Journal of Information Processing Vol.22 No.2 289–298 (Apr. 2014)

[DOI: 10.2197/ipsjjip.22.289]

Regular Paper

Learning of Task Allocation Method Based on Reorganization of Agent Networks in Known and Unknown Environments

Kazuki Urakawa[1,a)] Toshiharu Sugawara[1,b)]

Received: July 6, 2013, Accepted: January 8, 2014

Abstract: We propose a team formation method that integrates the estimating of the resources of neighboring agents in a tree-structured agent network in order to allocate tasks to the agents that have sufficient capabilities for doing tasks. A task for providing the required service in a distributed environment is often achieved by a number of subtasks that are dynamically constructed on demand in a bottom-up manner and then done by the team of appropriate agents. A number of studies were conducted on efficient team formation for quality services. However, most of them assume that resources in other agents are known, and this assumption is not adequate in real world applications. The contribution of this paper is threefold. First, we extend the conventional method by combining the learning of task allocation and the reorganization of agent networks. In particular, we introduce the elimination of links as well as the generation of links in the reorganization. Second, we revise the learning method so as to use only information available locally. Finally, we omitt the assumption that all resource information in other agents is given in advance. Instead, we extend the task allocation method by combining it with the resource estimation of neighboring agents. We experimentally show that this extension can considerably improve the efficiency of team formation compared with the conventional method even though it does not require knowledge of resources in other agents. We also show that it can make the agent network adaptive to environmental changes.

Keywords: distributed task allocation, team formation, coordination, learning, reorganization

1. Introduction

The task/resource allocation in multi-agent systems (MAS) contexts has received a great deal of interest since it is a fundamental problem applicable to many applications. For example, a service provided by networked computers on the Internet is realized by doing a corresponding task that consists of a number of subtasks dynamically constructed in a bottom-up manner [11], [16]. This framework is often referred to as service-oriented computing or service-oriented multi-agent systems [10], [16], in which an agent corresponds to a server or program. Because they are developed and deployed by a different organization or company to undertake specialized tasks and/or subtasks however, it is usually impossible in a distributed open environment to get all relevant information necessary for appropriate allocations within a reasonable time. Furthermore, these agents have to be selected for timely service provision by taking into account currently available resources/capabilities as well as the user's requirements. Since the failure or delay of only one subtask for the service results is a significant delay or failure of the required service, efficient distributed task allocation is a key goal for services meeting the required quality.

This type of resource allocation in a distributed environment is often referred to as team formation (or coalitional formation) problems in MAS literature [10], [14]. A *team* consists of (1) a set of agents that have sufficient available resources for doing all subtasks for the required task and (2) the assignment of the team's subtasks to the agents [18]. If we consider the timely and quality services on the Internet, the agents' teams must dynamically and adaptively as well as efficiently be formed in accordance with the request patterns and frequencies of services, workloads, and types of currently available agents. However, this is a challenging problem because agents always have to do this with uncertain information related to other agents.

A number of studies proposed dynamic team formation algorithms. For example, Abdallah and Lesser proposed an effective team formation method that uses reinforcement learning in a tree-structured, that is, hierarchical, agent network [2]. However, in their approach, after sufficient learning time, tasks are likely to be allocated only to specific agents and, thus, this results in unbalanced use of resources in the agent network. To avoid the imbalance, Ref. [12] proposed incorporating a learning-based reorganization method into that proposed in Ref. [2]. However, we found that their reorganization method often did not work well because the reorganization process stopped, and so became meaningless in the earlier stages of learning. Moreover, their learning method assumed information on resources other agents manage/have and

[1] Department of Computer Science and Engineering, Graduate School of Waseda University, Shinjuku, Tokyo 169–8555, Japan
[a)] k.urakawa@isl.cs.waseda.ac.jp
[b)] sugawara@waseda.jp

This work was supported in part by KAKENHI (25280087). This paper is the extended version of the conference paper [17].

internal intermediate learning states in other agents; this assumption is not acceptable in actual applications.

The contribution of this paper is threefold. First, we propose a more effective task allocation method that is incorporated with the reorganization method in a hierarchical agent network. We also added the link elimination process into our method to improve the efficiency. Second, we revised the learning method in Ref. [12] so as to assume no internal intermediate learning states in other agents. Third, we extend this method by incorporating it with the resource estimation method for *unknown environments*, meaning that the resources in other agents are unknown to each other. Similarly, we call environments *known environments* if all resource information in other agents is available in advance.

This paper is organized as follows. We first explain the related work and then describe the model of agent and tasks and the issue addressed here. In Section 4, the proposed learning with reorganization for the team formation problem and resource estimation method are presented. After that, we evaluate it experimentally by comparing it with the conventional and random methods and analyze the characteristics of the proposed methods. The experimental results show that the proposed method can achieve more effective team formation in accordance with the given environments.

2. Related Work

A number of studies have been done on efficient and dynamic team formation methods. From a theoretical aspect, for example, Sheholy and Kraus [14] proposed an optimal solution of allocating tasks to agents with resources. However, the algorithm used was exponential. They also introduced a polynomial-time method of calculating Pareto-optimal solutions [15], although the solutions did not provide the maximum utilities for the entire system. Reference [19] has shown an optimal team corresponding to the agents and multiple tasks using a genetic algorithm. However, their research assumed that the agents with their associated resources/capabilities and the tasks were already known to all agents in advance.

Genin and Aknine [8] proposed a method of learning for efficient team formation in a task-oriented domain. Agents retain information on which tasks are proposed and accepted, and they decide which agents are likely to join the team by using it. However, they assumed that all messages between agents are observable. Reference [9] revised this method to improve the efficiency of team formation by just using the past local experience and parameter learning, but this assumed that the resources/capabilities required by each task are known before task allocation.

As mentioned in the previous section, Abdallah and Lesser proposed an effective team formation method that uses reinforcement learning in a hierarchical agent network [2]. In their approach, all the agents in the network learn, from past experiences, the policies for task allocations to other agents at the lower levels of the hierarchy, and thus, effective and efficient team formation can be achieved. However, in their approach, after sufficient learning time, tasks are likely to be allocated only to specific agents. These learned but biased activities lead to an unbalanced use of resources in the network, thus, busy and unbusy agents may coexist. To avoid this imbalance, Ref. [12] proposed the reorganization method and incorporated it into the learning-based team formation proposed in Ref. [2]. However, their reorganization method was not effective because agents with relatively many resources are frequently identified as unbusy even though they are already assigned many tasks. Another approach for task allocations in MAS was proposed in Ref. [16]; the proposed method was quite efficient and applicable to the actual system. However, it was based on a mobile agent framework, so their approach is different from ours. More importantly, all the studies above assume that information about the resources in other agents is already known to each other, but this assumption is often not acceptable in real-world settings.

3. Agent and Task Model

3.1 Agent Network

An agent network is denoted by *graph* (A, E), where A is the set of agents and E is the set of edges connecting two agents. We assume that an agent network has a hierarchical structure as in Refs. [2], [12] because a hierarchical network can easily provide the efficiency needed for decision-making and actual computational nodes in large-scale application systems often form tree-structured overlay networks. An example is shown in **Fig. 1**, where a node expresses the agents running in the experimental equipment. In hierarchical networks, we assume that there are two kinds of agents: *managers* and *individuals*. An individual is placed on a leaf in the hierarchy and plays the role of allocating its computational resources to the equipped tasks for execution. Managers are on the nodes without leaves and play the role of selecting a set of subtasks and allocating it to one of the *child agents*, that is, the agents connected directly under them. The agent directly connected to the upper level of the hierarchy is called the *parent agent* (for example, m_1 is the parent agent of m_3 and m_4, and m_3 and m_4 are child agents of m_1 in Fig. 1). The manager at the root is the *root manager* and denoted as *RM*. In this paper, we first assume that agents know only (1) the names of the adjacent agents that are the parent and child agents and (2) the amount of resources each child agent manages or has. Resources that an individual has or a manager manages are described below. We will omit the latter assumption for unknown environments.

The smallest unit of simulation time is called a *step*, and each agent takes one primitive action such as message sending and task execution. We assume that one step corresponds to 10 msec.

3.2 Tasks and Resources

Suppose that $\mathcal{T}$ is the set of all kinds of tasks and p types of

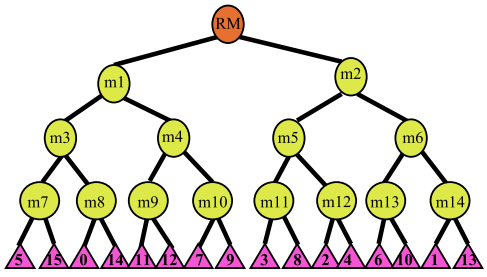

Fig. 1 Example of initial network structure.

resources are used to execute tasks, where p is a positive integer. Task $T_i \in \mathcal{T}$ consists of a number of subtasks and is denoted by $T_i = \{t_1^i, t_2^i, \cdots, t_q^i\}$, where t_j^i is the subtask of T_i. Subtask t_j^i is represented by the tuple $\langle u_j^i, r_1^{i,j}, r_2^{i,j}, \cdots, r_p^{i,j}\rangle$, where u_j^i is the utility earned if subtask t_j^i is allocated to the resources for execution and $r_k^{i,j}$ is the amount of k-th resources necessary for executing subtask t_j^i. Therefore, T_i is also expressed by the tuple $\langle U_i, R_1^i, R_2^i, \ldots, R_p^i\rangle$, where $U_i = \sum_{t_j^i \in T_i} u_j^i$ and $R_k^i = \sum_{t_j^i \in T_i} r_k^{i,j}$. We assume that utility is identical to the reward that is used in the reinforcement learning.

3.3 Agent Model

Individual $l \in A$ is denoted by the tuple $\langle cr_1^l, cr_2^l, \cdots, cr_p^l\rangle$, where cr_k^l is the amount of k-th resources owned by l. If l receives a set of subtasks from its parent agent m_i, it tries to allocate the resources for the given subtasks. The amount of resources that l has indicates the throughput of resources required in subtasks for one step. Thus, the time required for executing the subtask $t_j^i = \langle u_i, r_1^{i,j}, r_2^{i,j}, \cdots r_p^{i,j}\rangle$ is expressed by

$$D(l, t_j^i) = \left\lceil \max\left(\frac{r_1^{i,j}}{cr_1^l}, \cdots, \frac{r_p^{i,j}}{cr_p^l}\right)\right\rceil,$$

where, if $cr_k^l = 0$, we define

$$\frac{r_k^{i,j}}{cr_k^l} = \infty \text{ (if } r_k^{i,j} \neq 0) \text{ and } \frac{r_k^{i,j}}{cr_k^l} = 0 \text{ (if } r_k^{i,j} = 0).$$

We extend this notation for the set of subtasks W ($\subset T_i$); the time required for executing W is

$$D(l, W) = \left\lceil \max\left(\frac{r_1^W}{cr_1^l}, \cdots, \frac{r_p^W}{cr_p^l}\right)\right\rceil, \quad (1)$$

where $r_k^W = \sum_{t_j^i \in W} r_k^{i,j}$.

The resources that a manager manages are defined recursively from the bottom level of the total (that is, the sum of) resources of the child agents that are individuals or managers. That is, for manager agent $m \in A$, the amount of k-th resources managed by m is defined as

$$cr_k^m = \sum_{a \in ch(m)} cr_k^a, \quad (2)$$

where $ch(m)$ is the set of the child agents of m. We assume that the resources in agents are computational resources like CPUs/memory and the installed software. Thus, the resources are assumed to be stable and do not change so frequently. Therefore, the certain resources are always slow and have some delay in processing their tasks. In the unknown environment, agents do not know the resources that their child agents manage or have in our environment, so they have to estimate the amount of resources.

We also define parameter δ (a positive integer) as the deadline of team formation; that is, all managers have to try to find an individual that can process the set of subtasks within δ steps. If the manager cannot find such an individual, the allocated subtasks are discarded. For task T_i, the set of individuals involved in the process of subtasks in T_i is called the *team* and is denoted as C_{T_i}. For $W \subset T_i$, the set of individuals allocated subtasks in W is C_W. The agent network earns the utility of a task when all its subtasks are successfully allocated to individuals. The performance is measured by the sum of utilities earned by all agents in the network during a certain period. The problem addressed in this paper is how to make the agent network earn as many utilities as possible.

3.4 Communication Delay

We incorporate the concept of communication delay into the agent network model. Communication in this paper is any kind of message between agents, such as allocation of a set of subtasks, a success or failure response of the allocation, and information delivery. We assume that an agent can communicate only with adjacent agents in a network. A communication delay is usually defined as the sum of the latency of transmission (this is usually proportional to the distance) and elapsed time for processing, and the latter is further made up of the required times for a number of activities such as processing received messages, decision-making for subsequent messages based on the learned strategies, and preparing messages to be sent.

Due to recent broadband networks, we can assume that a communication delay is dominated by the required time for processing in each node; thus, we model the communication delay consistent with this consideration. This means that we should take into account the hop count, which is the number of nodes involved in messages, to determine the communication delay. Furthermore, because no agent has a global view, any agent can observe the communication delay as the difference from the time when a message is sent to the time when the response is received. Observed delay may change if their environment have temporally due to sudden traffic. However, we assume that such a delay vanished in a relatively a short time. Thus we think it does not significantly affect the proposed learning. If such a delay continues for a long time, agents identify that the environment moves to another state, and learns another policy adaptive to the new environment.

We define parameter $2d$ (≥ 0) for the round trip time (RTT) in the period between adjacent agents to express and control the degree of delay. Any manager can know whether the team formation request has succeeded or not after the observed response time, that is, from the time of requesting another agent to form or join a team for a set of subtasks W ($\subset T_i$) to the time of receiving notification of the completion of team formation or the acceptance to join a team. The response time observed by manager agent m_k plus the time required for execution of the allocated subtasks is expressed by $obs_W^{m_k}(d)$, which is a function of d. Note that $obs_W^{m_k}$ is the sum of twice the hop counts from m_k to the individuals and the maximum value of the times calculated by Eq. (1), which are the times required to process the allocated subtasks W in individuals under m_k. The response time at the *RM*, $obs_{T_i}^{RM}(d)$, is simply denoted by $obs_{T_i}(d)$ for $T_i \in \mathcal{T}$.

3.5 Team Formation

In each step, a finite multiset of tasks $\mathbf{T}$ whose elements are in $\mathcal{T}$ is given to the *RM*. Then, the *RM* immediately begins to process all the tasks in $\mathbf{T}$ one by one. For $T_i (\in \mathbf{T})$, the *RM* divides it into a number of subsets of subtasks and allocates each subset to the child agent placed directly below in accordance with the

policy derived by reinforcement learning (details are described in Section 4.1). When manager m_j receives a set of subtasks from its parent manager, it further divides the set into smaller subsets if needed and then allocates each of them to the child agent of m_j. Finally, individual l allocates its resources to the received subtasks so that they can be done within δ steps. At this time, we consider that l participates in the team that processes task T_i. Then, l reports that it has become a member of C_{T_i} with the value of the required time defined by Eq. (1). If one of the individuals cannot complete the given subtasks in less time than the deadline δ or one of the managers cannot find a child agent to allocate the subtasks to, the team formation for T_i fails. If all the subtasks of the task $T_i = \{t_1^i, t_2^i, \cdots, t_q^i\}$ are allocated to individuals, we can say that the team C_{T_i} is successfully formed. The set of teams is expressed by $C = \{C_T | T \in \mathbf{S}\}$, where $\mathbf{S} \subseteq \mathbf{T}$ is the set of tasks whose teams are successfully formed. Then, the team formation process is iterated. Note that an individual can join only one team at once. Thus, if a task is allocated when the individual belongs to another team, that task is discarded, and the team is not formed.

4. Proposed Learning and Allocation Methods

4.1 Reinforcement Learning based on Observed Delay

First, we describe how Q-values are calculated at any state of the agents. In the previous study [2], [12], the Q-value of action a at state s is adjusted by the following well-known iterative update function.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{3}$$

where s' is the next state after taking action a, α is the learning rate, and γ is the discount factor. The next action a', state s', and their associated values of $Q(s',a')$ are involved for all child agents, but it is not feasible that the agent always knows this information since it resides in lower-layer agents and changes frequently.

We propose modifying the reinforcement learning so that it does not use the information in neighboring agents. Instead, it uses the communication delay. In Eq. (3), the concept of delayed reward reflects the discount of the expected reward in the future in accordance with the number of actions until the reward is actually given. Thus, we use the communication delay, that is, delayed notifications of task completion:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[u_W \beta^{obs_W^m(d)} - Q(s,a)], \tag{4}$$

where β is the discount factor and m is the manager calculating the Q-values. u_W $(= \sum_{t_i \in W} u_i)$ is the reward when the team for all subtasks in W is successfully formed. Note that even if all subtasks in W $(\subset T_i)$ are allocated, all subtasks in T_i may not succeed in forming the team. In this case, no agents can earn the utilities, so all of the rewards are zero. Because the observed delay, $obs_W^m(d)$, means the lag time between the time sending a team formation request for W and the time of the arrival of the massage notifying team formation, it includes the time required to process the allocated subtasks in individuals. We should mention that the immediate reward, $r(s,a)$, in Eq. (3) is omitted in Eq. (4) because $r(s,a) \neq 0$ only when the agent is one of the lowest managers in the hierarchy.

4.2 State Representation

To improve the efficiency of Q-learning, we simplify the agents' states by abstraction. This abstraction is almost identical to that in Refs. [2] and [12]. We describe the state of an agent by the requested subtasks $W \subset T_i$. This means that managers make their decisions in accordance with the combination of the required resources and utilities; thus, the state is expressed by the pair $(u_W, r_1^W, r_2^W, \cdots, r_q^W)$, where u_W is the sum of the expected utilities of W and r_j^W is the j-th resource required for W. Then, the elements of the state are abstracted by using pairs of three-level ratings, *high*, *normal*, and *low*, for resources and utilities so as to reduce the number of states. For example, $(u_W, r_1^W, r_2^W) = (\text{high}, \text{normal}, \text{low})$ indicates the state in which the requested task requires a normal amount of the first resource and a low amount of the second resource $(q = 2)$, and the expected utility is high. Note that agents do not refer to the internal states and the associated policies in the upper- and lower-layer agents.

4.3 Task Allocation Process with Learning

First, we describe how an assigned task is divided into a number of subsets of subtasks and how they are allocated to agents (managers or individuals) at the next lower level of the organization by using the resource information given in advance. Then, we extend this method by incorporating it with the resource estimation method for unknown environments. This is explained in the next subsection.

When manager m is given a set of subtasks W by the parent or the environment, it identifies the learning state s in accordance with the required resources. Then, m selects action a by using the roulette selection in accordance with the third power of the Q-values for s. Action a indicates to which agent the subtasks are allocated. The selected agent is denoted by i. Manager m chooses a number of subtasks W_i from W, so that for $1 \leq k \leq m$, the following condition is satisfied.

$$D(i, W_i) \leq \delta \tag{5}$$

Therefore, W_i will be done within δ steps. Then, W is set to $W \setminus W_i$. If $W \neq \emptyset$, m selects the other actions for W and repeats this process until $W = \emptyset$.

After allocating W_i to the lower-level agent i, m waits for the completion or failure message from i. This waiting time corresponds to the observed delay $obs_{W_i}^m(d)$ that is used to revise the Q-values in Eq. (4).

4.4 Task Allocation Process with Learning and Resource Estimation

Since no agents have resource information in other agents in an unknown environment, manager agents use the estimated amount of resources that child agents manage or have. The estimation method is described in Section 4.5.

The learning process is almost identical to that described in the previous section except that they both sometimes allocate tasks randomly for exploring resources. Suppose that manager m has selected its child agent i. Then, m chooses a number of subtasks, W_i, from W by using the estimated amount of resources in i so that the following condition is satisfied for $1 \leq k \leq m$.

$$D(i, W_i) \leq 1. \quad (6)$$

Note that δ is not used in this condition. The details will be described in Section 4.5. However, when an individual has been requested the set of subtasks W that cannot complete within δ steps, it reports the failure to do W to the upper manager. Since the estimated amount of resources are small in the beginning of learning (actually, the initial value of the estimated resource is 0, as described below), no subtasks may be allocated, so in this case, $W_i = \emptyset$. Then, with the small probability ε or when $W_i = \emptyset$, m randomly selects a subtask $t \in W \setminus W_1$ and adds t to W_i.

This trial for exploration by adding one subtask is necessary to estimate the amount of resources. Then, W is set to $W \setminus W_i$, and this process is repeated until $W = \emptyset$. After all child agents are selected by roulette selection, some subtasks may still remain in W. In this case, each subtask in W is randomly added to W_i.

4.5 Resource Estimation during Task Allocation

Each agent has the estimated amount of resources of its child agents $i \in A$. This is denoted by $Er^i = \{er^i_1, er^i_2, \ldots, er^i_p\}$, and we initially define $er^i_k = 0$ for $1 \leq \forall k \leq p$. When the allocated subtasks, W_i, has been successfully allocated, that is, the message indicating successful task allocation has arrived, the estimated resources are updated as follows.

$$er^i_k = \max(er^i_k, r^{W_i}_k). \quad (7)$$

However, if it fails to allocate W_i, Er^i is left unchanged. This is because even if i manages or has sufficient resources for doing W_i, it may be busy doing the tasks allocated previously. In addition, because the amount of resources managed by agents decrease when links are eliminated, managers have to reduce their estimated resources. This is described in Section 4.6.2.

To be more precise, er^i is not the estimated amount of resources in i but is the estimated amount of resources required by subtasks that i can accept. Individuals accept the given subtasks if Eq. (5) is fulfilled. Therefore, i can accept them with a number of steps required for processing them. However, managers do not know the number of hop counts to individuals, so they cannot identify the actual processing time. Instead, we estimate the resources of tasks that can be accepted by each of the child agents. Thus, managers use Eq. (6) for allocations.

4.6 Reorganization Method

Along with the progress of learning, managers allocate the sets of subtasks to particular child managers and individuals. However, this emergent behavior through learning may develop biased task allocations; that is, both busy and non-busy agents coexist, resulting in inefficient utilization of resources under managers due to the mismatch between the patterns of resource usage and the deployment of individuals in the initial network. Therefore, a number of links to individuals are added or eliminated to resolve such situations.

4.6.1 Link Generation Process

The *link generation* process is initiated by root manager *RM*. The *RM* makes inquiries to all the individuals about the numbers of processed (sub)tasks along the hierarchical links. It then identifies the busiest individual l_{busy} and most inactive individual l_{inact} that reported the highest and lowest numbers of processed subtasks, respectively. Then, *RM* requests the manager m_{busy} that is located at the directly upper level of l_{busy} to make a new link to l_{inact}. When l_{busy} has multiple managers, *RM* selects the manager whose managing resources is the maximal among them because this manager may be allocated more subtasks by learning in its parent agent. If the link from m_{busy} to l_{inact} already exists, nothing occurs.

To avoid having too many parent agents, we also introduce the upper bound L, which is a positive integer and limits the number of links from managers to an individual. Hence, if l_{inact} already has L links, the link request is ignored. Note that when the link from m_{busy} to l_{busy} is added, m_{busy} has to prepare $Er^{l_{busy}}$ to estimate the amount of resources in l_{busy}.

4.6.2 Link Elimination

link elimination is an essential procedure for maintaining efficient organizational structure. Particularly, in the unknown environment, subtasks may irrelevantly be allocated in the beginning of resource estimation due to inaccurate information, so a number of meaningless links may be added. Thus, the link elimination is mandatory to adjust the structure of the agent network. In addition, after agents acquire accurate resource information, adding a new link is likely to result in another allocation pattern in the agent network. It is also probable that the distribution of service requests changes. Hence, a number of existing links between agents may fall into disuse. This prevents adding a really required link into the agent network.

For manager $\forall m$ at the directly upper level of individual l, l stores the numbers of tasks allocated by m for a certain period of time. This number is denoted by $N_l(m)$. Suppose that $N_l(m_{\min})$ is the lowest number and is much less than (K% of) the maximum number of $N_l(m)$. In this case, $m_{\min}$ seldom works as the manager of l. Then, if l has two or more links from managers, l eliminates the link from $m_{\min}$.

After a link is eliminated, $m_{\min}$ and its upper-level mangers must modify the amounts of the estimated resources as follows. First, $m_{\min}$ eliminates the resources estimated for l, $Er^l = \{er^l_i, \ldots, er^l_p\}$ and informs the parent agent that the managed resources have been lowered with the data of Er^l. This process is iterated until reaching the *RM*.

We call the method without the resource estimation proposed in Section 4.3 the *task allocation with link generation and elimination* (TALGE), and TALGE with the resource estimation proposed in this Section 4.4 is called the *extended TALGE* (eTALGE).

5. Experimental Evaluations

We conducted three experiments to evaluate our method. In the first and second experiments, we evaluate TALGE for the environment where all agents have resource information in other agents, and we compare its performance with those of the method proposed in Ref. [12] and a random method in which managers select the lower managers/individuals randomly. The method in Ref. [12] was called *adaptive team formation with reorganization* (ATFR) in their paper. We also measure the performance of TALGE without link elimination, so this is called *task allocation*

with (*only*) *link generation* (TALG), in order to investigate the effect of link elimination on the entire performance.

To simplify the graphs shown below, we omit the results of the method proposed in Ref. [2], but it is already known that ATFR is more efficient [12]. Note that the reorganization in ATFR is based on the amounts of unused resources for each individual, although our method is based on the number of processed (sub)tasks. The Q values in ATFR are calculated in accordance with Eq. (3), which assumes that the Q values in child agents are always available at no cost. We also set the number of resources p to 2 for the sake of simplicity.

In the third experiment, we compare the performance of the eTALGE with TALGE and TALG and eTALGE without link elimination (this is called the *extended TALG* (eTALG)).

5.1 Experimental Environment

In the first experiment (Exp. 1), to investigate whether effective task allocation can be realized, we prepared four types of individuals in accordance with their amounts of resources:

- Type 1: cr_0 and cr_1 are selected from the interval $[1, 5]$.
- Type 2: cr_0 and cr_1 are selected from the interval $[8, 12]$.
- Type 3: cr_0 is selected from $[1, 5]$, and cr_1 is done from $[10, 30]$.
- Type 4: cr_0 is selected from $[10, 30]$, and cr_1 is done from $[1, 5]$.

Note that cr_k is a positive integer and is selected randomly from the specified interval. The agents of types 1 and 2 have relatively balanced resources, although their amounts are different, and those of types 3 and 4 are biased to a specific resource.

Four types of subtasks are also defined in a similar way. For example, if the subtask $t = \langle u, r_0, r_1 \rangle$ is of type 1, then r_0 and r_1 are randomly selected from interval $[1, 5]$ when it is generated. We assume that utility u is identical to the sum of resources because the aim of our research is to allocate (sub)tasks effectively. This also means that we can simplify the expression of a learning state that depends on only the required resources $(r_1^W, r_2^W, \cdots, r_q^W)$.

In Exp. 1, the initial agent network structure was identical to that in Fig. 1. Four individuals of each type (a total of sixteen individual agents) were generated and deployed randomly to the leaf nodes. A task consisted of three to five subtasks (the numbers are randomly determined when it is generated), then two tasks were given to the *RM* every step. The communication delay parameter d was set to 1, so any communication between adjacent agents took one step. We set the deadline of processing δ to 10. To update Q values, we set $\alpha = 0.001$ and $\beta = 0.9$. The link generation process was invoked with a probability of 1/1,000, and the link elimination process was done with a probability of 1/3,000 in a step. Parameter L, which is the upper bound of the number of links from managers to each individual was set to 3. The experimental results shown below are the average values of 100 trials. In each trial, the earned utilities were stored every 100 steps and plotted in graphs. The experiments stopped at 60,000 steps because it seemed that the results became invariant after that.

5.2 Evaluation of Efficiency

Figure 2 plots the utilities earned by using TALG, TALGE, ATFR, and random methods [*1]. This figure shows that the utilities earned by TALG(E) are much larger than those earned by using the ATFR method and the random method. The utility earned at 80,000 steps by using TALG was approximately 30.3% higher and was approximately 38.2% higher than that of ATFR by using TALGE. Furthermore, TALGE outperformed TALG after sufficient learning time, although the learning speed in TALGE was slightly slow. Therefore, this result indicates that the combination of link generation and link elimination in accordance with the usages is required to improve the overall efficiency of the agent network.

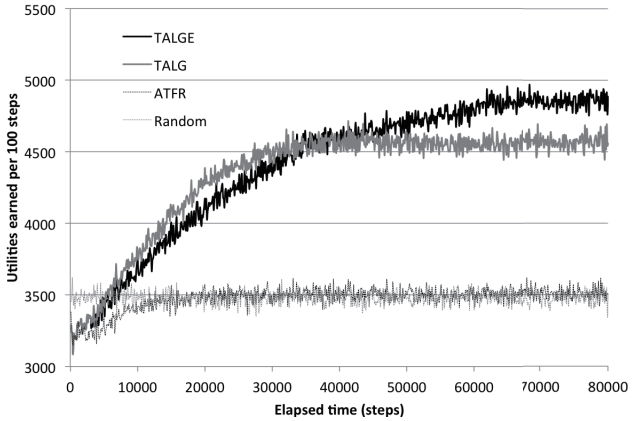

Fig. 2 Utilities earned every 100 steps.

Table 1 Structure of resulting agent network in TALG.

30,000 steps	80,000 steps
$m_7 \rightarrow (5, 15), 0, 3, 4, 14$	$m_7 \rightarrow (5, 15), 0, 3, 4, 8, 13, 14$
$m_8 \rightarrow (0, 14), 2, 7, 9, 12$	$m_8 \rightarrow (0, 14), 2, 5, 7, 8, 9, 10, 12$
$m_9 \rightarrow (11, 12), 1, 6, 8$	$m_9 \rightarrow (11, 12), 1, 2, 3, 6, 8$
$m_{10} \rightarrow (7, 9)$	$m_{10} \rightarrow (7, 9), 1, 10, 13, 15$
$m_{11} \rightarrow (3, 8), 11, 15$	$m_{11} \rightarrow (3, 8), 4, 6, 9, 11, 15$
$m_{12} \rightarrow (2, 4)$	$m_{12} \rightarrow (2, 4), 5, 6, 11, 15$
$m_{13} \rightarrow (6, 10), 5, 13$	$m_{13} \rightarrow (6, 10), 3, 5, 13$
$m_{14} \rightarrow (1, 13), 10$	$m_{14} \rightarrow (1, 13), 6, 7, 10, 11$

Table 2 Hierarchical structure of resulting agent network in TALGE.

30,000 steps	80,000 steps
$m_7 \rightarrow (5, 15)$	$m_7 \rightarrow (5, 15), 0, 2, 8, 11, 13, 14$
$m_8 \rightarrow (0, 14), 4, 6, 8, 10, 11$	$m_8 \rightarrow (0), 4, 6, 10, 11$
$m_9 \rightarrow (12), 1, 2, 3, 13$	$m_9 \rightarrow (12), 1, 3$
$m_{10} \rightarrow (7, 9)$	$m_{10} \rightarrow (7, 9)$
$m_{11} \rightarrow 6$	$m_{11} \rightarrow 6$
$m_{12} \rightarrow (4)$	$m_{12} \rightarrow 4$
$m_{13} \rightarrow 8, 12$	$m_{13} \rightarrow (10)$
$m_{14} \rightarrow (13)$	$m_{14} \rightarrow 8$

Tables 1 and **2** shows the resulting agents network after 30,000 and after 80,000 steps in TALG and TALGE in a certain trial among 100 trials. This trial was selected randomly and was not special. In these tables, m_i indicates the manager, and the integer n after the arrow indicates the individual (this is denoted by i_n), as shown in Fig. 1. The arrow indicates the direct links between them. For example, m_{11} has links to individuals, i_3, i_8, i_{11}

[*1] The experimental results shown in Fig. 2 are slightly different from those in Ref. [17]. In the previous experiments, we ignored the time for processing the given subtasks since it focused only on whether team formation succeeded or not. In this paper, we assume that the agent cannot join another team before completion of the given subtasks; this led to slightly lower performance in earning utilities.

Table 3 Numbers of links generated and eliminated in TALG(E) in Exp. 1.

Period	TALG	TALGE	
(step)	Generated Links	Generated Links	Eliminated Links
~10,000	6.48	7.2	4.04
~20,000	6.76	7.48	5.78
~30,000	6.38	7.22	5.88
~40,000	5.88	6.24	5.6
~50,000	4.74	6.36	5.4
~60,000	3.64	5.48	5.5
~70,000	0.3	0.64	0
~80,000	0	0	0
Total	34.18	40.62	32.2

and, i_{15} at the 30,000th step in Table 1. The numbers in parentheses mean the individuals that exist in the initial states. **Table 3** lists the average number of links generated and eliminated every 10,000 steps in TALG(E). Note that the link elimination process is called at a probability of 1/3,000, but it is possible that multiple links are eliminated simultaneously. However, only one link is added in the link generation process.

In TALGE, the link generation and elimination still worked till 60,000 steps (see Table 3). Because the learned data for the eliminated links were also eliminated, its learning was slower. However, links were only added in TALG, and the learned results were still usable (of course, further learning was required), although some links became useless. Thus, the learning in TALG converged faster before 30,000 steps as shown in Fig. 2. However, the unnecessary links prevented converging with a better control by learning; actually, the efficiency at the end in TALGE was higher. From Tables 1 and 2, the numbers of links to individuals at the 80,000th step were 54 in TALG and 22 in TALGE (initially, it was 16). Another observation is that some eliminated links were sometimes restored, as shown in Table 2. For example, the links from m_{13} to i_6 and i_{10} were eliminated at the 30,000th step, but the link from m_{10} to i_{10} came back at the 80,000th step. However, this kind of link restoration (or oscillation) did not occur after this; actually, links became stable after 70,000 steps.

5.3 Environmental Change

The distributions of types of required tasks may change in Internet services. The purpose of the second experiment (Exp. 2) was to examine whether our proposed method can adapt to the change in the required task types and whether the combination of link generation and elimination processes can contribute to the overall performance of the agent network. For this purpose, we considered the situations in which the tasks whose required resources are biased are given to the agent network whose individual agents also have the biased resources. Then, we tried to change the pattern of required tasks over time.

In Exp. 2, we prepared the following types of individuals:

- Type 3: (aforementioned)
- Type 4: (aforementioned)
- Type 5: cr_0 is selected from $[1,5]$, and cr_1 is done from $[8,12]$.
- Type 6: cr_0 is selected from $[8,12]$, and cr_1 is done from $[1,5]$.

As with Exp. 1, four agents of each type were generated and randomly placed on the leaves of the network shown in Fig. 1.

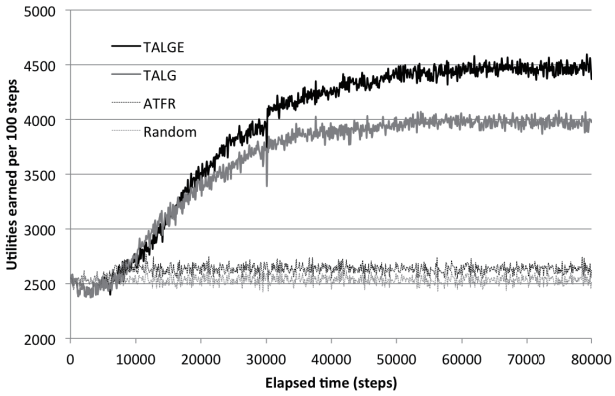

Fig. 3 Utilities earned in changing environment.

Table 4 Structures of agent networks in TALG (Exp. 2).

30,000 steps	80,000 steps
$m_7 \to (5,15), 2, 11$	$m_7 \to (5,15), 2, 11$
$m_8 \to (0,14), 3, 4, 6, 7, 8, 10, 12, 13$	$m_8 \to (0,14), 3, 4, 6, 7, 8, 10, 12, 13$
$m_9 \to (11,12), 1, 2, 6, 9, 13, 15$	$m_9 \to (11,12), 1, 2, 6, 9, 13, 15$
$m_{10} \to (7,9), 1, 11$	$m_{10} \to (7,9), 1, 11$
$m_{11} \to (3,8)$	$m_{11} \to (3,8), 1$
$m_{12} \to (2,4), 1, 11$	$m_{12} \to (2,4), 1, 9, 11$
$m_{13} \to (6,10), 11$	$m_{13} \to (6,10), 1, 11$
$m_{14} \to (1,13), 0, 10, 11$	$m_{14} \to (1,13), 0, 10, 11$

Table 5 Structures of agent networks in TALGE (Exp. 2).

30,000 steps	80,000 steps
$m_7 \to (5,15)$	$m_7 \to (5,15)$
$m_8 \to (0,14), 1, 2, 3, 8, 12, 13$	$m_8 \to (0,14), 1, 2, 3, 8, 12, 13$
$m_9 \to (11), 4, 7, 9, 10$	$m_9 \to (11), 4, 6, 7, 9, 10, 15$
$m_{10} \to (7,9), 6, 10$	$m_{10} \to 6$
$m_{11} \to 4, 13$	$m_{11} \to 4$
$m_{12} \to (4)$	$m_{12} \to (4)$
$m_{13} \to (10)$	$m_{13} \to (6)$
$m_{14} \to 4$	$m_{14} \to 4$

Similarly, we defined four types (Types 3 to 6) of subtasks, and two tasks consisting of 3 to 5 subtasks were generated every step. However, all subtasks belong to Type 3 or 5 during 0 to 30,000 steps and after 30,000 steps, all subtasks belonged to Types 4 or 6. Other parameters were identical to those in Exp. 1.

Figure 3 shows the utilities earned over time in the changing environment. **Tables 4** and **5** list the links between managers and individuals at the 30,000th and 80,000th steps. Figure 3 indicates that although the utilities slightly lowered when the environment changed, TALG and TALGE could quickly recover performance.

We see from Tables 4 and 5 that the organizational structures changed more in TALGE than in TALG. Elimination of less used links can encourage the reorganization of the agent network to adapt to the new situation. However, having too many links in TALG prevents the reorganization process. In fact, if we compare the ratios of earned utilities of TALG with those of TALGE in Exps. 1 and 2, TALGE exhibited a ratio of earned utilities approximately 6.0% higher than did TALG in Exp. 1 but 12.9% higher in Exp. 2. This suggests that the combination of link elimination and generation drove reorganization in accordance with the resources that individuals have. We believe that this is an important characteristic because the actual systems in an open environment are constantly changing.

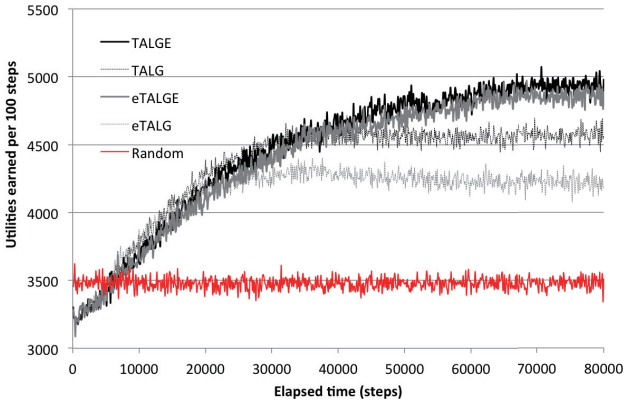

Fig. 4 Utilities earned every 100 steps.

Table 6 Hierarchical structures in eTALG.

30,000 steps	80,000 steps
$m_7 \rightarrow (5, 15), 0, 1, 2, 4, 6$ 10, 14	$m_7 \rightarrow (5, 15), 0, 1, 2, 4, 6, 8, 9, 10$ 12, 13, 14
$m_8 \rightarrow (0, 14), 4, 5, 6, 7, 13$	$m_8 \rightarrow (0, 14), 4, 5, 6, 7, 10, 13$
$m_9 \rightarrow (11, 12), 3, 8$	$m_9 \rightarrow (11, 12), 2, 3, 7, 8, 9$
$m_{10} \rightarrow (7, 9), 2, 10$	$m_{10} \rightarrow (7, 9), 1, 2, 3, 5, 6, 10, 11, 12$
$m_{11} \rightarrow (3, 8)$	$m_{11} \rightarrow (3, 8), 15$
$m_{12} \rightarrow (2, 4), 13$	$m_{12} \rightarrow (2, 4), 0, 13, 14$
$m_{13} \rightarrow (6, 10), 1, 13, 15$	$m_{13} \rightarrow (6, 10), 1, 9, 13, 15$
$m_{14} \rightarrow (1, 13)$	$m_{14} \rightarrow (1, 13)$

5.4 Task Allocation with Resource Estimation

In the third experiment (Exp. 3), we evaluated eTALG(E), which is the learning-based task allocation with reorganization and resource estimation. The experimental settings were identical to those in Exp. 1. Parameter ε used in the resource estimation was set to 0.01, and parameter K for link elimination defined in Section 4.6.2 was set to 10 (%). The results are shown in **Fig. 4**, which plots the utilities earned by using eTALG(E), TALG(E), and the random method.

This figure shows that the utility earned at 80,000 steps by using eTALG was 8.09% lower than that by using TALG, and that by using eTALGE was 1.59% lower than that using TALGE, although those earned by eTALG and eTALGE were much larger than those earned with the random method. Furthermore, eTALG seemed to become gradually lower after 30,000 steps.

The slightly lower performance of eTALGE originated from the resource estimation method, but it was slightly higher than we expected. Parameter ε was 0.01, but all agents in the hierarchical structure tried to allocate one extra task with this probability for the sake of resource estimation. If the depth of the hierarchy is 4 and each agent may have multiple child agents, we can easily expect that the possibility of failure of team formation will increase about 5–10% for the exploration of learning after sufficient steps. Furthermore, managers are likely to select child agents with higher Q values to add an extra subtask. This may result in the failure of task allocation, so their Q values were made to be lowered, although the influence on efficiency was not high. We believe that if we lower ε in accordance with the time steps, the performance deterioration may become smaller.

Tables 6 and **7** express the resulting agent network after 30,000 and 80,000 steps with eTALG(E) in a certain trial among 100 trials. If we compare Table 6 with Table 1, it is clear that eTALG generated many more links to individuals than did TALG. However, as shown in Table 7, the numbers of links were quite low in eTALGE because eTALGE could eliminate unnecessary links that might be generated in the beginning of the experimental steps.

Table 7 Hierarchical structures in eTALGE.

30,000 steps	80,000 steps
$m_7 \rightarrow (5, 15), 3, 6, 10$	$m_7 \rightarrow (5, 15), 1, 3, 4, 6, 10$
$m_8 \rightarrow (0, 14), 2, 13$	$m_8 \rightarrow (0, 14), 2, 13$
$m_9 \rightarrow (11, 12), 4$	$m_9 \rightarrow (11, 12), 1, 8$
$m_{10} \rightarrow (7, 9)$	$m_{10} \rightarrow (7, 9)$
$m_{11} \rightarrow (8)$	$m_{11} \rightarrow 4$
$m_{12} \rightarrow 1, 8$	$m_{12} \rightarrow 1, 3$
$m_{13} \rightarrow (10)$	$m_{13} \rightarrow 13$
$m_{14} \rightarrow 0$	$m_{14} \rightarrow (1, 13)$

Table 8 Numbers of links generated and eliminated in eTALG and eTALGE.

Period	eTALG	eTALGE	
(step)	Generated Links	Generated Links	Eliminated Links
~10,000	7.78	7.96	4.1
~20,000	7.9	7.46	5.32
~30,000	6.76	6.62	5.54
~40,000	6.36	6.28	5.32
~50,000	5.58	5.88	5.4
~60,000	5.62	5.58	5.0
~70,000	0.56	0.48	0
~80,000	0	0	0
Total	40.56	40.26	30.68

To more clearly identify how actively the agent networks are reorganized, we retained the numbers of links generated and eliminated per 10,000 steps; these data are shown in **Table 8**. If we compare it with Table 3, these tables show that the numbers of links generated with TALG and eTALG were 34.18 and 40.56. Because, at first, the estimated resources were not accurate and revised frequently, unnecessary links were generated. Furthermore, since no links were eliminated with eTALG and TALG, they still remained at the 80,000th step, and this lowered the efficiency of the task allocation. In comparison, eTALGE and TALGE generated and eliminated similar numbers of links; the difference was approximately ten on average. Particularly, we have to note that, at first, the estimated resources were inaccurate and generated unnecessary links in unknown environments, but eTALGE was able to eliminate them quickly after the accuracy was improved. This restricted the redundant links. Actually, only seven links increased in eTALGE from the initial state to the final state at 80,000th step in the trial instance as shown in Table 7.

6. Discussion

Our experimental results indicate that the ATFR method proposed in Ref. [12] is less effective for team formation in our experimental setup. This is the problem of ATFR mentioned in Section 1. In ATFR, the busiest and most inactive individuals are identified on the basis of the amount of unused resources in a certain period of time, although TALG and TAGLE refer to the numbers of processed tasks. This inefficiency was caused by two facts. First, any individual can belong to only one team at the same time (this assumption is quite common in team formation problems). Second, because the number of tasks an individual

takes simultaneously is limited due to the first fact, the amount of unused resources in an individual with high cr_k is likely to be large, so it is identified as an inactive agent. Therefore, even when a new link is added, the individual may not be able to accept the task because it already has; thus, the team formation fails. In addition, the individual with high cr_k is frequently identified as still inactive. Thus, no new link is generated because the link already exists. This resulted in less and slow reorganization.

Figures 2 and 3 show that the learning speed in TALGE was lower than that in TALG. We think that the link elimination process stimulates the generation of new links and vice versa. Therefore, the organizational structure frequently changes in the first half of the learning time, and this change makes the learning results so far partially invalid. Therefore, the active organizational change results in more effective agent networks after all.

TALGE provides effective team formation, and clearly link elimination contributes to this effectiveness. We can consider a number of reasons. First, if managers have many links, the number of possible actions also becomes large, so the learning efficiency becomes low. Second, Q-values of important links may be made lower by unnecessary links, which have relatively lower Q values. Suppose that a manager has a link whose Q value is low. Though the probability is low, tasks may be allocated to individual i via this link. Agent i also has another link whose Q value is relatively high and receives a task via this link. However, i cannot accept this second task because it already has another task. In this case, the Q value of the important link lowers; thus, the learning is confused by unnecessary links. Finally, of course, the number of links from a manager is limited by the upper bound L. These facts prevent adding a new link in TALG. Actually, Table 4 indicates that reorganization was performed only a few times in TALG. However, TALGE, which has the link elimination process, can improve the overall effectiveness of the agent network.

The utilities earned for eTALG gradually lowered after 30,000 steps, as shown in Fig. 4. We can consider a number of reasons for this phenomenon. First, if managers have many links, the number of possible actions also becomes large, so the learning efficiency becomes low. Second, Q-values of important links may be made to be lowered by unnecessary links which have relatively lower Q values. For example, let us assume that a manager has a link to individual l, whose Q value is low. Although the probability is not high, it sometimes allocates subtasks to l via this link for the exploration of learning. However, l also has another high Q-value (very important) link. If l received subtasks via the important link after receiving subtasks via the unnecessary link, l cannot accept the task request, and thus, the Q value of the important link will lower. This confuses the learning process. This suggests that the link elimination was essential for improving the learning results

Finally, we have to say that the influence on the resource estimation by link elimination was insignificant. Table 8 shows that no links were eliminated after 60,000 steps, and this was identical to the cases with TALGE shown in Table 3.

We think that our proposed method is applicable to a number of systems with hierarchical structures. First, Abdallah and Lesser proposed the distributed information gathering system in the grid-computing environment using their methods [3]. We think our method can also be used in their system and the similar applications. Other systems to which our method may be applicable are, for example, the smart sensor network system in which a number of Web-based sensor nodes, field servers, and manager nodes are hierarchically connected to provide concrete views from field sensor data [7], the hierarchical intrusion detection system for secure network environments including ad hoc networks [6], and the hierarchical middleware compositions on a multi-domain platform in grid and cloud computing environments [13].

7. Conclusion

In this paper, we proposed two learning-based task allocation methods for use in tree-structure agent networks. The first one is TALGE withch is task allocation with reorganization method in known environments, meaning that all resource information in other agents is available. We also extend this method as eTALGE for unknown environments by incorporating it with resource estimation. We experimentally showed that these methods outperformed the conventional method in this type of network.

Some studies, such as Refs. [1], [5], assumed that agent networks do not have a hierarchical structure. We think this assumption is important because the structures of a number of existing systems are not hierarchical. For example, it is known that the Internet has a scale-free structure with high cluster coefficients [4]. We will extend our method for other agent network structures. In addition, we need to apply our method to a large-scale MAS.

References

[1] Abdallah, S. and Lesser, V.: Multiagent Reinforcement Learning and Self-Organization in a Network of Agents, *Proc. 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Honolulu, IFAAMAS, pp.172–179 (2007).

[2] Abdallah, S. and Lesser, V.R.: Organization-Based Cooperative Coalition Formation, *2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (*IAT 2004*), pp.162–168, IEEE Computer Society (2004).

[3] Abdallah, S., Zhang, H. and Lesser, V.: The Role of an Agent Organization in a Grid Computing Environment, *Proc. 14th International Conference on Automated Planning and Scheduling* (*ICAPS 2004*), *Workshop on Planning and Scheduling for Web and Grid Services*, Whistler, Canada, pp.1–8 (2004).

[4] Easley, D. and Kleinberg, J.: *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*, Cambridge University Press (2010).

[5] Eberling, M. and Büning, H.K.: Self-Adaptation Strategies to Favor Cooperation, *Proc. 4th KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications, Part I* (*KES-AMSTA '10*), pp.223–232, Springer-Verlag, Berlin, Heidelberg (2010) (online), available from ⟨http://dl.acm.org/citation.cfm?id=1883838.1883867⟩.

[6] Fu, P., Zhang, D., Wang, L. and Duan, Z.: Intelligent Hierarchical Intrusion Detection System for Secure Wireless Ad Hoc Network, *Advances in Neural Networks - ISNN 2005*, Wang, J., Liao, X.-F. and Yi, Z. (Eds.), Lecture Notes in Computer Science, Vol.3498, pp.482–487, Springer, Berlin, Heidelberg (online), DOI: 10.1007/11427469.78 (2005).

[7] Fukatsu, T., Kiura, T., Tanaka, K. and Hirafuji, M.: Hierarchical Agent System for Web-Based Sensor Network, *International Symposium on Applications and the Internet Workshops* (*SAINT Workshops 2007*), p.77 (online), DOI: 10.1109/SAINT-W.2007.50 (2007).

[8] Genin, T. and Aknine, S.: Coalition Formation Strategies for Self-Interested Agents in Task Oriented Domains, *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol.2, pp.205–212 (2010).

[9] Hamada, D. and Sugawara, T.: Autonomous decision on team roles for

efficient team formation by parameter learning and its evaluation, *International Journal of Intelligent Decision Technologies*, Vol.7, No.3, pp.163–174 (online), DOI: 10.3233/IDT-130160 (2013).
[10] Huhns, M.N., Singh, M.P., Burstein, M., Decker, K., Durfee, E., Finin, T., Gasser, L., Goradia, H., Jennings, N., Lakkaraju, K., Nakashima, H., Parunak, V., Rosenschein, J.S., Ruvinsky, A., Sukthankar, G., Swarup, S., Sycara, K., Tambe, M., Wagner, T. and Zavala, L.: Research Directions for Service-Oriented Multiagent Systems, *IEEE Internet Computing*, Vol.9, pp.65–70 (online), DOI: http://doi.ieeecomputersociety.org/10.1109/MIC.2005.132 (2005).
[11] Jurasovic, K., Kusek, M. and Jezic, G.: Team Formation and Optimization for Service Provisioning, *Proc. 4th KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications, Part I* (*KES-AMSTA '10*), *LNCS 6070*, pp.213–222, Springer-Verlag, Berlin, Heidelberg (2010) (online), available from ⟨http://dl.acm.org/citation.cfm?id=1883838.1883866⟩.
[12] Katayanagi, R. and Sugawara, T.: Efficient Team Formation Based on Learning and Reorganization and Influence of Communication Delay, *International Conference on Computer and Information Technology*, Vol.0, pp.563–570 (online), DOI: http://doi.ieeecomputersociety.org/10.1109/CIT.2011.18 (2010).
[13] Mathias, E. and Baude, F.: Multi-domain Grid/Cloud Computing Through a Hierarchical Component-based Middleware, *Proc. 8th International Workshop on Middleware for Grids, Clouds and e-Science* (*MGC 2010*), pp.2:1–2:7, ACM (online), DOI: 10.1145/1890799.1890801 (2010).
[14] Sheholy, O. and Kraus, S.: Methods for Task Allocation via Agent Coalition Formation, *Journal of Artificial Intelligence*, Vol.101, pp.165–200 (1998).
[15] Shehory, O. and Kraus, S.: Feasible Formation of Coalitions among Autonomous Agents in Nonsuperadditive Environments, *Computational Intelligence*, Vol.15, No.3, pp.218–251 (1999).
[16] Stein, S., Gerding, E. and Jennings, N.R.: Optimal Task Migration in Service-Oriented Systems: Algorithms and Mechanisms, *Proc. 19th European Conference on Artificial Intelligence*, Amsterdam, The Netherlands, IOS Press, pp.73–78 (2010) (online), available from ⟨http://dl.acm.org/citation.cfm?id=1860967.1860983⟩.
[17] Urakawa, K. and Sugawara, T.: Reorganization of Agent Networks with Reinforcement Learning Based on Communication Delay, *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology* (*WI-IAT 2012*), Vol.II, pp.324–331 (online), DOI: 10.1109/WI-IAT.2012.105 (2012).
[18] Wooldridge, M.: *An Introduction to MultiAgent Systems* (*2nd Edition*), John Wiley (2009).
[19] Yang, J. and Luo, Z.: Coalition formation mechanism in multi-agent systems based on genetic algorithms, *Appl. Soft Comput.*, Vol.7, No.2, pp.561–568 (online), DOI: 10.1016/j.asoc.2006.04.004 (2007).

Kazuki Urakawa is a graduated student of Department of Computer Science and Engineering, Graduate School of Fundamental Science and Engineering, Waseda University, Japan, since April 2012. He received his B.E. degree and M.E., in 2012 and 2014. His research interests are in multi-agent systems and machine learning.

Toshiharu Sugawara is a professor of Department of Computer Science and Engineering, Waseda University, Japan, since April 2007. He received his B.S. and M.S. degree in Mathematics, 1980 and 1982, respectively, and a Ph.D, 1992, from Waseda University. In 1982, he joined Basic Research Laboratories, Nippon Telegraph and Telephone Corporation. From 1992 to 1993, he was a visiting researcher in Department of Computer Science, University of Massachusetts at Amherst, USA. His research interests include multi-agent systems, distributed artificial intelligence, machine learning, internetworking, and network management. He is a member of IEEE, ACM, AAAI, Internet Society, IPSJ, Japan Society for Software Science and Technology (JSSST), and Japanese Society of Artificial Intelligence (JSAI).