Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 176 (2020) 675–684

Procedia Computer Science

www.elsevier.com/locate/procedia

24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Multi-Agent Task Allocation Based on the Learning of Managers and Local Preference Selections

Yuka Ishihara[a], Toshiharu Sugawara[a,*]

[a]*Department of Computer Science and Communications Engineering, Waseda University, Shjinjuku Tokyo 1698555, Japan*

Abstract

This paper discusses an adaptive distributed allocation method in which agents individually learn strategies for preferences to decide on the rank of tasks which they want to be allocated by a manager. In a distributed edge-computing environment, multiple managers that control the provision of a variety of services requested from different locations have to allocate the corresponding tasks to appropriate agents, which are usually programs developed by different companies. In our proposed method, each agent learns which manager will allocate tasks it performs well and how to declare its preferred tasks. We experimentally evaluated the proposed learning method and showed that agents using the proposed method could effectively execute requested tasks and could adapt to changes in patterns of the requested tasks.

Keywords: Multi-agent systems; Distributed task allocation; Reinforcement learning; Edge computing

1. Introduction

Task and resource allocation among heterogeneous agents by negotiation is a central issue in multi-agent systems [1] because it severely affects the performance of the entire system. However, appropriate allocation by autonomous agents in a distributed environment is challenging, because managers have to allocate tasks to provide services by taking into account many factors, such as capabilities/skills (especially, relative capabilities among agents), priority of tasks from the viewpoint of the system's requirements and utilities, the agents' individual preferences for executing tasks, the requirements/specifications of the tasks, and the workloads of managers. However, it is almost impossible to acquire all related information and to achieve optimal allocations in actual applications. Furthermore, the real-world is dynamic, so even current optimal solutions may become non-optimal soon.

For example, let us consider a system in which many types of services developed by many independent companies can be requested via smart phones in various places. The tasks necessary to achieve these services should be processed

* Corresponding author. Email: sugawara@waseda.jp ; fax: +81-3-5286-2596.
E-mail address: y.ishihara@isl.cs.waseda.ac.jp, sugawara@waseda.jp

1877-0509
10.1016/j.procs.2020.09.040

as quickly as possible. In particular, due to high-speed networks, such as 5G and more advanced future networks, the processing load on CPUs will increase due to not only executing tasks for requested services but also dispatching them to appropriate agents; thus, this may become a new bottleneck. We also have to consider the latency of interactions via communications. This suggests that as many high-demand services as possible must be processed on edge machines of wireless networks, although its processing powers are limited [19]. Moreover, the workload of service requests varies and is likely to be imbalanced, depending on locations and times. Thus, the systems must be able to dynamically allocate the requested tasks to local and neighboring edge machines, while not only using the limited available data but also by learning of reasonable allocations based on the results of past coordinated behaviors.

There are many studies on task/resource allocations through negotiation processes using no or less non-local information, such as the capabilities and the current status of other agents. They have been used in some applications [5, 16]. A well-known and landmark method is the *contact net protocol* (CNP) proposed by Smith [17] which has since been extended to overcome its drawbacks and limitations and to apply it to more complex environments [1, 16]. These methods exhibited good performances, but the required tasks are often allocated mainly to agents by looking at predefined uniform capabilities, resulting in imbalanced allocations ([6, 18]). Manager agents (or simply managers) for applications in a location-dependent and dynamic environment have to allocate tasks in a more balanced manner and to respond quickly to changes in the environment. In contrast, agents that execute the allocated tasks also have to be able to select appropriate managers (1) to be allocated more tasks that fit their capabilities and (2) to help very busy managers.

Therefore, we propose an allocation method in which all agents that are executing allocated tasks can declare which tasks they want to be allocated using their own preferential orders, whereas the manager agents allocate tasks on the basis of the agents' preferences, as well as the system's required performance, without using additional information on the agents' capabilities and their current statuses. We also consider the manager's particularity, meaning that each manager has a set of tasks requested by users, depending on location and time. Thus, agents have to be able to dynamically decide which manager they want to help with the required tasks by taking into account, in a balanced manner, their workloads, what tasks are socially more important, and the degree of similarity between an agents' capabilities and the capabilities needed to execute tasks that will be allocated by each manager. We already proposed a task allocation method using agents' preferences [10], but the previous study assumed a centralized environment in which only one manager allocates tasks to all agents. Their model, however, is too simple to use it for our current applications, in which centralized control cannot be used, because the services are provided by many independent companies; thus, we have to consider which services should be provided first, on the basis of fairness and social viewpoints.

2. Related Work

Task allocation in distributed multi-agent environments is a central issue for real-world applications for providing intelligent and sophisticated services by assigning tasks to appropriate agents in accordance with capabilities and particularities. Therefore, many studies have been conducted on task allocation methods via negotiation in distributed multi-agent environments. There are a number of approaches to this problem, such as task allocation via negotiation, a market-based approach and a team-formation problem. For example, in the team-formation problem, by assuming that a task can be done by a team of agents, agents individually try to find team members and allocate the appropriate parts of a task to each member to maximize efficiency [7, 11, 5, 20]. In Cisneros-Cabrera *et al.* [5] a team formation service for temporal coalitions in collaborative manufacturing, among various types of enterprises was proposed. Juárez and Brizuela [11] discussed team formation to find teams in social networks that have the required skills to meet the multi-objective issues.

One notable and well-known study on the negotiation-based approach is the CNP proposed by Smith [17]. This protocol has a variety of extensions [1, 4, 16]. For example, Sandholm proposed an additional mechanism for cancellation by paying a cost. Aknine, Pinson and Shakun [1] proposed an extended CNP that allows agents to declare temporal bids that can be updated if the situation has changed. These extensions could give more incentives to agents for bidding on tasks to be allocated. Noack, March and Shekh [12] compared a number of negotiation protocols for task allocations in dynamic environments. CNP and its extensions were applied to many applications, such as task allocation for multiple robots [14], cloud service composition [13], QoS in ubiquitous computing[3], supply chains[8],

control between renewable energy plants and the grid [2], and factory power management [9]. These methods usually assumed a centralized manager or distributed managers but with a shared set of tasks to allocate. However, our method focuses on a distributed environment in which multiple managers have sets of their own specific tasks that may vary over time and thus, each agent identifies with a manager and should send it a list of preferred tasks ranked by learned preference functions.

3. Preliminary

3.1. Model of Agents and Tasks

Let $A = \{1, \ldots, n\}$ be a set of n agents that executes tasks and $\mathcal{M} = \{1, \ldots, M\}$ be a set of M managers that allocates tasks, each of which corresponds to a certain service requested by a user. Agent $i \in A$ has a (computational) resource, which is represented by $R^i = (r^i_1, \ldots, r^i_H)$, where H is the number of resources, and r^i_k is a positive number. We assume that an agent with a large r^i_k has a higher capability or processing performance corresponding to the k-th resource. We also introduce a discrete time t whose unit is *tick*.

Task T is represented by tuple (S^T, w_T, d_T), where w_T is a reward that the manger and agent involved in the execution of T can receive, and d_T is the deadline of T. $S^T = (s^T_1, \ldots, s^T_H)$ is the resource required to execute T. We assume that the execution time of T by agent i is

$$E^i(T) = \max_{1 \le k \le H} \lceil s^i_k / r^i_k \rceil, \tag{1}$$

so T must be allocated to i by $d_T - E^i(T)$. If task T cannot be completed by d_T, it will be discarded and counted as a *dropped task*. Although the reward given by completing tasks T and the amount of resources/capabilities required by T may not be correlated, we assume $w_T = \sum_{k=1}^{H} s^T_k$ because we want to improve the pure performance of the entire system.

3.2. Overview of Task Allocation Process

An agent has two states, busy or free; an agent is busy (or in a busy state) if it has a task allocated that has to be performed, whereas it is free (or in a free state) when it has no allocated task. Manager $m \in \mathcal{M}$ has a set of tasks $\mathcal{T}^m_t = \{T^m_{t,1}, \ldots, T^m_{t,N_{m,t}}\}$ requested by users at time t (where $N_{m,t}$ is a non-negative integer) and m has to allocate each of them to an appropriate agent. Therefore, $\mathcal{T}^m_t$ is open to be accessed by any agents. Meanwhile, if agent $i \in A$ is free at time t, i selects one manager m with a certain strategy, and accesses $\mathcal{T}^m_t$. Then, i can also select at most $N > 0$ tasks from $\mathcal{T}^t_m$ on the basis of i's *preferential order* and generates a list whose elements are pairs of the selected task and its *utility value* (or simply *values*), whereby the value is a measure of the system's performance if the task is done by i. Then, i sorts the list according to the preferential order of the tasks, and sends it to m, which is counted as i's intention to contribute. Then, after m has received a sufficient number of sorted lists, m assigns each task in $\mathcal{T}^m_t$ to an appropriate agent with a certain allocation method and requests the agent to execute it.

We would like to clarify the difference between an agent's preferential order and the values of tasks. First, we assume that the value of task T if it is done by i is comparable and so all managers in our system want to increase the sum of their total values. Let $u^i_t(T) \ge 0$ be the value when T is allocated to i at time t for execution. For task T and agents $i, j \in A$, if $u^i_t(T) < u^j_t(T)$, it is better to allocate T to j than to i. Examples of such utilities are the rewards of tasks (w_T) and processing time taken to complete T, if it is done by i (in this case smaller is better, so for instance, we can use its reciprocal number), the duration between the time of the request of T of manager m and the time m receives a notification of completion from i, the measurable quality of the result of T done by i, and a combination of these measures. Thus, from the system's viewpoint, m tries to allocate tasks, so as to maximize their total values.

In contrast, the preference (or preferential order) of each agent is decided locally and independently, and is not comparable with the preferences of other agents in the sense that it is impossible to decide whether to prioritize a first-ranked task of one agent or that of another agent. It is also assumed that no agents can directly influence others' preferences. Therefore, the preference is independent from and may not be consistent with the value based on the system's utility. We will introduce a number of examples of preferences later.

3.3. Task Allocations based on Utility and Preference

We used a task allocation method based on the system's utility and local preference, called *single-object resource allocation with preferential order* (SORA/PO) [10, 15]. We first introduce preferential orders to all agents. For a given finite set of tasks $\mathcal{T} = \{T_1, T_2, \ldots, \}$ and agents $\forall i \in A$, $p^i(T_l)$ (≥ 1) is a positive integer indicating i's rank of task $T_l \in \mathcal{T}$. The sequence $\mathcal{T}_{p^i} = (T^i_1, \ldots, T^i_N)$ denotes the the top N of $\mathcal{T}$ that are arranged in the rank order of preference p^i, where $N \leq |\mathcal{T}|$.

When manager $m \in \mathcal{M}$ opens a task set $\mathcal{T}^m_t$ to all agents at t, agent $i \in A$ generates a sequence of pairs of a task and its associated value,

$$B^i_t = ((T^i_1, u^i_t(T^i_1)), \ldots, (T^i_N, u^i_t(T^i_N))),$$

where $p^i(T^i_k) = k$ and sends B^i_t to m. This sequence B^i_t is called the *preference list* of i and N is called the *length of the preference list*. We denote the set of agents that sent B^i_t to m at t as $A^m_t \subset A$. After a sufficient number of lists has arrived at m, attempts are made to allocate each task in $\mathcal{T}^m_t$ to agents in A^m_t. In the SORA/PO framework, tasks are allocated to appropriate agents to maximize the sum of the utility values subject to no declarations of dissatisfaction. The details are as follows.

For a subset of agent $A' \subset A$ and a set of tasks $\mathcal{T}$, the allocation of tasks to agents is defined as the subset of product $L \subset A' \times \mathcal{T}$, whereby each element in A' and $\mathcal{T}$ appears only once in L. We can naturally generate a function

$$a_L \ : \ A' \rightarrow \mathcal{T} \cup \{null\}$$

associated with L. Note that $a_L(i) = null$ indicates that no task has been allocated to $i \in A'$ and tasks in $\mathcal{T} \setminus a_L(A')$ have not yet been allocated to any agents. Then, the SORA/PO problem is to find an allocation L^* that maximizes the sum of the values $\sum_{T \in a_{L^*}(A')} u^{a^{-1}_{L^*}(T)}_t(T)$ subject to the following condition holding:

(C1) If task T is allocated to i and $\exists T'$ that appears in B^i_t s.t. T' is ranked higher than T in i $(p^i(T) > p^i(T'))$ and is allocated to another agent j, then, $u^j_t(T') \geq u^i_t(T')$.

Note that a^{-1}_L is the inverse function of a_L, i.e., $a^{-1}_L(T) \in A'$ is the agent that has been allocated T $(\in \mathcal{T})$ by L and

$$a_L(A') = \{T \in \mathcal{T} \mid T = a_L(i) \text{ s.t. } \exists i \in A'\}.$$

We also define $u^i_t(null) = 0$ for $\forall i \in A$. Obviously, if condition (C1) holds, no agents declare dissatisfaction based on local preference, because if T' is allocated to j such that $u^j_t(T') < u^i_t(T')$, i can socially contribute more to the system by doing T' than j and i prefers T' to the allocated T and thus, i has expressed dissatisfaction with the current allocation L.

For this task allocation process, we have to clarify (1) how agents decide their own preferences, (2) how a manager decides which agent a task of the tasks in $\mathcal{T}^m_t$ should be allocated to, and (3) how agents decide on the manager from which they want to receive a task. Before moving on to an explanation of the proposed method, I want to describe how managers decide on allocations. The optimal solution of the SORA/PO problem can be found using a linear integer programming package (e.g. CPLEX etc.). However, the cost of finding the optimal solution is usually high, so we use the SRNF/RD method to find solutions [10] in our experiments, because it can be used efficiently to find near-optimal solutions without incurring dissatisfaction.

4. Proposed Method

4.1. Task Allocation by Multiple Managers

We would like to extend the framework proposed by Iijima and Sugawara [10] to an environment in which multiple managers attempt to allocate tasks. In our target applications, we assume that managers are deployed in different locations and that they have their own tasks requested from users, some of which are location- and situation-dependent and that the number of requested tasks varies over time. In contrast, agents are cooperative, and therefore attempt to execute as many tasks as possible in an efficient manner. Thus, they want to execute tasks that fit their specific

capabilities as described by resource R^i (for $\forall i \in A$), but at the same time, they need to execute tasks of busy managers to fulfill the system's goals.

At time t, manager $m \in \mathcal{M}$ has a set of tasks $\mathcal{T}_t^m$ that should be allocated to appropriate agents. The total number of tasks per tick requested from all managers $W_t = \sum_{m \in \mathcal{M}} |\mathcal{T}_t^m|$ indicates the *workload* of the system. Meanwhile, when agent $i \in A$ is free, i selects one manager $m \in \mathcal{M}$ with a strategy learned by i individually and accesses m's task set $\mathcal{T}_t^m$. Then, i generates a preference list B_t^i based on preference p^i that is also selected locally through learning and sends it to m. Manager m decides on the allocation L_t^m by using the SORA/PO framework and sends the corresponding tasks to the chosen agents. In the following sections, we explain how agents learn to decide on preferences and managers within the framework of our proposed method.

4.2. Preference Strategy for Manager Agents

We assume that all agents have a set of managers $\mathcal{M}$. This assumption is based on a situation in which agents know all managers around them in a widely distributed system . The proposed learning method, which is used to decide on individual preferences of each agent is similar to one previously proposed [10], except that it uses different preference functions for each manager.

We describe how an agent decides their preferences depending on the manager to which it will submit its preference list. Agent $\forall i \in A$ has a set of *preferences* P consisting of the following functions to decide on i's preference:

Highest reward first (HRF): The HRF preference function determines the rank of tasks based on their rewards. So, agents with this preference are likely to give higher priority to tasks which require a large amount of resources because $w_T = \sum_{k=1}^{H} s_k^T$.

Earliest deadline first (EDF): An agent with this preference function gives a higher rank to tasks that are close to and can be executed by the deadline, and so they try to improve the system's performance by choosing tasks that are likely to be discarded. The EDF will reduce the number of tasks dropped due to run-out.

Shortest processing time first (SPTF): Agent i with a SPTF preference function gives a higher rank to task T, whose execution time $E^i(T)$ is shorter. Agents with SPTF attempt to complete as many tasks as possible, so are likely to choose easy-to-complete tasks whose sum of resources $\sum_{s \in S^T} s$ is smaller. Therefore, if a system's utility is to increase the total rewards by task completion, SPTF may give an opposite preference.

Cost effectiveness (CE): Agent i with a CE preference function gives a higher rank to tasks T whose reward per tick $w_T / E^i(T)$ is large. This also indicates that i prefers tasks whose required resource structure S^T is similar to (so, fits) i's resource structure R^i.

Agent i learns the appropriate preference strategy $p \in P$ for manager $\forall m \in \mathcal{M}$, where P = {HRF, EDF, SPFT, CE} to maximize the expected value by executing the tasks allocated by m. Then, when agent i is free, i generates a preference list B_t^i using p for m and submits it to m.

To submit B_t^i to manager m at time t, i selects the preference function $p^i(m,t) \in P$ whose Q-value $Q_t^i(m,p)$ is the largest;

$$p^i(m,t) = \arg\max_{p \in P} Q_t^i(m,p). \tag{2}$$

If there are multiple strategies that result in the maximum Q-value, only one of them will be randomly selected. Agents adopt ε-greedy strategy for the actual selection, i.e., they select $p^i(m,t)$ as defined in Formula (2) with probability $1 - \varepsilon_p$; otherwise they select a preference function from P randomly.

The Q-value $Q_t(m,p)$ is updated as follows. First, $Q_0^i(m,p)$ is initialized to 0. Suppose that agent i submitted $B_{t_0}^i$ (using preference function p^i) to m at t_0 and received task T allocated by m (using the SORA/PO framework) at t_1. Then, i updates $Q_t^i(m,p)$ by

$$Q_{t_2+1}^i(m,p^i) = (1-\alpha_p) \cdot Q_{t_2} t^i(m,p^i) + \alpha \cdot u_{t_1}^i(T) \tag{3}$$

only when i has completed the allocated task T at t_2, for which $0 \le \alpha_p \le 1$ is the learning rate and $t_0 < t_1 < t_2$. Note that $u_{null}^i = 0$. We also note that $Q_t^i(m,p)$ is not updated except in the above case.

4.3. Selection of Manager

Agent i also learns which manager is likely to allocate better tasks to i using recent experience. For this purpose, i has a value of $Q_t^i(m)$ for $\forall m \in \mathcal{M}$ and selects manager m_t^i at t to send the preference list by

$$m_t^i = \arg\max_{m \in \mathcal{M}} Q_t^i(m) \tag{4}$$

with probability $1 - \varepsilon_m$; otherwise m_t^i is selected randomly from P.

Agent i updates the value of $Q_t^i(m)$ as follows. After submitting $B_{t_0}^i$ to $m = m_{t_0}^i$ at time t_0, i is allocated task T from m (T may be a null task) at t_1 and then executes it immediately. When i has completed T at t_2, i updates $Q_t^i(m)$ by

$$Q_{t_2+1}^i(m) = (1 - \alpha_m) \cdot Q_{t_2} t^i(m) + \alpha \cdot u_{t_1}^i(T) \tag{5}$$

only when i has completed the allocated task T at t_2, whereby $0 \leq \alpha_m \leq 1$ is the learning rate. Note that agents first select manager m in accordance with the value of $Q_t^i(m)$ and look at the task set. Then, they select the preference functions in accordance with $Q_t^i(m, p)$ to generate a preference list B_t^i to send to m.

5. Experimental Evaluation and Discussion

5.1. Experimental Setting

We experimentally evaluated the proposed method in a simulated environment, in which multiple managers had their own tasks in order to provide many and various services to users. We defined that the system's utility value, which is shared with all agents in this system, is cost effectiveness (CE) $w_T / E^i(T)$; therefore, managers try to earn as many rewards as possible. We showed that learning of the preference functions and manager selection could lead to excellent performance and agents using our method could respond to changes in the types and numbers of requested tasks flexibly. We also compared these performances with others, by combining cases when (1) all agents use a common and fixed preference function, (2) they have their own fixed managers and (3) they select their managers randomly every time. Note that in case (1), we assume that all agents have adopted the CE function because it resulted in an improved performance compared with other preference functions and the CE is compatible with the system's utility. In addition, this case is close to the case of using CNP, which is a well-known method. This comparison is useful to see the difficulty of allocations in distributed environments, which consist of autonomous agents with a uniform selection.

We set the number of computational resources to $H = 3$ and introduced three types of agents with capabilities based on the characteristics of a given resource $R^i = (r_1^i, r_2^i, r_3^i)$. Agent $i \in A$, which is called the *balanced agent*, has balanced abilities in the sense that the amounts of the three available resources are almost similar; so, we defined i's resources randomly subject to

$$4 \leq r_k^i \leq 5 \quad (k = 1, 2, \text{ and } 3). \tag{6}$$

Other types of agents have unbalanced resources but have one relatively higher capability corresponding to the 1st or the 3rd resource. Thus, their resources were set under the following conditions

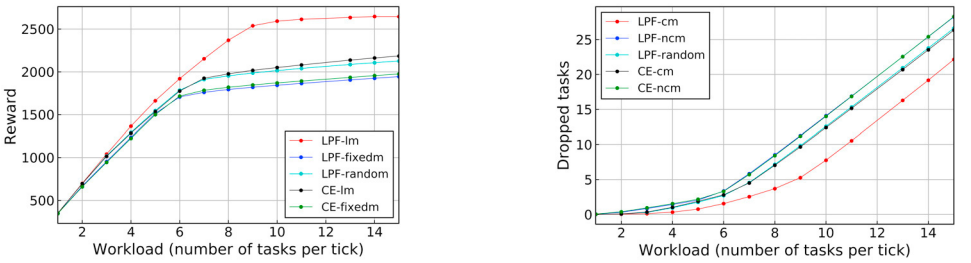
$$\begin{aligned} 6 \leq r_k^i \leq 7 \quad & (k = 1, \text{ or } 3) \\ 2 \leq r_k^i \leq 3 \quad & (\text{for other } k) \end{aligned} \tag{7}$$

Note that the value of k depends on the type of resource. Thus, agent $i \in A$ with a higher first resource has the resources of $r_1^i = 6$ or 7, and $r_2^i, r_3^i = 2$ or 3. The type of agent with a high 1st resource is called the *type-1 agent*). Similarly, an agent with a high third resource is called the *type-3 agent*. Balanced agents, type-1 agents, and type-3 agents were generated at the rate of $g_b \geq 0$, $g_1 \geq 0$, and $g_3 \geq 0$, respectively in our experiments, where $g_b + g_1 + g_3 = 1$. We set $g_b = 0.2$ and $g_1 = g_3 = 0.4$ in our experiments.

We also introduced four types of tasks based on the resource structure $S^T = (s_1^T, s_2^T, s_3^T)$ required to execute them. A *balanced tasks* that requires the almost same amount of computational resources; thus, the amount of required

Table 1: Parameters and values used in the experiments.

Parameter	Value (or initial value)
Number of agents ($\|A\|$)	$n = 300$
Number of managers $\mathcal{M} = \{M_1, M_2, M_3\}$	$M = 3$
Workload (number of tasks per tick for each manager) $W_t/3$	Variable (1 to 15)
Deadline of task completion	Random value between 15 and 25



(a) Reward per tick

(b) Number of dropped tasks per tick

Fig. 1: Performance comparison.

resources of a balanced task were set randomly to

$$30 \leq s_k^i \leq 60 \quad (1 \leq k \leq 3) \tag{8}$$

The other three types of unbalanced tasks are defined so that only one specific resource has to be set to be large.

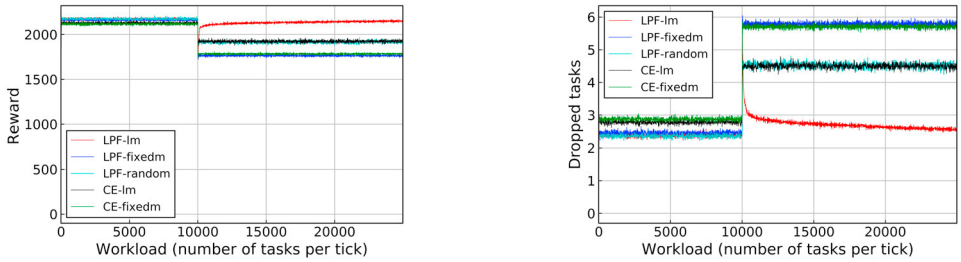
$$\begin{aligned} 50 \leq s_k^i \leq 80 \quad & (k = 1, 2 \text{ or } 3 \text{ depending on the type}) \\ 15 \leq s_k^i \leq 35 \quad & (\text{Other than the above } k) \end{aligned} \tag{9}$$

The unbalanced task that required a larger amount of the k-th resource is called the *type k task*, after this. The request ratios of these types of tasks are denoted by h_b, h_1 h_2 and h_3, where $h_b + h_1 + h_2 + h_3 = 1$. We set the probabilities of randomness in ε-greedy to $\varepsilon_m = \varepsilon_p = 0.01$ and the learning rates to $\alpha_m = \alpha_p = 0.1$. Other parameters used in our experiments are listed in Table 1. The data shown below are the average values of 100 experimental runs.

5.2. Experiment 1 — Performance comparison

We investigated the total reward for completing tasks and the number of dropped tasks to evaluate the performance of the entire system when using our proposed method. In the first experiment (Exp. 1), we examined a situation in which managers M_1 M_2 and M_3 have different task request patterns; for all managers, $h_b = 0.1$, but $h_1 = 0.9$ for M_1, $h_2 = 0.9$ for M_2, and $h_3 = 0.9$ for M_3. Hence, for example, $h_b = 0.1, h_1 = 0.9$ and $h_2 = h_3 = 0$ for M_1. With various values of W_t from 3 to 45 (even workload for each manager), I plotted the number of executed tasks before their deadlines and the number of dropped tasks, as shown in Fig. 1, where each label represents the method agents chose; label "LPF" indicates the learning of preference functions, label "CE" indicates the use of the fixed preference function CE, label "lm" indicates the learning of the managers, "random" indicates a random selection of managers, and "fixedm" indicates that each agent is assigned to one manager randomly and cannot change it for the duration of an experimental run. The conditions of each experiment will be represented by the corresponding labels, after this. Hence, the proposed method corresponds to label *LPF-lm* and therefore is called LPF-lm method (other methods are also named using the above mentioned labels in the following sections).

Figure 1a indicates that the proposed method always received the highest rewards, meaning that the resources of agents were being used effectively. Similarly, the number of dropped tasks were the smallest whenever agents used the proposed *LPF-lm* method. If we compared these results with methods *LPF-fixedm* and *LPF-random*, the adaptive and



(a) Received rewards

(b) Dropped tasks

Fig. 2: Received rewards and number of dropped tasks in Exp. 2

autonomous selection of managers by individual agents improved the adaptive uses of agents' resources, resulting in a higher efficiency and a lower number of dropped tasks. We can also observe that the results using the *LPF-random* method were better, because even if agents selected managers randomly, they could learn which preference function was better for a selected manager.

The performances when using the *CE-lm* and *CE-fixedm* methods were lower than of the proposed method. We think that these methods are similar to CNP [17], because managers opened the sets of tasks (i.e. announced) to all agents and agents (bidders in the CNP) could bid on tasks of given or selected managers. The resulting allocations were usually decided on the basis of the system's utility, which is the CE in our experiment. Our experimental results show that by declaring a list of tasks with their ranks of preference flexibly based on the preference functions learned, agents could contribute to the system's performance, and thus, received more rewards.

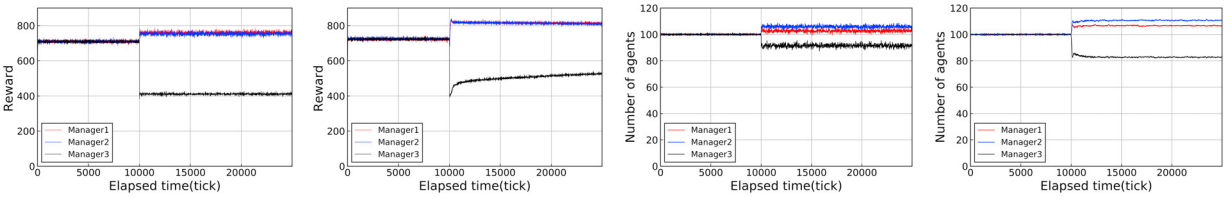
5.3. Experiment 2 — Response to changes and reorganization

In the second experiment (Exp. 2), we investigated the possibility of adaptively following changes in task request patterns and the resulting efficiency. For this purpose, we fixed the workload of each manager to 7 (so, a total of $W_t = 21$ tasks per tick were requested in the entire system. Under this condition, the number of dropped tasks started to increase). In Exp. 2, all managers were assigned the same task request ratios of $h_b = 0.1$ and $h_1 = h_2 = h_3 = 0.3$ until 10000 ticks were reached. After 10000 ticks, managers M_1 M_2 and M_3 had different task request patterns, the same as those in Exp. 1. All plots of all figures in Exp. 2 are the averages of the data of every 10 ticks.

The received rewards and the number of dropped tasks for agents adopting various methods are plotted in Fig. 2. This figure indicates that agents using our proposed method (*LPF-lm*) could exhibit the best performance in terms of the received rewards and the number of dropped tasks, and quickly adapted to any changes in the task request patterns. In the first half of Exp. 2, the methods, including the learning of preference functions (*LPF*) were determined to be slightly superior to those with a fixed preference function, although there was not a large difference. This is because all managers had the same pattern of tasks and the learning of appropriate managers did not have much influence. However, after the latter half of Exp. 2, all methods, except for the proposed method exhibited lower rates of received rewards and increased numbers of dropped tasks. In contrast, the proposed method outperformed others and quickly recovered from the temporarily reduced efficiency, probably because the combination of the learning of the selections of preference functions and of managers sending task lists worked effectively.

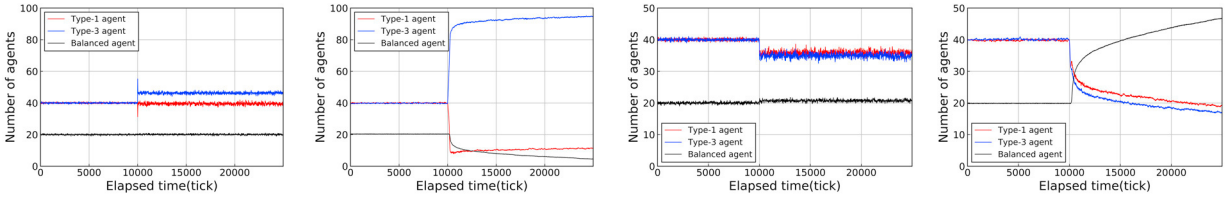
To understand how agents using the proposed learning followed changes, we analyzed transitions in rewards received by individual managers and the numbers of agents that selected each manager over time. These results are plotted in Fig. 3. By comparing Figs. 3a and 3b, managers M_1 and M_2, which were using the proposed method could improve more than those using the CE-lm method. In addition, M_3's temporarily reduced efficiency improved immediately. Note that M_3 had many type-2 tasks, but no agent had the resources that matched type-2 tasks well, so M_3 received lower rewards than others. Figures 3c and 3d, in which the numbers of agents selecting each manager are plotted, indicates that agents switched and therefore were able to select better managers for themselves more often when using the proposed method than those using the *CE-lm* method.

We analyzed which type of agents selected managers in the first and latter halves. Figure 4 plotted the the transition of the number of breakdowns by type of agent that selected M_2 or M_3. As shown in Figs. 4a and 4c, the changes in



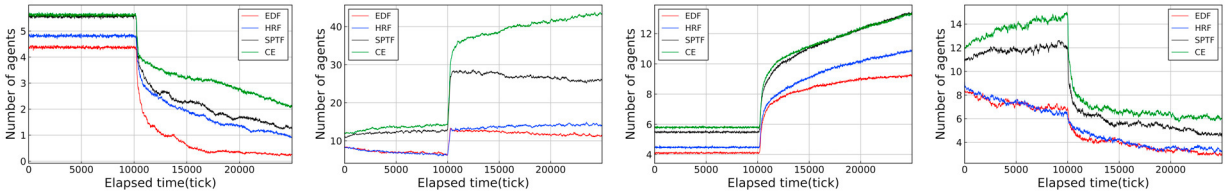
(a) Rewards of each manager (CE-lm) (b) Rewards of each manager (LPF-lm: Proposed) (c) Number of agents selecting each manager (CE-lm) (d) Number of agents selecting each manager (LPF-lm: Proposed)

Fig. 3: Rewards managers received and number of agents selecting each manager.



(a) Agents selecting M_2 (CE-lm) (b) Agents selecting M_2 (Proposed) (c) Agents selecting M_3 (CE-lm) (d) Agents selecting M_3 (proposed)

Fig. 4: Breakdown by agent types selecting M_2 and M_3.



(a) Balanced agents selecting M_2 (b) Type-3 agents selecting M_2 (c) Balanced agents selecting M_3 (d) Type-3 agents selecting M_3

Fig. 5: Selected preference functions.

agents that selected these managers were limited after the task request pattern was changed to 10000 tick. In particular, type-3 agents should concentrate more on M_2 as they could perform M_2's tasks more efficiently. Similarly, type-1 agents also had better outcomes when switching to M_3, however, they did not. In contrast, Figs. 4b and 4d indicate that all types of agents were dynamically reorganized by autonomously switching to managers that provided tasks matched their resources.

If we look closely at around 10000 ticks in these graphs, type-1 and type-3 agents first switch to M_1 and M_2. Then, this movement caused a bias in the number of agents selecting each manager, especially as M_3 became busier. Thus, balanced agents started switching to M_3. Note that Fig. 3d indicates that the number of agents selecting M_3 changed little after 10000 ticks but the rewards received by M_3 gradually improved, as shown in Fig. 3b. Although the number of agents did not change, Fig. 4d shows that many agents still continued to change their managers.

Finally, we investigated a breakdown of the selected preference functions. The results are shown in Fig. 5. This figure indicates that agents adaptively selected preference functions in the first and latter halves. We could also observe that they mainly selected the CE (i.e., received rewards per tick) because this also supported the system's utility to improve. However, agents with comparatively lower resources than other agents chose other preference functions to be allocated tasks and increased their chances of contributing to the system, because if these agents would have selected CE, like others, they might have been allocated no tasks due to their lower resources.

6. Conclusion

We proposed an adaptive distributed allocation method, in which agents individually learn strategies for preferences to decide on the rank of tasks they want to be allocated in an environment, in which multiple managers have specific tasks to provide the requested services. Our method is an extension of the method proposed by Iijima and Sugawara [10], in so far as agents can autonomously select appropriate managers on the basis of their resources/capabilities and the types of requested tasks. We conducted experiments to evaluate the proposed method and showed that agents using the proposed method were able to select managers appropriately and to switch dynamically according to the nature of the tasks given by the managers. We plan to extend our method for applications in an environment in which communication delays cannot be ignored.

References

[1] Aknine, S., Pinson, S., Shakun, M.F., 2004. An extended multi-agent negotiation protocol. Autonomous Agents and Multi-Agent Systems 8, 5–45. doi:10.1023/B:AGNT.0000009409.19387.f8.

[2] Bari, Z., Ben Yakhlef, M., 2016. A MAS based energy-coordination by contract net protocol for connecting the production units of energy renewable based wind-photovoltaic energy storage system to Grid, in: 2016 International Conference on Information Technology for Organizations Development (IT4OD), pp. 1–8. doi:10.1109/IT4OD.2016.7479321.

[3] Bhirud, N.S., Tatale, S.B., 2016. Optimization of quality of service (QoS) parameters using Contract Net Protocol in Ubiquitous Computing, in: 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), pp. 1126–1130.

[4] Cano, J., Carbó, J., 2006. Protocol for a truly distributed coordination among agents in competitive survival video-games, in: Developing Ambient Intelligence, Springer Paris, Paris. pp. 48–59.

[5] Cisneros-Cabrera, S., Sampaio, P., Mehandjiev, N., 2018. A B2B Team Formation Microservice for Collaborative Manufacturing in Industry 4.0, in: 2018 IEEE World Congress on Services (SERVICES), pp. 37–38.

[6] Gu, C., Ishida, T., 1996. Analyzing the Social Behavior of Contract Net Protocol , in: de Velde, W.V., Perram, J.W. (Eds.), Proceedings of 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 96), LNAI 1038, Springer-Verlag. pp. 116 – 127.

[7] Hayano, M., Hamada, D., Sugawara, T., 2014. Role and member selection in team formation using resource estimation for large-scale multi-agent systems. Neurocomputing 146, 164 – 172. doi:10.1016/j.neucom.2014.04.059.

[8] Hsieh, F.S., 2015. Scheduling Sustainable Supply Chains Based on Multi-agent Systems and Workflow Models, in: 10th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), pp. 252–259. doi:10.1109/ISKE.2015.20.

[9] Iijima, K., Ikemoto, Y., Fukumoto, T., 2015. A Decentralized Negotiation Method for Stable Resource Allocation Based on Symbiotic-Autonomous Decentralized System Concept, in: 2015 IEEE 12th International Symp. on Autonomous Decentralized Systems, pp. 130–135.

[10] Iijima, N., Sugiyama, A., Hayano, M., Sugawara, T., 2017. Adaptive Task Allocation Based on Social Utility and Individual Preference in Distributed Environments. Procedia Computer Science 112, 91 – 98. doi:10.1016/j.procs.2017.08.177.

[11] Juárez, Julio and Brizuela, Carlos A., 2018. A Multi-Objective Formulation of the Team Formation Problem in Social Networks: Preliminary Results, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA. p. 261 - 268. doi:10.1145/3205455.3205634.

[12] Noack, K., Marsh, L., Shekh, S., 2015. Negotiation Protocol Comparison for Task Allocation in Highly Dynamic Environments, in: MODSIM2015, 21st International Congress on Modelling and Simulation, Modelling and Simulation Society of Australia and New Zealand. pp. 490–496. URL: http://www.mssanz.org.au/modsim2015/C6/noack.pdf.

[13] Padmavathi, S., Dharani, V.P., Maithrreye, S., Devi, M.M., Durairaj, S., 2016. Multi-Agent Framework For Cloud Service Composition, in: 2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS), pp. 1–5. doi:10.1109/ICETETS.2016.7603011.

[14] Pu-cheng, Z., Yusheng, H., Mo-gen, X., 2007. Extended contract net protocol for multi-robot dynamic task allocation. Information Technology Journal 6, 733–738.

[15] Saito, K., Sugawara, T., 2016. Assignment Problem with Preference and an Efficient Solution Method Without Dissatisfaction, in: Jezic, G., Chen-Burger, J.Y.H., Howlett, J.R., Jain, C.L. (Eds.), Proceedings of 10th KES International Conference on Agent and Multi-Agent Systems: Technology and Applications. Springer, pp. 33–44. doi:10.1007/978-3-319-39883-9_3.

[16] Sandholm, T., 1993. An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations, in: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 256–262.

[17] Smith, R.G., 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers C-29, 1104–1113.

[18] Sugawara, T., Hirotsu, T., Kurihara, S., Fukuda, K., 2008. Adaptive Manager-side Control Policy in Contract Net Protocol for Massively Multi-Agent Systems, in: Proc. of 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS2008), IFAAMAS. pp. 1433–1436.

[19] Yi, S., Hao, Z., Qin, Z., Li, Q., 2015. Fog computing: Platform and applications, in: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pp. 73–78. doi:10.1109/HotWeb.2015.22.

[20] Zakarian, A., Kusiak, A., 1999. Forming teams: an analytical approach. IIE Transactions 31, 85–97. doi:10.1023/A:1007580823003.