisher market, having task performing agents as buyers in the market and tasks as goods. Then, the corresponding Fisher market clearing allocation is found. When the allocation resulted in tasks that are shared among a number of agents, an ad-hoc coalition was generated, which includes the agents that received a share of the task. In the second phase, a distributed ordering heuristic is performed for agents to decide on the schedule for performing tasks, and to coordinate mutual task performance by coalitions that share tasks. The Fisher market clearing outcome is guaranteed to be envy free and Pareto optimal [Devanur *et al.*, 2002; Reijnierse and Potters, 1998]. This unique combination results in an allocation where agents share important and complex tasks efficiently. FMC_TA was shown to dominate state of the art centralized and distributed task allocation algorithms. These included general optimization algorithms such as Simulated annealing and designated algorithms such as Coalition formation with look ahead (CFLA) [Farinelli *et al.*, 2008; Ramchurn *et al.*, 2010; Nelke *et al.*, 2020].

However, FMC_TA as proposed in [Nelke and Zivan, 2017] is a synchronous algorithm in which in each iteration of the algorithm, agents perform calculations only after they received all messages that they expect to be sent to them by their neighbors [Zhang *et al.*, 2005; Maheswaran *et al.*, 2004; Zivan *et al.*, 2014]. Unfortunately, such a synchronous algorithmic design incurs a number of drawbacks. Each synchronous operation is performed only after all messages sent in the previous iteration arrive, i.e., if a message is delayed, the iteration starts late and if a message is lost, the agents are in a deadlock. Moreover, the algorithm presented in [Nelke and Zivan, 2017] was dependent on perfect knowledge that the agents hold regarding the existence and importance of tasks to be performed. Thus, the team of agents was not independent and relied on updates from a centralized system.

In order to overcome these limitations of FMC_TA when facing dynamic scenarios and imperfect communication, we propose FMC_ATA, an asynchronous version of the algorithm that was designed while taking into consideration the possibility that messages can be delayed or lost (i.e., communication aware). Moreover, in FMC_ATA agents can detect dynamic events such as new tasks that need to be performed or a change in the importance of a task that is currently being handled, and trigger execution of the algorithm, for the team to adapt to the evolved problem.

We investigated the properties of scenarios in which a distributed implementation of FMC_ATA is preferred over a centralized implementation in which a central system is updated by the agents regarding dynamic events, calculates and updated allocation and updates the agents. Our results show that in the presence of message latency, a clear threshold exists for distributed performance to be motivated, and when message loss cannot be avoided, distributed performance is always preferred.

In this work we contribute to the applicability of multi agents task allocation algorithms to realistic problems by: **1)** Presenting the single phase version of the Fisher Market Clearing Asynchronous Task Allocation algorithm FMC_ATA. **2)** Demonstrating empirically that, not only does the asynchronous version (FMC_ATA) converge to the same solution as FMC_TA, but that it is also robust to message delays and to message loss up to some extent. **3)** Investigating the communication conditions that make it more beneficial to perform the algorithm distributively (and not centrally).

Our approach in investigating the performance of communication aware task allocation follows recent publications on the performance of distributed local search algorithms and distributed incomplete inference algorithms in the presence of imperfect communication [Rachmut *et al.*, 2022; Zivan *et al.*, 2021]. Our results demonstrate that FMC_ATA is a highly robust algorithm, that is compatible for dynamic environments with imperfect communication.

2 Preliminaries

In this section we formalize the General Task Allocation Problem (GTAP), which is an abstraction of LEP, the law enforcement problem (LEP is a specific instance of GTAP) that was presented in [Nelke and Zivan, 2017]. We present both static and dynamic versions, and specify how the formalization represents communication limitations. In addition we present the FMC_TA algorithm that was proposed for solving such problems in [Nelke and Zivan, 2017].

2.1 General Task Allocation Problem (GTAP)

A GTAP includes a set of cooperative agents $A = \{a_1, a_2, ..., a_n\}$, a set of tasks $V = \{v_1, v_2, ..., v_m\}$ and a set of unique skills $S = \{s_1, s_2, ..., s_l\}$. Each agent a_i has a subset $S_i \in S$ of skills and each task v_j has a subset $S_j \in S$ of required skills. We denote by v_j^k the sub-task of v_j for which there is a specified skill s_k. For each sub-task v_j^k there is a required workload w_j^k which specifies the workload of a specific skill $s_k \in S_j$ that should be applied to v_j in the combined effort for completing this task. Thus, v_j is completed only if for each of its sub-tasks, the proper amount of work has been performed by agents that posses the required skills. Each task and each agent has a physical location. The set of all possible locations is denoted by L. We denote by $\rho : L \times L \rightarrow [0, +\infty)$, the time required to travel between two locations.An allocation of tasks to agents is denoted by a $n \times m \times k$ matrix X where entry x_{ijk} is the fraction of sub-task v_j^k that is assigned to agent a_i.

Agents can only perform a single sub-task at a time and utilize only one skill. Let M_i be the number of sub-tasks, agent a_i is allocated to. The schedule of a_i is denoted by σ^i and it contains a sequence of M_i triplets (v_j^k, st, ft) specifying the skill that is being performed on a specific sub task, and the start and end time for applying that skill by the agent, respectively. Multiple agents can share a single sub-task. The amount of time an agent must spend on the task is equal to its allocated fraction of the workload $ft - st = x_{ijk} w_j^k$.

The utility that agents derive from applying a skill to some task depends on the number of agents $q \in \mathbb{N}$ that handle it simultaneously. It is denoted by the non-negative capability function, $Cap(v_j^k, q)$. Let $d_q^{v_j^k}$ be the time that q agents are working together on v_j^k in some solution (schedule), ξ, for the problem. Thus $\frac{q d_q^{v_j^k}}{w_j^k}$ is the relative part of the mission that is performed by q agents simultaneously. The initial utility (inu_j^k) that can be derived by the agents for completing the performance of v_j^k in ξ is: $inu_j^k(\xi) = \sum_{q=1}^{n_j^k} \frac{q d_q^{v_j^k}}{w_j^k} Cap(v_j^k, q)$, where n_j^k is the limited amount of agents required for handling v_j^k. The total initial utility for performing all skills of v_j in ξ is $inu_j(\xi) = \sum_{s_k \in S_j} inu_j^k(\xi)$.

The utility derived for completing the performance of task v_j depends also on the soft deadline function $\delta(v_j, t) : V \times [0, +\infty) \rightarrow (0, 1]$, which is monotonically non-increasing in t. Thus, the discounted utility (du_j) for task v_j, which is initially handled at time t_{v_j}, is: $du_j(\xi) = \delta(v_j, t_{v_j}) inu_j(\xi)$.

2.2 Dynamic GTAP

The dynamic problem is represented as a sequence of static problems, each instantiated when a new task arrives. In the dynamic problem, tasks arise over time. We denote by $\alpha(v_j)$ the time at which task v_j is commenced. Thus, the discounted utility for performing a task is dependent on the task's start time: $du_j(\xi) = \delta(v_j, t_{v_j} - \alpha(v_j)) inu_j(\xi)$. The current task (if any) that is being performed by agent a_i and the current skill that it is using, are denoted by ct_i and cs_i, respectively. Agents can interrupt the performance of their current task. The penalty for task interruption, $\pi(ct_i, \Delta w_{ct_i}^{cs_i})$ (The full description of the $\pi(v_j, \Delta w_{ct_i}^{cs_i})$ is shown in [Nelke and Zivan, 2017]), depends on the interrupted task ct_i and the amount of workload left for skill cs_i while abandoning, $\Delta w_{ct_i}^{cs_i}$. The total utility derived for v_j is thus: $U_{v_j}(\xi) = du_j(\xi) - \sum_{a_i : ct_i = v_j \wedge v_{s_1}^i \neq ct_i} \pi(ct_i, \Delta w_{ct_i}^{cs_i})$. Where $v_{s_1}^i$ is the first task in agent's schedule. The total team utility for solution ξ is: $SW(\xi) = \sum_{v_j \in V} U_{v_j}(\xi)$. GTAP can be solved centrally or distributively. In a centralized implementation, all information is transferred to a central unit that solves the problem and propagates the allocation it produces to the agents. In a distributed implementation, agents solve the problem using a message passing algorithm.

2.3 Communication Aware GTAP

Previous work on GTAP [Nelke and Zivan, 2017] assumed that instances of the problem are multi-agent scenarios with perfect communication among agents, enabling the use of synchronous algorithms. However, in many realistic task allocation applications like disaster response, perfect communication is unrealistic, making it necessary for the design of algorithms to be communication-aware, i.e., asynchronous algorithms in which the possibility that messages can be delayed or lost are considered.

In order to allow the design of such communication aware algorithms, we extend GTAP to represent communication disturbances, by using a Constrained Communication Graph (CCG), that represents the possibly dynamically changing uncertainty in the communication between agents [Rachmut *et al.*, 2022]. To represent the communication limitations, in a CCG, every link of communication between agents (the vertexes) is represented by an edge and the constraint on each edge defines the latency and probability of a message loss. Formally, for every edge e_{ij} in the CCG, $td_{e_{ij}}$ is the function that represents the delay on e_{ij} (i.e., it calculates the time between when a message is sent and when it is received via edge e_{ij}). In addition, $pl_{e_{ij}} \in [0, 1)$ is the probability that a message sent through the communication link represented by e_{ij} is lost.

2.4 Fisher Market Clearing Task Allocation (FMC_TA)

As described in [Nelke and Zivan, 2017], FMC_TA is an algorithm for heterogeneous task allocation that can be implemented both in a centralized and distributed manner. The distributed implementation is by design synchronous. The algorithm is composed of two phases. The first generates the allocation and defines the cooperation among agents. The second determines the schedule according to which agents will perform the sub-tasks allocated to them.

The first phase manipulates a Fisher market clearing (FMC) algorithm in order to generate the allocation and ad-hoc coalitions that will perform tasks. This is done using a matrix a R that is a $3-dimensional$ matrix of size $n \times m \times l$ is generated. Each entry r_{ijk} in R represents the personal utility that agent a_i will derive if it will perform sub-task v_j^k, assuming it will start moving towards it and perform the sub-task when it arrives. If agent a_i does not possess skill s_k or task v_j does not require it, the value of entry r_{ijk} will be zero. The utility is constructed ignoring the inter-task ordering constraints. It does take into consideration the maximum utility that agents can derive by performing the sub-task (according to the capability function), a penalty for late execution and for not completing the current task the agent is performing. Formally: $r_{ijk} = \delta(v_j, \rho(a_i, v_j))Cap(v_j^k, n_j^k) - \pi(ct_i, \Delta w_{ct_i}^{cs_i})$, where the penalty is omitted if $ct_i = v_j^k$.

The Fisher market-clearing allocation can be found distributively or centrally, using the proportional response algorithm [Zhang, 2011]. In the distributed version, for each task there is an agent representing it. To differentiate between the active agents that perform tasks and the agents representing tasks, we will denote an active agent i by aa_i and a task agent representing task v_j by ta_j. Active agents iteratively submit bids to the task agents regarding specific sub-tasks and are in turn awarded provisional allocations, which they use to modify their bids in the next round. Formally, the bid of aa_i on sub-task v_j^k at time t is denoted by $b_{ijk}(t)$. The allocation that is calculated by the task agent ta_j is $x_{ijk} = \frac{b_{ijk}(t)}{\sum_i b_{ijk}(t)}$ and the next bid is $b_{ijk}(t+1) = \frac{u_{ijk}(t)}{u_i(t)} b_{ijk}(t)$ where $u_{ijk}(t) = (x_{ijk} r_{ijk})$ and $u_i(t) = \sum_j \sum_s u_{ijk}(t)$. The algorithm converges to a Fisher market equilibrium in pseudo-polynomial time [Zhang, 2011].

In the centralized version, this algorithm can be simulated by a single machine. There are other algorithms for computing the Fisher market clearing allocation in a centralized manner, such as the one proposed in [Dean and Nair, 2014]. In the second phase of FMC_TA, the sub-tasks allocated to each of the active agents are scheduled to reflect the spatial and temporal inter-task and inter-agent constraints. Each aa_i orders the sub-tasks allocated to it greedily prioritizing them according to the ratio between the utility derived from performing them and the required workload (i.e, Bang per Buck). Using the order of sub-tasks, the active agent aa_i calculates the estimated arrival time for each allocated sub-task (v_j^k), denoted by $t^i_{v_j^k}$. This sequence arrival times constitutes its initial schedule. Then, aa_i sends $t^i_{v_j^k}$ to ta_j (for all sub-tasks in σ^i). The task agent ta_j receives, updates and maintains the start times ($t_{v_j^k}$). If v_j^k is a shared sub-task, ta_j then communicates $t_{v_j^k}$ to all active agents that share v_j^k. Upon receiving a $t_{v_j^k}$ message, aa_i checks if its individual schedule can be improved by advancing sub-tasks that are not shared, without delaying the execution of shared sub-tasks. The result of this phase is that each active agent holds a schedule of the sub-tasks allocated to it, in the order that it will perform them.

3 Fisher Market Clearing Asynchronous Task Allocation (FMC_ATA)

The asynchronous version of FMC_TA (FMC_ATA) was designed to be (and evidently is) robust both to message latency and message loss. The three major aspects that differentiate it from FMC_TA are: **1)** In FMC_ATA, agents do not wait for messages to arrive from all their neighbors in order for a computation step to begin. On the contrary, each message received triggers such a computation step. **2)** In FMC_ATA the two phases of FMC_TA are merged into a single step and performed simultaneously. **3)** In FMC_TA (both in the centralized and in the distributed version), it is assumed that there is a central entity that is aware of the location and importance of all tasks present in the scenario and that it propagates this information to the agents. In FMC_ATA, agents dynamically discover tasks and propagate the relevant information to their peers.

FMC_ATA is a distributed asynchronous algorithm. As explained for FMC_TA, in FMC_ATA there are two types of

Algorithm 1 FMC_ATA code of Task Agent ta_j

```
1: $ci \leftarrow false$
2: while not $ci$:
3:   when message $m$ received:
4:     $b_{ik} \leftarrow m.b_{ik}$
5:     $p_k \leftarrow calculate\_price(b_k)$
6:     $X_k \leftarrow calculate\_allocation(p_k, b_k)$
7:     $T^k_j \leftarrow m.t^i_{v^k_j}$
8:     $t^1_{v^k_j} \leftarrow max(T^k_j)$
9:   $ci \leftarrow is\_price\_converged(\vec{p}, \epsilon)$
10:    send messages to all $a_i \in A^j$
```

entities: active agents and task agents. In practice, the role of a task agent is performed by one of the active agents, e.g., the first to identify the task. The communication graph is bipartite, i.e., the neighbors of each active agent are only task agents and vice versa.

Every task agent ta_j representing task v_j, has a local view $\langle A^j, X^j, B^j, \vec{p}, T^j\rangle$ where $A^j = \{a_i \in A | S_i \cap S_j \neq \emptyset\}$ is a set of active agents that can apply at least one skill as part of the performance of v_j and derive positive utility (i.e., neighboring active agents), $X^j_{|A^j|\times|S_j|}$ and $B^j_{|A^j|\times|S_j|}$ are matrices of allocations and bids of neighboring agents to the required skills of the task v_j. The $\vec{p}$ is a vector of prices where $|\vec{p}| = |S_j|$ and $T^j_{|A^j|\times|S_j|}$ is the matrix of the earliest times that the agents in A^j are able to perform the sub-tasks. The price for each sub-task is the sum of the latest bids for this task that arrived from each of the active agent neighbors. Each message sent from aa_i to the ta_j includes $\langle aa_i, ta_j, s_k, bid_{ik}, t_{v^k_j}\rangle$, where aa_i and ta_j indicate the sender active agent and the receiving task agent, bid_{ik} is the bid of aa_i for utilizing skill s_k and $t_{v^k_j}$ is the earliest time that aa_i can perform the sub-task.

Algorithm 1 presents the actions of task agent ta_j when handling incoming messages. First, for each new message, matrix B is updated with the new bid b_{ik} (line 4). Next, the prices for all skills are calculated by $p_k = \sum_{i\in A} b_{ik}$ in $\vec{p}$ (line 5); and new allocations in X_k are determined by $x_{ik} = \frac{b_{ik}}{p_k}$ (line 6). Lines 7 and 8 refer to the scheduling process, which is integrated into a single iterative computation (in contrast to FMC_TA, where the scheduling was performed in a separate step). The time $t_{v^k_j}$ is updated in T^j and the earliest time that allocated agents to sub-task can perform concurrently, $t^1_{v^k_j}$, is updated with the max time in the row T^j_k. The task agent's algorithm runs until the prices for all skills converge, $\forall p_k \in \vec{p} | \Delta p_k < \epsilon$ (in our experiments we used $\epsilon = 10^{-5}$). Once they converge, a convergence indicator ci is updated (line 9). Finally, messages are sent to each $aa_i \in A^j$ with the up-to-dated allocations $x_{ik} \in X$, the earliest sharing time $t^1_{v^k_j}$ and the convergence indicator ci (line 10).

Every active agent aa_i also has a local view $\langle V^i, R^i, X^i, \vec{i}, T^1 \sigma^i\rangle$ where V^i is a set of neighboring task agents that aa_i is aware of. $R^i_{|V|\times|S_i|}$ is the reward matrix in which every entry $r_{v^k_j}$ specifies the expected utility derived by aa_i for applying a skill s_k for performing task v_j, and $X^i_{|V|\times|S_i|}$ is the allocation matrix that specifies the portion of workload that aa_i is allocated for each such combination of skill and task. $\vec{ci}$ indicates whether each neighboring task agent has converged where $\left|\vec{ci}\right| = \left|V^i\right|$; and $T^1_{|V|\times|S_i|}$ consists of the earliest shared execution times. σ^i is the current schedule of agent a_i. There are two types of messages that active agent aa_i can receive: a "handshake" message HSM and a standard message SM. The "handshake" message contains initial information regarding new task v_j discovered by one of the agents. This type of message allows tasks to be asynchronously detected by the active agents. The standard message includes the dynamic changing information throughout the execution of the algorithm (as described in Algorithm 1 line10)

Algorithm 2 presents the active agent aa_i actions when receiving messages. First, the agent distinguishes between the two types of messages, and reacts accordingly. If the message type is a HSM, the new task v_j is added to the set V^i and the personal utility r_{jk} is calculated for each of the skills that aa_i has and v_j requires (as described in Section 2.4)(line 5-6). If the message type is a standard message (i.e., the task already exists in V^i): X^i is updated with x_{ik}, convergence vector indicator $\vec{ci}$ is updated with ci_j, and the earlier shared execution time $t^1_{v^k_j}$ is updated in T^1 (lines 8). After updating its local view, the agent proceeds to re-calculate its bids as follows: $b_{jk} = \frac{r_{jk} * x_{jk}}{\sum_{j',k'} r_{j'k'} * x_{j'k'}}$. For new tasks, for which there is not yet an allocation in the agent's local view, we assume $x_{jk} = 1$ (lines 9). The following steps of the active agent's algorithm are equivalent to the second phase in FMC_TA. The active agent calculates its own schedule only for old tasks. The initial schedule is determined by sorting all allocations to tasks in X^i (the tasks that were allocated to active agent aa_i) according to their Bang per Buck i.e., $\frac{r_{ijk} x_{ijk}}{\sum_{x_{ijz} \in x_i} r_{ijz} x_{ijz}}$ (line 10).

According to initial schedule the active agent's arrival time for each of the tasks allocated to it is calculated. The calculation of the arrival time to the first task considers only the travel time. The calculation of the arrival times to the rest of the tasks allocated to it takes into consideration in addition to the travel time, the time that the active agent spend performing previous tasks (line 11). Then, for every sub-task v^k_j in σ^i it checks if the time $t^1_{v^k_j} \in T^1$ is larger than the time that was scheduled for the allocation, it tries to promote in the queue tasks where sharing is not required; and the final time $t^i_{v^k_j}$ of the arrival to task is determined (lines 12). Finally, each aa_i sends messages to all neighboring task agents in V^i, with the updated bids, b_{jk}, and its arrival time $t^i_{v^k_j}$, as it appears in the up-to-date schedule σ^i(line 13).

4 Experimental Design

To investigate the performance of FMC_ATA, we used a distributed asynchronous simulator, in which agents were im-

Algorithm 2 FMC_ATA code of Active Agent aa_i

```
1: $V^i \leftarrow \emptyset$
2: while not $tasks\_converged(\vec{i})$:
3:   when message received:
4:     if $HSM$ received:
5:       add task $v_j$ to set $V^i$
6:       $r_{jk} \leftarrow calculate\_r(v_j, s_k)$
7:     else:
8:       $update\_local\_view(m.x_{jk}, m.ic_j, m.t^1_{v^k_j})$
9:     $B^i \leftarrow calculate\_bids(R^i, X^i)$
10:    $\sigma^i \leftarrow create\_initial\_schedule(X^i, V^i)$
11:    $calculate\_start\_and\_end\_times(\sigma^i)$
12:    $check\_if\_tasks\_can\_promoted(\sigma^i)$
13:    send messages to all $v_j \in V^i$
```

plemented as Python threads. The simulated environment was dynamic and included random incoming tasks. In addition, the simulator allows to examine scenarios with imperfect communication by enabling any pattern of message delays and any probability for message loss. Imperfect communication was simulated according to the method suggested in [Zivan and Meisels, 2006] and was used in recent work for communication aware Distributed Constraint Optimization Problems (DCOP) [Rachmut *et al.*, 2021]. All messages sent by agents in the simulator are passed to an abstract *mailing agent*, which dictates when messages are delivered to their destination, according to the selected latency pattern or probability of message loss. The delay is selected in terms of the number of *Non-Concurrent Logic Operations* (NCLO), an independent measure for evaluating the performance of algorithms performing in asynchronous distributed settings. NCLO is an independent metric for evaluating the runtime in simulations of distributed asynchronous algorithms performed on thread-based simulators [Zivan and Meisels, 2006; Netzer *et al.*, 2012]. The simulator's code is public and available[1].

In each experiment, we randomly generated 50 different problem instances. The results presented in the graphs are an average of those 50 runs. Each scenario in an experiment included two types of agents, active agents and task agents, and for each of them, a random geographic location (coordinates x and y) was selected uniformly between 0 and 10^6. Each problem instance included active agents with a set of three unique skills. An active agent was characterized by its set of skills. The skills for each active agent were selected randomly with probability 0.5 for possessing each skill. If the result of the process was that the agent did not possess any skill, one was selected randomly and assigned to it. All agents the same consistent speed, a single unit of distance per NCLO. Each task requires three skills. The value of n^k_j (i.e., maximum number of agents required for handling v^k_j) was consistent and set to 5. For each skill in a task, the value of $Cap(v^k_j, n^k_j)$ was selected from a uniform distribution as follows: $Cap(v^k_j, n^k_j) \sim U(0, 10^5)$. For $q < n^k_j$ the value was relative such that $Cap(v^k_j, q) = Cap(v^k_j, n^k_j)\frac{q}{n^k_j}$. The workload required (w^k_j) was also selected from a uniform distribution and was affected by $Cap(v^k_j, n^k_j)$ such that $w^k_j \sim U(10^5, 10^5 + Cap(v^k_j, n^k_j))$.

Throughout the experiments, we investigated a variety of imperfect communication scenarios, using a CCG graph as described in section 2.3. We focused on two types of communication degradation – messages that are lost and messages that are delayed. The magnitude of latency and the probability of a message loss were both correlated to the distance between the location of the agents exchanging information.

We performed two sets of experiments: static and dynamic. The static experiments demonstrated the convergence process of the algorithms in the presences of a imperfect communication. Each static problem instance, included a set of 25 random tasks that were present from the start to the end of the experiment. We examined how the algorithms scale by evaluating their performance on problems with different amounts of active agents (e.g., 20, 40 and 60), we present the average team utility as a result of the schedule that would have been produced by each algorithm every 1000 NCLOs. We considered two versions of the proposed FMC_ATA algorithm in our experiments. The first, FMC_ATA_task_aware, is informed by a central unit of all tasks included in the problem upon their arrival. The second, FMC_ATA, is not dependent on a centralized entity. Agents discover tasks and inform one another using *handshake* messages. We compared those algorithms with the performance of the synchronous FMC_TA as suggested by [Nelke and Zivan, 2017][2].

To investigate the resilience of the distributed asynchronous version of FMC_ATA to dynamic events in the presence of imperfect communication, in comparison with the centralized implementation of FMC_TA [Nelke and Zivan, 2017], we generated different instances of the Dynamic GTAP (as described in subsection 2.2). Each problem instance in these experiments included 60 active agents. 10 random tasks were initiated prior to the start of the algorithm's performance and another 15 were discovered throughout the execution of the simulation. The time between the addition of new arriving tasks (*tbt*) was randomly selected from an exponential distribution, i.e., $tbt \sim exp(\beta)$ where the parameter β controls the mean time between tasks. We used $\beta = 10^5$ in our experiments.We consider the execution time of the algorithms and its effect on the team utility measurements.

5 Experimental Evaluation

Figures 1 and 2 present the team utility of the allocations generated by the different algorithms, as a function of NCLOs, when solving static problems. In Figure 1, we present the results of the algorithms when solving scenarios that include message latency, taken from a uniform distribution ($td_{e_{ij}} \sim U(0, UB^{d_{ij}})$). Each sub-graph presents a different magnitude of latency, i.e., different values for UB. The type of lines (e.g., dashed or not) indicate the amount of active agents. The curves correspond to different versions of

[1]https://github.com/benrachmut/Simulation_For_Research

[2]FMC_TA agents are informed by a central unit of all tasks included in the problem upon their arrival

the algorithm. The black line is the average performance of FMC_TA team utility with perfect communication (PC) and was used a benchmark. It is clear from the results that the algorithms converge to solutions with similar solution quality regardless to the latency magnitude. Our results demonstrate that FMC_ATA converges to solutions with similar quality to the solutions FMC_TA converges to. Our analysis indicates that there is no statistical significance between the algorithms.

Nevertheless, the algorithms differ in their convergence rate. The convergence rate of FMC_ATA_task_aware, in which all tasks are known from the beginning of the run (green curve), is faster in comparison to FMC_ATA, in which agents are required to discover tasks (blue curve). Furthermore, FMC_ATA's convergence rate increases when more active agents are present. That can be explained by the decrease in the agents' idle time. Message exchange occurs more frequently and agents perform more computations. In comparison to FMC_TA (red curve), FMC_ATA has an improved convergence rate in scenarios with 60 active agents. This happens despite the fact that FMC_TA agents are informed of active tasks by a central entity. This demonstrates the vulnerability of the synchronous algorithm to message latency.

In figure 2, we examine scenarios with message loss. Each sub-graph represents a different probability function, where the first two sub-graphs are dependent of the distance between the entities (i.e., $P = e^{-\psi d_{ij}}$) where $\psi = 1, 2$ respectively. The last graph examines a scenario where the probability is constant (i.e., independent of the distance between the entities) and is set on 0.1 such that only 10% of the messages are expected to arrive to their destination. Notice that FMC_TA was not included in the experiments where messages are lost since it deadlocks in such scenarios. Again, even in radical conditions, where messages are loss, the solution quality of FMC_ATA remains consistent. In regards to the effect of amount of active agents present, we observe that as the number of agents increases, the convergences rate decreases. Since agents in the asynchronous algorithm compute after each message they receive, message loss does not affect the agents idle time (with the exception of extreme scenarios where communication is cut off completely). Therefore, the convergence rate is simply affected by the complexity of the problem, i.e., the amount of agents.

FMC_ATA is evidently robust to a variety of communication limitations. However, when communication is close to perfect, it does converge slower than FMC_TA, e.g., when there is no message loss and message delays are very small, agents in FMC_ATA perform many steps of computation following each message reception. As a result, they send much more messages, i.e., network load grows, and consequently, the convergence process slows down.

Figures 3 and 4 present results of experiments where the algorithms solve dynamic problems, with message delay and message loss, respectively. These experiments investigate the conditions in which distributed execution is preferred over centralized execution. In the distributed version, agents detect arriving tasks, inform each other and adjust their performance accordingly, without involving a centralized entity. In the centralized implementation, all the agents information is transferred to a centralized entity that computes the agents'

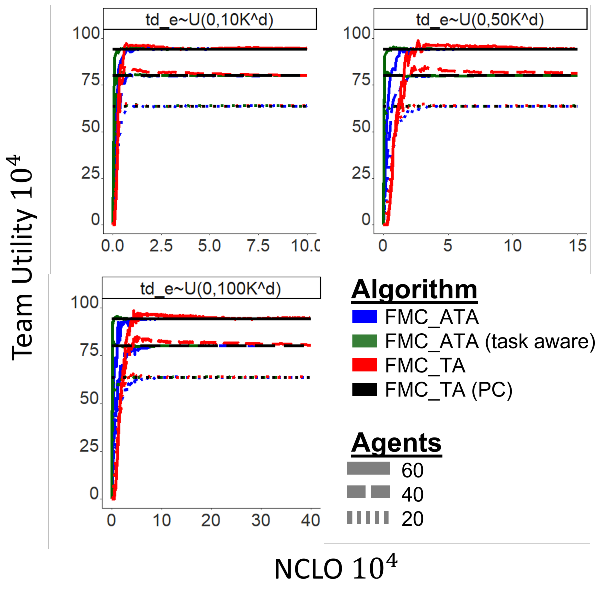

Figure 1: Team utility as a function of NCLOs in static problems. Message delays sampled from a Uniform distribution.

schedules and informs them of the result. Whenever a new task is discovered by the agents, they update the central entity, which recomputes and updates the agents with the new schedules. The shaded areas in each figure represent the estimated average error bar, based on the standard deviation of the team utility.

Since we are interested in the conditions in which distributed execution is preferred over a centralized implementation, the results presented are a function of the distance between the agents and the centralized unit. That is, what is the quality of the evolving solutions in a distributed scenario, when the centralized system is placed in different distances from the active agents. Each sub graph in figure 3 presents a the results of the algorithms solving problems in scenarios with a different communication delay pattern, i.e., a different value for the parameter UB where $td_{e_{ij}} \sim U(0, UB^{d_{ij}})$. Notice, that since the quality of solution of the distributed implementation is not dependent on the location of the centralized entity, the results of the distributed asynchronous implementation (FMC_ATA) in each of the graphs is consistent.

It is apparent from the presented results that when the central entity is placed close to the active agents, there is a slight advantage over the distributed version. This is expected since there are minor communication requirements by the centralized entity performing the centralized algorithm throughout its execution, in comparison to the repetitive message exchange between agents in the asynchronous distributed version. Notice that for distributed FMC_ATA, the team utility function decreases as the latency magnitude increases (see the blue curves in the different sub-graphs). This can be explained by the delay in convergence. The accuracy of the components used to compute the team utility (i.e., capability and soft deadline function) is harmed by delayed information delivery. Yet, this effect is much more drastic in the centralized version, when the central computing entity is

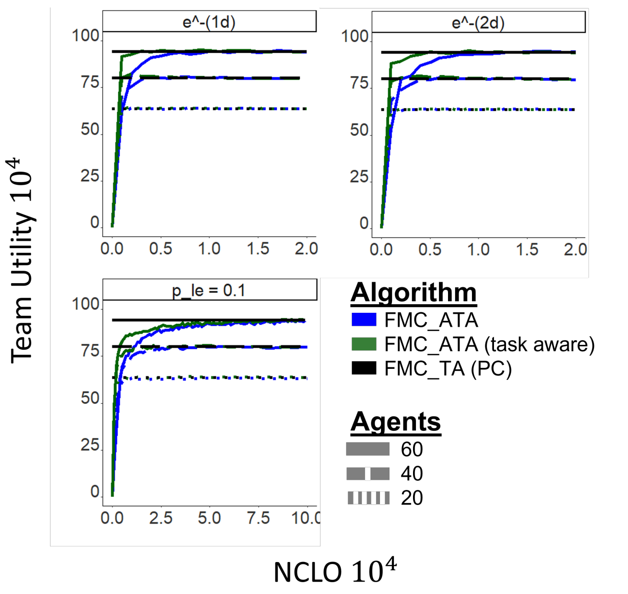

Figure 2: Team utility as a function of NCLOs in static problems with message loss probabilities sampled from a $e^{-\psi d_{ij}}$ and constant probability (0.1).

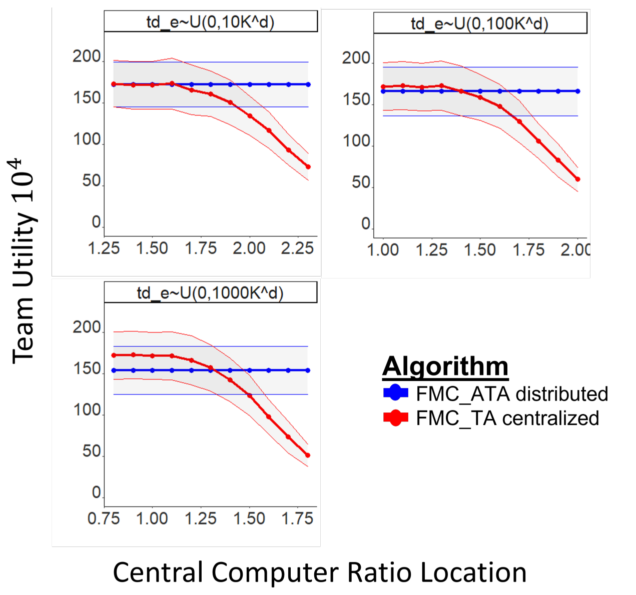

Figure 3: Team utility as a function of the central computer's location in dynamic problems, with message delays

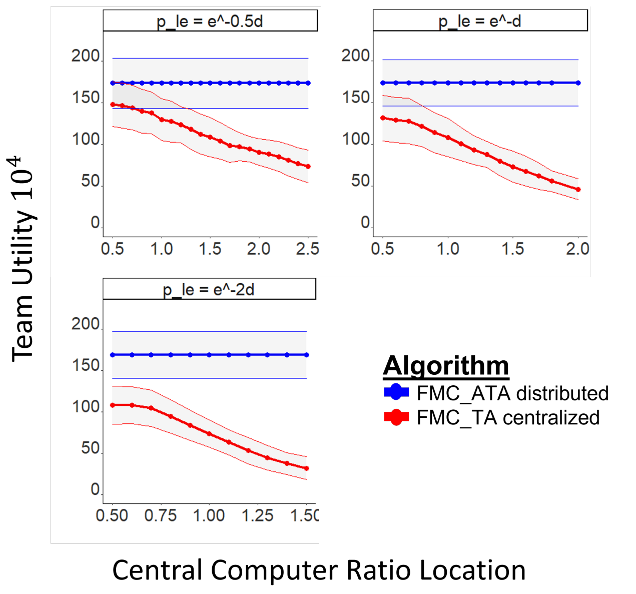

Figure 4: Team utility as a function of the central computer's location in dynamic problems with message loss

placed far from the active agents. In such cases the solution quality decreases and the distributed version is substantially favourable. Although all messages sent from the agents to the central computer and back eventually arrive at their destination, the inconsistency of the information used by the agents, decreases the quality of solution. The decisions of the centralized algorithm following dynamic events may arrive late, and as a result, the actions performed by the agents until the information arrives are not in line with the problem's requirements and thus, the utility decreases.

The results of a set of experiments, in which the scenarios included messages that were lost, is presented in Figure 4. The different values of ψ determine the probability for a message to be lost: $pl_{e_{ij}} = e^{-\psi d_{ij}}$. The solution quality of FMC_ATA remains consistent even when the probability for message arrival decreases. For centralized FMC_TA the quality of solution is substantially low even when the central computer is placed in the center of the map (very close to the active agents). Agents fail to report new information to the central entity upon the discovery of tasks, and therefore the central entity does not assign active agents to handle them.

6 Conclusions

FMC_TA is an algorithm for solving multi agent task allocation problems that has been shown to outperform state of the art centralized and distributed algorithms. However, a number of major drawbacks prevent it from being used in realistic distributed applications including imperfect communication and dynamic events. We proposed FMC_ATA, a novel asynchronous distributed algorithm in which agents perform single phase iterations and are able to detect new events and adjust their allocation and schedules accordingly. We demonstrate that FMC_ATA is robust to imperfect communication, and maintains the level of solution quality in a verity of conditions. FMC_ATA converges to the same market clearing solution as FMC_TA, thus it preserves the solution properties of FMC_TA. Moreover, while FMC_TA is not applicable to scenarios that include message loss, FMC_ATA is shows resilience in such scenarios. We presented the results of an investigation of the conditions in which a distributed FMC_ATA is preferred over the centralized implementation.

Acknowledgments

This research is partially supported by US-Israel Binational Science Foundation (BSF) grant #2018081 and US National Science Foundation (NSF) grant #1838364.

References

[Carrillo *et al.*, 2021] Estefany Carrillo, Suyash Yeotikar, Sharan Nayak, Mohamed Khalid M Jaffar, Shapour Azarm, Jeffrey W Herrmann, Michael Otte, and Huan Xu. Communication-aware multi-agent metareasoning for decentralized task allocation. *IEEE Access*, 9:98712–98730, 2021.

[Dean and Nair, 2014] Matthew D Dean and Suresh K Nair. Mass-casualty triage: Distribution of victims to multiple hospitals using the save model. *European Journal of Operational Research*, 238(1):363–373, 2014.

[Devanur *et al.*, 2002] Nikhil R Devanur, Christos H Papadimitriou, Amin Saberi, and Vijay V Vazirani. Market equilibrium via a primal-dual-type algorithm. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 389–395. IEEE, 2002.

[Farinelli *et al.*, 2008] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceeding of the 7th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 639–646, 2008.

[Farinelli *et al.*, 2017] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. Distributed on-line dynamic task assignment for multi-robot patrolling. *Autonomous Robots*, 41(6):1321–1345, 2017.

[Fioretto *et al.*, 2018] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018.

[Jones *et al.*, 2007] E Gil Jones, M Bernardine Dias, and Anthony Stentz. Learning-enhanced market-based task allocation for oversubscribed domains. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2308–2313. IEEE, 2007.

[Macarthur *et al.*, 2011] Kathryn Macarthur, Ruben Stranders, Sarvapali Ramchurn, and Nicholas Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 701–706, 2011.

[Maheswaran *et al.*, 2004] R. Maheswaran, J. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In *Proceedings of PDCS*, pages 432–439, 2004.

[Nelke and Zivan, 2017] Sofia Amador Nelke and Roie Zivan. Incentivizing cooperation between heterogeneous agents in dynamic task allocation. In *Proceedings of the 16th International Conference on Autonomous Agents and Multi-agent Systems, (AAMAS)*, pages 1082–1090, 2017.

[Nelke *et al.*, 2020] Sofia Amador Nelke, Steven Okamoto, and Roie Zivan. Market clearing-based dynamic multi-agent task allocation. *ACM Transactions of Intelligent Systems Technology.*, 11(1):4:1–4:25, 2020.

[Netzer *et al.*, 2012] Arnon Netzer, Alon Grubshtein, and Amnon Meisels. Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, 193:186–216, 2012.

[Nunes *et al.*, 2017] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, 2017.

[Otte *et al.*, 2020] Michael Otte, Michael J Kuhlman, and Donald Sofge. Auctions for multi-robot task allocation in communication limited environments. *Autonomous Robots*, 44(3):547–584, 2020.

[Rachmut *et al.*, 2021] Ben Rachmut, Roie Zivan, and William Yeoh. Latency-aware local search for distributed constraint optimization. In *20th 2nd International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 1019–1027, 2021.

[Rachmut *et al.*, 2022] Ben Rachmut, Roie Zivan, and William Yeoh. Communication-aware local search for distributed constraint optimization. *Journal of Artificial Intelligence Research*, 75:637–675, 2022.

[Ramchurn *et al.*, 2010] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. Decentralized coordination in robocup rescue. *Computer*, 53(9):1447–1461, 2010.

[Reijnierse and Potters, 1998] J Hans Reijnierse and Jos AM Potters. On finding an envy-free pareto-optimal division. *Mathematical Programming*, 83(1):291–311, 1998.

[Tadokoro *et al.*, 2000] Satoshi Tadokoro, Hiroaki Kitano, Tomoichi Takahashi, Itsuki Noda, Hitoshi Matsubara, Atsushi Shinjoh, Tetsuhiko Koto, Ikuo Takeuchi, Hironao Takahashi, Fumitoshi Matsuno, et al. The robocup-rescue project: A robotic approach to the disaster mitigation problem. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 4, pages 4089–4094. IEEE, 2000.

[Wei *et al.*, 2016] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Dynamic task allocation for multi-robot search and retrieval tasks. *Applied Intelligence*, 45(2):383–401, 2016.

[Zhang *et al.*, 2005] W. Zhang, G. Wang, Z. Xing, and L. Wittenberg. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1–2):55–87, 2005.

[Zhang, 2011] L. Zhang. Proportional response dynamics in the fisher market. *Theoretical Computer Science*, 412(24):2691–2698, 2011.

[Zivan and Meisels, 2006] Roie Zivan and Amnon Meisels. Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence(AMAI)*, 46:415–439, 2006.

[Zivan *et al.*, 2014] Roie Zivan, Steven Okamoto, and Hilla Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, 2014.

[Zivan *et al.*, 2021] Roie Zivan, Omer Perry, Ben Rachmut, and William Yeoh. The effect of asynchronous execution and message latency on max-sum. In *27th International Conference on Principles and Practice of Constraint Programming, CP*, pages 60:1–60:18, 2021.