Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

Coordinated Area Partitioning Method by Autonomous Agents for Continuous Cooperative Tasks

Vourchteang Sea[1,a)] Chihiro Kato[1,b)] Toshiharu Sugawara[1,c)]

Received: April 30, 2016, Accepted: October 4, 2016

Abstract: We describe a method for decentralized task/area partitioning for coordination in cleaning/sweeping domains with learning to identify the easy-to-dirty areas. Ongoing advances in computer science and robotics have led to applications for covering large areas that require coordinated tasks by multiple control programs including robots. Our study aims at coordination and cooperation by multiple agents, and we discuss it using an example of the cleaning tasks to be performed by multiple agents with potentially different performances and capabilities. We then developed a method for partitioning the target area on the basis of their performances in order to improve the overall efficiency through their balanced collective efforts. Agents, i.e., software for controlling devices and robots, autonomously decide in a cooperative manner how the task/area is partitioned by taking into account the characteristics of the environment and the differences in agents' software capability and hardware performance. During this partitioning process, agents also learn the locations of obstacles and the probabilities of dirt accumulation that express what areas are easy to be dirty. Experimental evaluation showed that even if the agents use different algorithms or have the batteries with different capacities resulting in different performances, and even if the environment is not uniform such as different locations of easy-to-dirty areas and obstacles, the proposed method can adaptively partition the task/area among the agents with the learning of the probabilities of dirt accumulations. Thus, agents with the proposed method can keep the area clean effectively and evenly.

Keywords: coordination, area partitioning, cooperation, multi-agent systems, autonomous task division, division of labor, continuous sweeping, security surveillance

1. Introduction

Continuing advances in computers, devices and robotic technologies have led to applications combining computers, sensors and robots that perform tasks normally done by people. These tasks will cover a wide range; for example, (1) computer systems will monitor patient conditions in hospitals and resident activities in nursing homes, and (2) robots operating in public spaces, in nuclear plants or on planets will explore, maintain security/safety or clean the environment. Since these applications involve the monitoring of a huge amount of data in realtime and/or operating in wide areas, a single robot/computer cannot handle the entire space/data. There is thus a strong requirement for coordination and cooperation using multiple agents each of which is software for controlling a system/robot. We focused on their application for cleaning and patrolling tasks in a given environment.

The real-world environments where agents operate are diverse, so it is almost impossible to design a system by completely anticipating the environmental characteristics in the design stage. For example, in the cleaning task, there are a number of locations where dirt may tend to accumulate, and these locations depend on many factors such as the shape of the environment and the locations of furniture and fixtures. Similarly, in security applications, the locations near entrances, near windows and around safes should be kept more secure than other locations. This means that agents for cleaning or security must visit the locations in the given area with different frequencies in accordance with the characteristics of the environment. Furthermore, the agents may be most-advanced or old models and may have been developed by different makers; this means that they have different hardware/software and thus exhibit different levels of performance. The agents must therefore work cooperatively by taking into account these differences in order to efficiently perform the tasks in a more balanced manner.

There are two conventional approaches to implementing coordinated and cooperative patrolling activities for cleaning and security tasks. The first approach is for agents to share the working area and clean it in a coordinated manner. For example, agents could patrol the area by using different cleaning algorithms or different visitation cycles to uniformly cover the entire area [5], [13], [21]. Another strategy for this approach is for the agents to move around the area in formation (e.g., Refs. [1], [7], [15]). However, in these approaches, an agent's strategy affects the other's, and this interaction makes cooperation complex. The second approach is to divide the area into a number of subareas and make each agent responsible for a different subarea [2], [9]. However, performing fair division is not trivial in the latter approach; if the characteristics of the area are not uniform and the agents have different capabilities, equal-size

[1] Department of Computer Science and Communications Engineering, Waseda University, Shinjuku, Tokyo 169–8555, Japan
[a)] sea.vourchteang@isl.cs.waseda.ac.jp
[b)] c.kato@isl.cs.waseda.ac.jp
[c)] sugawara@waseda.jp

subareas are inappropriate.

We thus propose a new approach in which the agents autonomously decide their responsible subareas so as to divide the task fairly on the basis of their capabilities and the characteristics of each subarea in this paper[*1]. If new agents are added, agents autonomously reconfigure their subareas through coordinated interaction over time. The key idea is that each agent monitors the locations it recently visited and calculates its *expansion power*, which is based on the degree of task completion, such as the the expected amount of dirt remaining in its subarea in cleaning tasks and the number of important locations to keep them secure. It then negotiates with adjacent agents to determine which agents should expand their current subarea so as to balance the cleanliness levels of the area. However, it is difficult to identify in advance which agents that have different hardware/software capabilities perform better in the environment and what areas are easy to be dirty. Thus, our study aims, using a cleaning task application, at the proposal of coordination method for area partitioning without this kind of knowledge.

We already proposed the method, called *performance-based partitioning* (PBP) along this line in Ref. [12], but it did not examine whether their method could reflect the differences between agents' algorithms into the area partitioning. Furthermore, it assumed that knowledge about what areas are easy to be dirty was given to all agents. However, providing this knowledge in advance is difficult because it depends on many factors such as locations of objects, intake/exhaust vents, doors and windows in the environments. Furthermore, environments may have a number of obstacles for robots. Because the locations and shapes of obstacles differ in individual environments and may change, it is not easy to accurately specify their information in advance. Thus, we excluded this assumption and extended the previous method by adding (1) the learning capability to agents for identifying easy-to-dirty areas and (2) the function to find and maintain the locations of obstacles through their operations. We also show the detailed results with extensive experiments.

The structure of this paper is as follows. We describe related work in the next section and our models for agent, area and dirt accumulation in Section 3. Section 4 explains the proposed method with which agents autonomously determine the area they should clean by taking into account the environmental characteristics and the differences in their exploration algorithms, without knowledge about what areas are easy to accumulate dirt. Section 5 presents experimental results indicating that agents with the proposed method achieves more efficient and more balanced area partitioning than the conventional method in a variety of environments. We also demonstrate that they can learn what areas are easy-to-dirty while identify the locations and shapes of obstacles. We then discuss our results in Section 6 and finally state our conclusion.

2. Related Work

There have been a number of studies applying agents, which are software programs for autonomously generating robot activities, to cleaning and patrolling problems using single or multiple robots. For example, Ahmadi and Stone [2] proposed a method in which an agent moves around in search of events that occur with different probability. However, that study did not address collaborative movement with multiple agents. Kurabayashi et al. [13] proposed a centralized off-line method in which a single server generates the entire route for a sweeping task. The route is then divided into fragments and allocated to individual agents to minimize the working time. Yoneda et al. [21] proposed a distributed method in which agents autonomously decide their search/exploration strategies in a multi-robot sweeping problem using reinforcement learning. Sampaio et al. [17] proposed the gravity-based model in which the locals that were not visited for a long time have the stronger gravity, and thus, agents tend to visit such locations for uniform patrolling. Unlike our method, these methods are based on the assumption that agents traverse a shared area along different routes or with different exploration algorithms.

We can formalize the patrolling problem from more theoretical perspective. For example, Chevaleyre et al. [5] formalized the patrolling problem as a traveling salesman problem with multiple agents and then compared the number of cyclic routes and route division methods from the viewpoint of minimal route length. Elmaliach et al. [8] proposed an algorithm that finds the shortest Hamiltoniam cycle in grids that are used to patrol in the areas and to spread the agents there evenly. These methods also assume that robots move in the shared areas. Furthermore, most of these studies did not consider the case of agents visiting the locations at different frequencies.

Another approach is to partition the area into subareas so that agents can divide the labor. Ahmadi and Stone [3] extended their previous work [2] to cases with multiple patrolling agents. Their extended method segments the region of responsibility for individual agents, which exchange boundary information, and agents visiting a boundary region more frequently tend to take charge of the region. Volonoi-based techniques are also another methods that do not require graph descriptions of the environments, such as Refs. [4], [6], [18]. However, these require computational costs that limit their applicability.

Bio-inspired computation models are also used to cover the areas. Ranjbar-Sahraei et al. [16] introduced the indirect communication using pheromone-based stigmatic communications to identify the regions that should be covered. McCaffery [14] proposed the graph partitioning algorithm using the foraging behaviors. The resulting subgraphs are allocated to agents so that they cover the whole environment. Elor [9] proposed a segmentation method for covering a region by integrating the ant pheromone and balloon models: a region is segmented into subregions that are individually assigned to different agents. Each subregion is considered to be a balloon, the pressure of which represents the size of the subregion. The pressure values are then indirectly exchanged using the pheromone communication model. However, the differences in agent performance and the environmental characteristics are not considered in these methods, thus the region is likely to be divided into equal-size subregions. Furthermore, the implementation of pheromone communications in decentralized

[*1] This paper is an extended version of our conference papers [12] and [19].

multiple-robot applications is not trivial. Kato and Sugawara [12] proposed the method *performance-based partitioning* (PBP), for partitioning a given area so that agents keep the environment evenly clean by performing the cleaning task in a balanced manner by taking into account these differences although it is not a bio-inspired approach. However, unlike our method, it assumes that agents have knowledge about what areas are easy to be dirty in the environment, but providing this knowledge in advance is difficult because it depends on many factors. Sea and Sugawara [19] proposed the PBP with the learning of dirty places and indicated that it exhibited the good performance in the environments with obstacles. However, these did not sufficiently evaluate whether their methods could reflect the differences in agents' hardware/software into the area partitioning [12], [19].

3. Model and Problem Definition

3.1 Models of Agent and Environment

An agent here is a control program installed on a portable cleaner robot capable of autonomously deciding its actions and sending/receiving messages. We assume that it has a map (graph) of the area. Such information may often be unknown. However, many algorithms for creating a map, identifying agent locations, and avoiding collisions have been proposed (Refs. [10], [11], [20]). Because we focus on autonomous learning for area partitioning for balanced work division, we use this assumption here.

Let $A = \{1, \ldots, n\}$ be a set of agents. The area in which the agents move around is described by a connected graph with obstacles, $G = (V^+, E, O)$, where V^+ and E are sets of nodes and edges, and O ($\subset V^+$) is the set of obstacles. A node in O is called the *obstacle node*. In general, we assume that a number of obstacles, $\{O_i \mid O_i \subset V^+$ for $1 \leq i \leq k$, and O is the connected set$\}$ exist in the environment and we define $O = O_1 \cup \ldots \cup O_k$. The edge that connects nodes $v_i, v_j \in V^+$ is expressed by $e_{i,j}$. We introduce a discrete time with a unit called a *tick*. An agent moves between nodes in $V = V^+ \setminus O$ and cleans each node it visits. Without the loss of generality, we can assume that the length of an edge in E is one (by adding dummy nodes if necessary), so any agent can move from a node to another along an edge and then clean the visited node in one tick. However, it cannot move to any node in O. We assume that $V \setminus O$ is connected, i.e., for $\forall v, w \in V$, at least one path from v to w consisting of only non obstacle nodes exists.

3.2 Battery Consumption and Charge

Let positive integers $B^i_{\max}$ and $b^i(t)$ be the maximal capacity and the remaining power of the battery in agent i at time t. Agent i consumes a constant amount of power per tick, B^i_{drain} when it moves around. Thus, $b^i(t)$ is updated using

$$b^i(t+1) \leftarrow b^i(t) - B^i_{drain} \tag{1}$$

every tick. Agent i can thus continuously operate at most $\lfloor B^i_{\max}/B^i_{drain} \rfloor$ ticks, which is called the *maximum running time* and is denoted by M_i. Agent i charges its battery at its charging base, $v^i_{base} \in V$. The required time for a full charge starting from time t, $T^i_{charge}(t)$, is proportional to the battery power consumed:

$$T^i_{charge}(t) = k^i_{charge}(B^i_{\max} - b^i(t)), \tag{2}$$

where k^i_{charge} (> 0) is the proportionality factor indicating the speed of charge. Agents with a full battery start to move around and perform cleaning; they return to their bases and recharge their batteries. Agents iterate this *cleaning cycle* to keep their assigned area clean.

For any node $v \in V^i_t$, i calculates the *potential*, which is the minimal capacity of battery required to return to i's charging base v^i_{base}. The potential of v for i is denoted by $\mathcal{P}(v)^i$ and calculated using

$$\mathcal{P}(v)^i = d(v, v^i_{base}) \cdot B^i_{drain}, \tag{3}$$

where $d(v, v')$ is the shortest path length. Because agents know G, they can identify the shortest path using Dijkstra's algorithm or the A* algorithm. We say that it is *safe* for i to move to neighbor node v at time t if

$$b^i(t) \geq \mathcal{P}(v)^i + B^i_{drain}, \tag{4}$$

where v^i_t is the node where i is currently located. Agent i moves to only safe nodes; if the next node is not, i returns to v^i_{base} along the shortest path and then recharges.

3.3 Model of Dirt Accumulation

We represent the degree to which dirt is easy to be accumulated per tick at node $v \in V$. The *amount of accumulated dirt* at v at time t, $L_t(v)$, is initially defined as $L_0(v) = 0$ for $\forall v \in V$ and is updated by

$$L_t(v) \leftarrow \begin{cases} L_{t-1}(v) + 1 & \text{with probability } p_v \text{ (a piece of} \\ & \text{dirt is accumulated at } t) \\ L_{t-1}(v) & \text{otherwise,} \end{cases} \tag{5}$$

where event probability p_v ($0 \leq p_v \leq 1$) is called the *dirt accumulation probability* (DAP) for v. However, if an agent has visited v at t, v is cleaned so $L_t(v) = 0$. Note that agent i cannot know the actual value of $L_t(v)$ except the current position, v^i_t.

Each agent has a *responsible area* (RA) that it tries to keep clean. The RA of agent i at time t is the connected subgraph $G^i_t = (V^i_t, E^i_t)$, where $V^i_t \subseteq V$ and $E^i_t = \{e_{i,j} \in E \mid v_i, v_j \in V^i_t\}$. We assume that $v^i_{base} \in V^i_t$ and V^i_t, and V^j_t are disjoint for i, j ($\in A$ and $i \neq j$). An agent may change the size of its RA, which is $|V^i_t|$, to keep the area evenly clean through balanced cooperative work.

3.4 Performance of Cleaning Tasks

The purpose of cleaning tasks is to minimize the amount of pieces of dirt in the environment without neglecting them. Hence, we use the *sum of the amount of remaining dirt* for the whole area at certain intervals of time as the performance measure of the agents' collective tasks. This is defined as

$$D_{t_s,t_e} = \frac{\sum_{v \in V} \sum_{t=t_s}^{t_e} L_t(v)}{t_e - t_s}, \tag{6}$$

where positive integers t_s and t_e are the start and end times of the interval. A smaller performance value is better, so agents try to minimize the values of D_{t_s,t_e}.

Although D_{t_s,t_e} is an important measure, we also consider the balanced task allocation for cooperative cleaning in which agents that can handle more work take part of the RAs of other agents

that are busy and/or that have less efficient exploration algorithms. Thus, we also pay attention to the sizes of the RAs, V_t^i, to investigate whether or not an efficient agent could do more work in a larger RA, and calculate the amount of remaining dirt in i's RA, which is denoted by $D^i_{t_s,t_e}$. D_{t_s,t_e} and $D^i_{t_s,t_e}$ are often denoted by D and D^i if there is no confusion. Note that balanced task allocation does not necessarily mean equal size of V_t^i.

The proposed probabilistic model of dirt accumulation can also be modified for other patrolling domains such as surveillance. For example, the important locations that require high-level security, such as around safes and entrances/exits correspond to the dirty areas, thus they have higher probabilities, p_v. Furthermore, we can change these probabilities in accordance with time of day. So, for example, agents can visit the important locations more frequently during nighttime hours.

4. Proposed Method

We describe the proposed *extended performance-based partitioning* (ePBP) method, which fairly partitions the given area by taking into account the performances of the individual agents and the characteristics of the area. In our proposed method, we assume that agents have information of V^+ and E but do not know (1) the set of the DAP of nodes, $\{p_v | v \in V\}$ nor (2) the set of obstacles, O (initially agents assume that $O = \emptyset$). Therefore, agents with ePBP concurrently learn the DAPs of their RAs to see which locations in the RAs are easy to become dirty, and the set of obstacles while they decides and negotiates the responsible area with other agents.

4.1 Expansion Power

Although agents do not know the values of p_v for $\forall v \in V$, if agents estimate the values of p_v for $\forall v \in V$, i can estimate $L_t(v)$ using the expected amount of accumulated dirt on v, which is calculated by

$$E(L_t(v)) = p_v^i \cdot (t - t_v^i),$$

where p_v^i is the estimated value of p_v by learning of the dirt accumulation in i, and t_v^i is the most recent time when i visited and cleaned node $v \in V_t^i$; if i never visited v, t_v^i is the time when v was included in its RA, G_t^i. How i calculates p_v^i is explained in Section 4.6. We also define $L_t(V_0) = \sum_{v \in V_0} L_t(v)$ and $E(L_t(V_0)) = \sum_{v \in V_0} E(L_t(v))$ for set of nodes V_0 ($\subset V$).

Agent i calculates its *expansion power* for the current RA when it returns to the charging base at time t. Intuitively, it expresses how efficiently i could have covered the current RA during the latest cleaning cycle. First, i calculates the expected amount of accumulated dirt in the RA at time t

$$E(L(G_t^i)) = \sum_{v \in V_t^i} E(L_t(v)) = \sum_{v \in V_t^i} p_v^i \cdot (t - t_v^i). \tag{7}$$

Then, the expansion power $\xi(i,t)$ of i at time t is defined as the inverse of the expected value:

$$\xi(i,t) = E(L(G_t^i))^{-1}. \tag{8}$$

If $E(L(G_t^i)) = 0$, $\xi(i,t)$ is set to a sufficiently large number. Agents retain their calculated expansion power until the next calculation time.

4.2 Expansion of Responsible Area

The cleaning cycle of each agent starts when it leaves its base node with a fully charged battery to clean its RA using its own exploration algorithm. If it determines that it has mostly cleaned the RA, it may decide to expand its RA. For this decision, agent i calculates the expected amount of accumulated dirt in its RA at a certain future time, $E(L_{t_0+\gamma}(G_{t_0}^i))$, when i leaves from v_{base}^i at time t_0, where γ ($\leq M_i$) is a positive integer. Agent i also stores the number of visited nodes, $N_{vis}(t)$, and the amount of vacuumed dirt, $N_d(t)$, at t ($> t_0$) during the current cleaning cycle, which started from t_0. It then tries to expand its current RA, V_t^i, if the following conditions are fulfilled.

$$N_{vis}(t) \geq R_1 \cdot |V_t^i| \tag{9}$$

$$N_d(t) \geq R_2 \cdot E(L_{t_0+\gamma}(G_{t_0}^i)), \tag{10}$$

where $0 \leq R_1, R_2 \leq 1$, and $0 \leq \gamma \leq M_i$ are the parameters used by agents to determine if they have cleaned most of the current RA. Note that we introduce parameter γ, which specifies the expected amount of dirt in the RA at a certain future time, because dirt will continue to accumulate while the agents move around. Of course, agents may compute $E(L_t(G_t^i))$ every time and can use it in Condition (10) instead of $E(L_{t_0+\gamma}(G_{t_0}^i))$. However, we use $E(L_{t_0+\gamma}(G_{t_0}^i))$ in the following experiments to avoid the frequent calculations of the expected value. These conditions indirectly reflect both the capabilities of the agent's hardware and the quality/performance of the exploration algorithms. Agents with a simple algorithm cannot effectively move around the area (for example, the agents may visit the same nodes many times and/or may skip some nodes). Agents that can move more quickly have a sophisticated exploration algorithm, or have a large-capacity battery can more easily satisfy two conditions and thus are likely to expand their RAs.

Note that a larger R_1 and R_2 make agents more conservative about expanding their RAs. There is a trade-off between conservativeness and eagerness: eager agents with a small R_1 and R_2 try to expand their RAs even if their RAs are not clean enough while conservative agents will avoid expanding their RAs even if they are able to do so, and the adjacent agents have smaller expansion powers. We will discuss this in Section 6.

4.3 Area Expansion Trial

When Conditions (9) and (10) are satisfied, agents believe that they can clean a larger area. Hence, they start an *area expansion trial* (AET), which is the process of trying to expand an RA to cover other nodes that are not covered by other agents or are in the RAs of busier agents. In the AET, we took into account two factors. The first one is the distances from their bases because visiting only far nodes may reduce the agents' and thus the system's overall performances. Second, we also try to avoid frequent failures of expansion in a certain direction in which unbusy agents operate.

When agent i finds that Conditions (9) and (10) are satisfied at time t during its cleaning cycle, i initiates an AET, which consists of two parts. First, i identifies the nodes I^i that should be included in its RA using the *expansion strategy*. It then negotiates with neighbor agents to decide which agent should take charge of

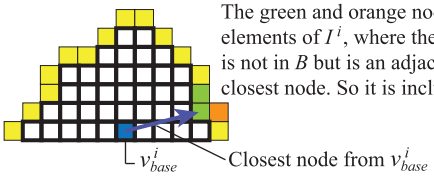

Fig. 1 Expansion strategy (nearest boundary expansion). Squares with bold lines represent the current RA, and yellow and green squares represent boundary of subarea. Blue node represents the base node.

the identified nodes, assuming that part of I^i is in the RAs of the neighbor agents.

The expansion strategy tries to include the boundary nodes closer to the charging base. First, agent i identifies the boundary of its current RA, which is denoted by $B(V^i_t) \subset V$. For example, in **Fig. 1**, where the environment G is a grid, V^i_t is the set of nodes (squares) with bold lines, and the boundary, $B(V^i_t)$, is the set of the yellow and green colored nodes. Agent i selects I^i_{inc} ($\subset B(V^i_t)$), which is the set of k_{inc} nodes that are not in I^i_{avoid} (this will be defined below) and are the nearest from the base, v^i_{base}, for positive integer k_{inc}. It then defines I^i as the nodes in I^i_{inc} and their adjacent (north, south, east, and west) nodes that are not in V^i_t and not in I^i_{avoid}. For example, in Fig. 1, the green and orange nodes express I^i when $k_{inc} = 1$ and $I^i_{avoid} = \emptyset$. If $I^i = \emptyset$, the AET ends, and no nodes are added to i's RA.

If one of the adjacent agents can afford to clean a larger RA, an attempt to take nodes from its RA may fail. To avoid frequent failures of the AET, i stores into I^i_{avoid} the nodes that it failed to take and does not select them as elements of I^i in the next k_{avoid} times of AET, where k_{avoid} is a positive integer. Note that I^i_{avoid} is initially set to $\emptyset$.

4.4 Negotiation for Expanding Responsible Areas

After i identifies I^i, it starts negotiation to determine which agent nodes in I^i should be included in its RA. This process is as follows:

(0) Revise the responsible area:
V^i_t is set to $V^i_{t-1} \cup I^i$.

(1) Send request message for area expansion:
Agent i broadcasts I^i with its current expansion power $\xi = \xi(i,t)$.

(2) Accept/Reject area expansion request:
Suppose that agent j has received a request message for area expansion from i at time t. If $V^j_t \cap I^i = \emptyset$, j does nothing. Otherwise, j compares j's expansion power, $\xi(j,t)$, with ξ.

(2.1) If $\xi(j,t) \geq \xi$, j sends a rejection message with $V^j_t \cap I^i$ and $\xi(j,t)$ to i.

(2.2) If $\xi(j,t) < \xi$, j sends an acceptance message with $V^j_t \cap I^i$ and $\xi(j,t)$ to i. Then j revises its RA to $V^j_t = V^j_t \setminus I^i$.

(3) Expand responsible area:
If i has received a rejection message from j, it excludes the nodes from V^i_t and stores the information about the excluded nodes with j's expansion power. These nodes are stored into I^i_{avoid} and are not be included in I^i in the next k_{avoid} times of AET to avoid frequent failures.

Note that during these message exchanges, i continues to clean the current RA. We assume that AET is invoked only once per cleaning cycle even if i has enough battery to continue in order to avoid excess expansion but, of course, we can omit this restriction.

4.5 Identifying Locations of Obstacles

We assume that agent i can detect obstacles using sensors (e.g., touch/sonar/infrared sensor, proximity sensor and camera) and in this paper, i can detect a node of obstacle by hitting it using touch sensor which is the simplest way. Agent i starts moving from its charging base v^i_{base} along the path generated by an exploration algorithm. It then memorizes the nodes that it cannot move which is defined as *block node* O^i, whose initial value is the empty set. Then, when i hits a node of an obstacle during the cleaning process, it adds them into O^i. Furthermore, if the elements in O^i surround other nodes, these are the part of the obstacles. Thus, they are added into O^i. This enables i to recognize which nodes are the parts of obstacle. After their RAs changed or O^i was revisited, agents recalculated the shortest distance between nodes in the RAs when they arrive at their charging bases.

4.6 Learning of Dirt Accumulation Probabilities

To identify which nodes are easy to become dirty in the RAs, agent i learns p^i_v for $\forall v \in V^i_t$, which are the estimated values of the DAPs of V^i_t. First, when node v is added in $V^i_{t_v}$ at time t_v, i initializes as $p^i_v = 0$ and the last time when i visited v, $t^i_{LV}(v)$, is set to t_v.

Right after i has vacuumed up dirt at node v at time t, i calculates the interval, $I^i_t(v)$, between the current and the last time visited v:

$$I^i_t(v) = t - t^i_{LV}(v). \tag{11}$$

Then, the DAP of v is estimated by $L_t(v)/I^i_t(v)$. However, the reliability of such an estimated value depends on the length of interval, $I^i_t(v)$. Thus, we introduce the variable learning rate, $\alpha(x)$, which weighs the obtained probability according to the length of the interval, and p^i_v is updated as:

$$p^i_v = (1 - \alpha(I^i_t(v)))p^i_v + \alpha(I^i_t(v))\frac{L_t(v)}{I^i_t(v)}. \tag{12}$$

Then $t^i_{LV}(v)$, is set to t. The learning rate function $0 < \alpha(x) < 1$ in Eq. (12) is monotonically increasing and is defined as the linear function with the upper bound:

$$\alpha(x) = \max(\delta x, \alpha_{\max}) \tag{13}$$

in the experiments below, where $0 < \delta \ll 1$ is the gradient of the learning rate, and $0 < \alpha_{\max}$ is the upper bound.

5. Experimental Evaluation

5.1 Environments

We evaluated the ePBP method using two simulated environments, as shown in **Fig. 2**, to clarify the performance and features of the ePBP method in a variety of situations. Cleaning area G is defined as a 51×51 grid. Node v is expressed by (x, y), where $-25 \leq x, y \leq 25$. Four agents $A = \{a_1, a_2, a_3, a_4\}$ move around G starting from their charging bases v^i_{base} $(i = 1, 2, 3, 4)$. The set of obstacles, O, is empty if nothing is stated.

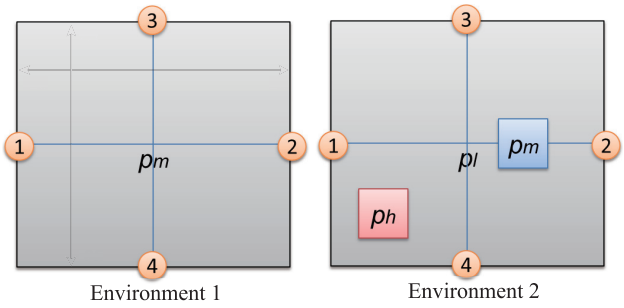

Fig. 2 Experimental environments.

The DAPs for all nodes are shown in the figure, where parameters p_l, p_m, and p_h are defined as

$$p_l = 2 \cdot 10^{-6},\ p_m = 2 \cdot 10^{-5},\ p_h = 2 \cdot 10^{-4}. \quad (14)$$

Whereas dirt accumulates uniformly in the first environment (Env. 1), there are rectangular regions that more easily accumulate dirt in the second environment (Env. 2), where the red region is specified by $(-20, -20)$ and $(-10, -10)$ and the blue region by $(5, -5)$ and $(15, 5)$, so the sizes of these regions are 121. The encircled integers indicate the locations of the charging bases; for example, the charging base of a_1 is at $(-25, 0)$. The subarea whose DAP is p_h in Env. 2 referred to here as the *easy-to-dirty* subarea. Note that since the DAP in Env. 1 is p_m, so Env. 1 is dirtier than Env. 2.

We assume that all agents have the same batteries and we set that $B_{\max} = B^i_{\max} = 900$, $k_{charge} = k^i_{charge} = 3$, and $B_{drain} = B^i_{drain} = 1$ in our experiments if nothing is stated. This means that agents can continuously operate up to 900 ticks ($M_i = 900$) and that a complete recharge takes 2,700 ticks if the battery is empty, making the maximum length of a cleaning cycle 3,600 ticks. We defined these values in accordance with the specifications of an actual robot cleaner [*2]. The parameters for selecting AET were defined as $R_1 = 0.7$, $R_2 = 0.7$, and $\gamma = 300$ ($= M_i/3$). The control parameters k_{inc} and k_{avoid} used in the AET were set to 15 and 7, respectively. The parameters used in the learning of DAPs were set as $\delta = 0.0001$ and $\alpha_{\max} = 0.5$. We then stored, every 3,600 ticks (which is the maximal cleaning cycle) up to 1,000,000 ticks, the sum of the amount of remaining dirt, D, the expansion powers $\xi(a_i, t)$ calculated when the agents returned to their base, and the sizes of the RAs, $|V^i_t|$. The experimental results given below are the average values for 100 trials. We compare these results with those of a conventional distributed partitioning method [9], in which agents try to divide the area into equal-size subareas by comparing the current sizes of their RAs. We call it the *balloon method* [9] hereafter.

We conducted four experiments. In the first experiment (Exp. 1), we compared cleaning performance and examined how the environments were divided in accordance with the environmental characteristics. The second experiment (Exp. 2) investigated how the ePBP could reflect the difference in algorithms of exploration. In the third experiment (Exp. 3), we introduced the agents with the enhanced battery to know how hardware differences affected the RA partitioning. Finally, we added a number of obstacles into the environments to investigate how the ePBP method decided the RAs by reflecting the obstacles, especially a intricately-shaped obstacle, in the fourth experiment (Exp. 4).

[*2] One tick is approximately 4 seconds, the velocity is 0.25 m/s, the maximum operation time is 1 hour, and the maximum battery charging time is 3 hours in our experiments.

5.2 Algorithms for Exploration in Experiments

Agents move around the RAs by using certain exploration algorithms and to verify that the proposed PBP method can determine the RAs by taking into account the differences in algorithm performance, and we assume that the agents use one of three exploration algorithms described below. Because the focus in the experiments is on area partitioning for division of labor, these algorithms are quite simple and non-intelligent; improvement of exploration algorithm out of scope, but agents can use more effective algorithms in our framework.

With the *random exploration* (RE) algorithm, agent i randomly selects target node v from V^i_t and then moves to v along the shortest path from the current node. After reaching the node, i randomly selects another node, i.e., it iterates this select-and-move action.

With the *directed depth-first exploration* (DDFE) algorithm, which is a simple depth-first search, i selects the first target node, $v \in V^i_t$, whose expected amount of accumulated dirt $E(L_t(v))$ is the largest when it leaves v^i_{base}, moves to it along the shortest path, and pushes the node on top of its stack. It then randomly selects one of the adjacent nodes except for a previously visited one, moves to it, and pushes the node on top of its stack. This process is iterated as long as i can select an unvisited node. Then, if i cannot select it, i pops the top node from its stack and backtracks one step; that is, i moves back to the previous node. It then tries to select another unvisited node. If i returns to the first target node after repeating this movement, it returns to its base node, v^i_{base}. Although Ref. [12] used the *(random) depth-first exploration* (DFE) algorithm that is also a depth-first search simpler than DDFE, we did not use it here. DDFE relies on the learned DAPs, so it is better to see the effect of the DAP learning on the performance.

The DDFE algorithm is better than the RE one since an agent using RE may visit the same nodes many times but one using DFE does not visit the same node in a cleaning cycle except when backtracking. Note that agents using these algorithms move to only safe nodes, as we mentioned in Section 3.1. If they find that the next node is not safe, they directly return to their base nodes via shortest paths.

5.3 Performance of Cleaning and Sizes of RAs

For the purpose of Exp. 1, we compare the sum of the amount of remaining dirt, D, in two environments. We assumed that all agents used the DDFE exploring algorithm. We also examined the PBP method in Exp. 1 to investigate the differences in performance between the PBP (the DAPs were given) and the ePBP (the DAPs were learned) methods. The results are plotted in **Fig. 3**. The average values of $D = D_{t_s,t_e}$ observed between $t_s = 800{,}000$ and $t_e = 1{,}000{,}000$ in Env. 1 and Env. 2 and the improvement ratios of the ePBP method to the conventional method are also listed in **Table 1**. For Env. 1, it is reasonable that the area di-

Table 1 Average values of remaining dirt between 800,000 and 1,000,000 ticks.

	Exp. 1		Exp. 2		Exp. 3		Exp. 4	
	D_{t_s,t_e} in Env. 1	D_{t_s,t_e} in Env. 2	D_{t_s,t_e} in Env. 1	D_{t_s,t_e} in Env. 2	D_{t_s,t_e} in Env. 1	D_{t_s,t_e} in Env. 2	D_{t_s,t_e} in Env. 3	D_{t_s,t_e} in Env. 4
Conventional method	127.9	81.6	171.0	106.4	92.8	58.2	131.4	85.6
ePBP method	130.2	67.4	165.7	81.1	85.6	44.0	134.5	68.3
Improvement ratio (%)	−1.80	17.40	3.10	23.78	7.76	24.40	−2.35	20.12

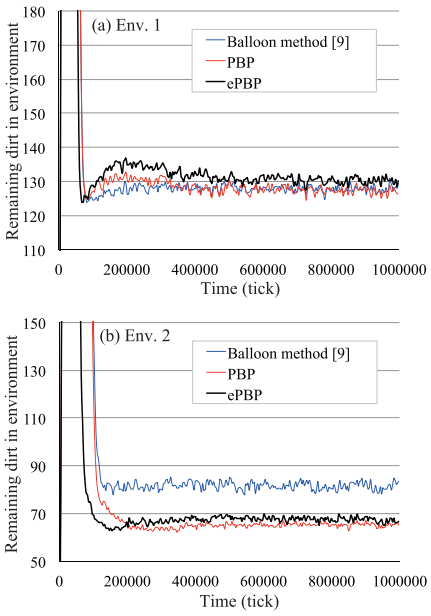

Fig. 3 Amount of Remaining Dirt, D, using PBP and ePBP in Exp. 1.

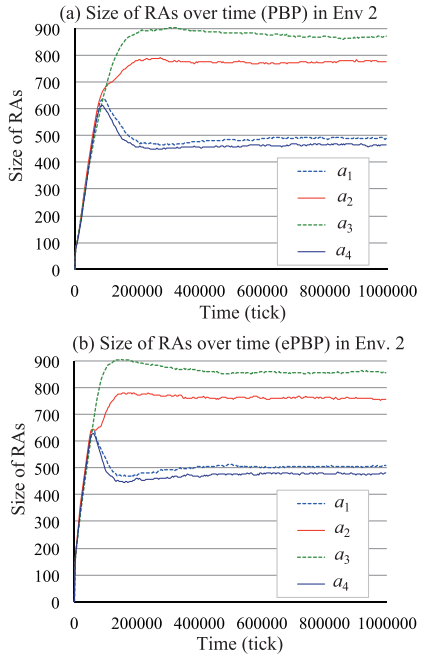

Fig. 4 Sizes of RAs, $|V_t^i|$, in Exp. 1.

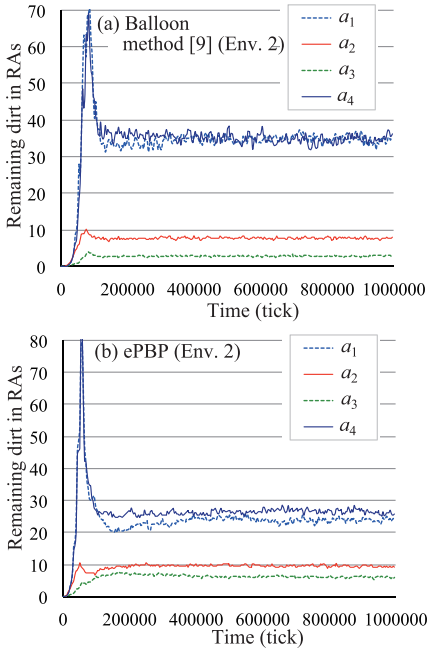

Fig. 5 Remaining Dirt in $|V_t^i|$ in Exp. 1.

visions are equal in size because the DAPs $\{p_v\}_{v\in V}$ are constant. Hence, the differences between the PBP, ePBP, and conventional methods were small although ePBP exhibited slightly lower performances (the improvement ratio was −1.80% in Table 1). In Env. 2, the ePBP and PBP methods resulted in a much smaller D, and the improvement ratio was 17.40% (Table 1), because it partitioned the area in accordance with the characteristics of the environment. We can also observe that the ePBP and PBP exhibited the almost identical performance in both environments (Fig. 3) although agents with the ePBP were not given the values of DAPs.

We investigated how the RAs expanded and were partitioned depending on the PBP and ePBP methods over time in Env. 2; the results are plotted in **Fig. 4**. Note that the results for Env. 1 are omitted because Env. 1 is uniform, so they partitioned the equal-size RAs. However, Env. 2 has two easy-to-dirty subareas, so the equal-size partitioning is inappropriate. First, Fig. 4 (a) and (b) indicates that the sizes of RAs of a_i, $|V_t^i|$, were almost indifferent between the PBP and ePBP methods in Env. 2. With both methods, agents a_1 and a_4 had bases near the easy-to-dirty subarea, so their RAs were smaller than those of the others (note that p_h is 10 times larger than p_m). The RA of a_3 was the largest because the area near its charging base rarely got dirty.

Figures 3 and 4 indicate that the convergences were slightly faster when agents adopted the ePBP method. Because they initially believed that the environment was uniform and had no easy-to-dirty subareas, they tried to extend their RAs to proactively clean the wider areas.

Figure 5 plotted the amount of remaining dirt, D^i, in Env. 2 when agents adopted the conventional or ePBP method. It shows that the differences in D^i were quite smaller in the ePBP method than those in the conventional method; this is the result of better partitioning of RAs for balanced work by taking into account the characteristics of Env. 2. Note that Fig. 5 (b) indicates that the values of D^i did not become identical. The main reason is

that when they charged (maximally, 2,700 ticks), the amount of dirt increased, especially, in the nodes whose DAP were high. Actually, in Env. 1, D^i converged to an identical value in all experiments below. As an example, we will show this fact in Exp. 2 because its experimental setting was more diverse than that of Exp. 1.

5.4 Effect of Different Exploration Algorithms

In Exp. 2, we gave agents two different exploration algorithms, RE and DDFE, described in Section 5.2; agents a_1 and a_2 used RE and a_3 and a_4 used DDFE. The subarea near the charging bases for a_1 and a_4 would be the dirtiest although RE is less effective than DDFE. In realistic situations, the agents using a better algorithm should be allocated to the dirtier areas. We did not do this because we wanted to clarify the effect of the differences in algorithms and environments on performance and RA partitioning.

Figure 6 is the set of graphs showing the amount of remaining dirt, D, in Envs. 1 and 2 over time. We also listed the average value of D between 800,000 and 1,000,000 ticks in Table 1. These data indicate that the ePBP method could clean more effectively, especially in Env. 2 like Exp. 1, than the conventional method.

We also plotted the sizes of RAs of a_i with the ePBP method in Envs. 1 and 2 in **Fig. 7**. In Env. 1, agents with the ePBP method autonomously divided their RAs in accordance with their used exploration algorithms. In Env. 2, by adding the easy-to-dirty subareas, the tendency was more notable; for example, a_3 and a_4 used the DDFE, but a_3 had no dirty subareas near its charging base, so the size of its RA became 1,000 nodes approximately. In contrast, a_4 had the small RA that was smaller than a_2's RA. The graphs in **Fig. 8** show the amount of remaining dirt in RAs, D^i (for $i = 1, 2, 3, 4$), when agents adopted the conventional method (Fig. 8 (a) and (c)) and the ePBP method (Fig. 8 (d) and (d)). We can find that the proposed ePBP method could clean the RAs more evenly in both environments. Note that in Env. 2 (Fig. 8 (d)), the values of D^1 and D^4 were relatively larger although agents tried to divide the RAs for balanced work. This reason is identical to the case in Exp. 1; the amount of dirt increased in the nodes whose DAP were high when they charged. Note again that p_h is 10 times larger than p_m.

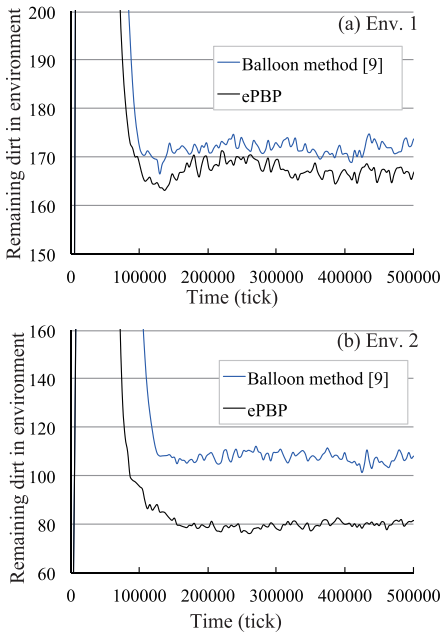

Fig. 6 Amount of Remaining Dirt, D, in Exp. 2.

5.5 Effect of Hardware Difference

We conducted Exp. 3 to see the effect of hardware difference, more specifically different capacities of batteries, on the sizes of RAs and on the performance of cleaning. We assumed that a_1 and a_2 had the same battery in the previous experiments, but a_3 and a_4 had a better (long-life) battery that is specified as $B^3_{\max} = B^4_{\max} = 1{,}800$ (and other battery specifications k^3_{charge}, k^4_{charge}, B^3_{drain}, and B^4_{drain} are identical to other's batteries). Other experimental setting was identical to that of Exp. 1.

Figure 9 (a) and (b) shows the amount of dirt remaining in the environments over time. The average value of D between 800,000 and 1,000,000 ticks is also listed in Table 1. They indicate that the ePBP method could outperform the conventional method due to better allocations of RAs as shown in **Fig. 10** (a) and (b), which shows the sizes of RAs allocated to agents, a_i in Envs. 1 and 2. Figure 10 (a) indicates that a_3 and a_4 had larger sizes of RAs than those of a_1 and a_2 in accordance with their battery capacity specifications. In Env. 2, Figure 10 (b) exhibits more interesting curves: until 5,000 ticks, the sizes of RAs were similar to those in Env. 1. After that, because the agents began to include the dirtier subareas in their RAs and to learn the DAPs, agents changed the sizes of their RAs by reflecting the environment and the battery capacities shortly. Thus, the RA of a_4, for example, became smaller although it has better battery, and conversely, the size of a_2's RA became larger. Note again that p_h is 10 times larger than p_m.

Figure 11 (a) and (b) shows the amount of remaining dirt in RAs. We omitted the graphs in Env. 1 but found that the ePBP method could keep clean uniformly in Env. 1 due to the balanced

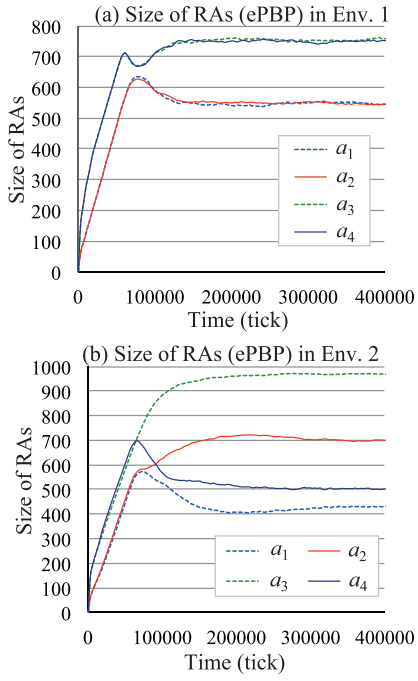

Fig. 7 Sizes of RAs, $|V^i_t|$, in Exp. 2.

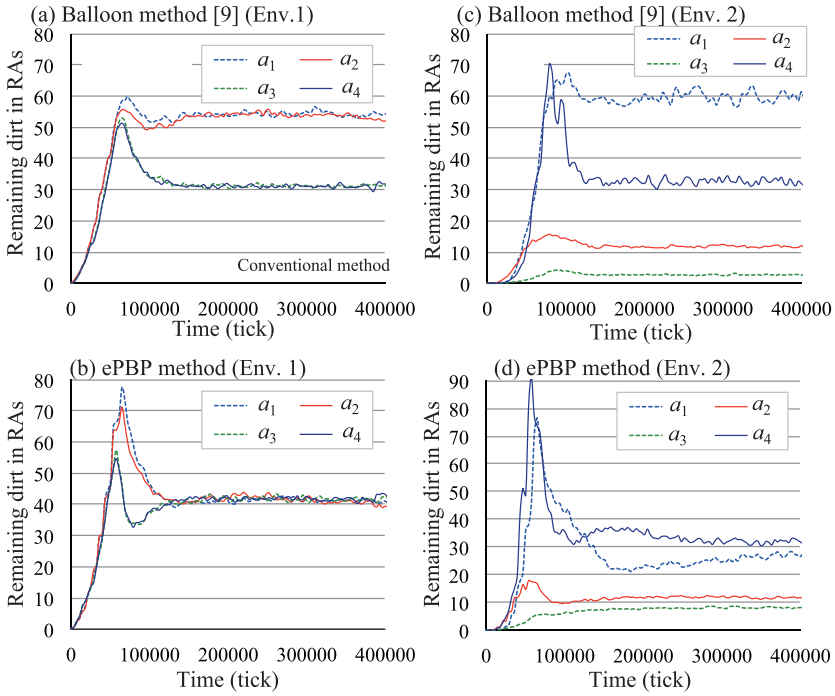

Fig. 8 Remaining Dirt in $|V_t^i|$ in Exp. 2.

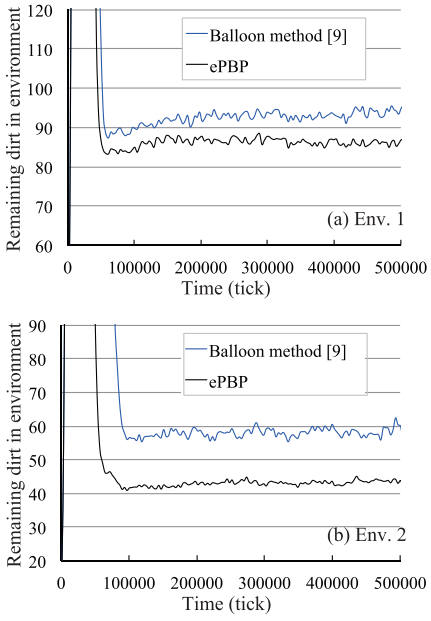

Fig. 9 Amount of Remaining Dirt, D, using Conventional Method and ePBP in Exp. 3.

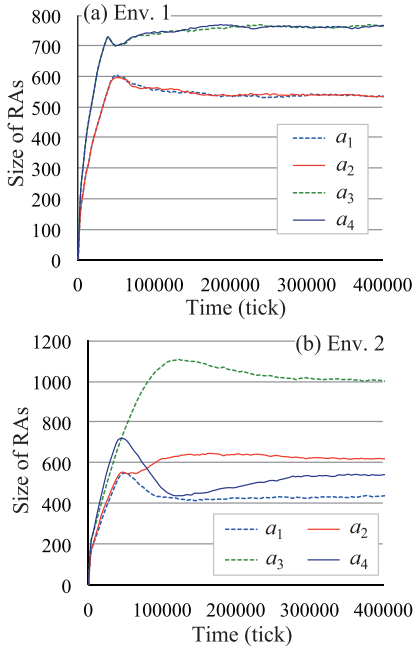

Fig. 10 Sizes of RAs, $|V_t^i|$, in Exp. 3.

work allocations. It also made the difference in remaining dirt between RAs smaller in Env. 2 (Fig. 11 (a) and (b)); the difference was caused by the increase of dirt in the easy-to-dirty subarea during battery charge.

5.6 Balanced RA allocations with Obstacles

In Exp. 4, we investigated how existence of obstacles and their shapes affected their sizes of RAs. For this purpose, we put three obstacles into Envs. 1 and 2 with different shapes, including square, rectangular and E-shape, as shown in **Fig. 12**. These environments are referred to as Env. 3 and Env. 4, respectively. The square obstacle is specified by $(-18, -3)$ and $(-13, 2)$, while the rectangular is specified by $(13, -6)$ and $(18, 3)$. The size and location of the E-shape obstacle is shown in Fig. 12. Note that the rectangular obstacle partly overlapped the dirtier subarea whose DAP is p_m.

When a number of obstacles exist in the environment, we could observe the slightly different phenomenon. **Figure 13** presents how remaining dirt, D, varied overtime (until 1,000,000 ticks) in Envs. 3 and 4. We also listed the improvement ratios of D between 800,000 and 1,000,000 ticks in Table 1. Figure 13 and Table 1 indicate that the ePBP method left slightly more dirt in Env. 3 than the conventional method as in Exp. 1, although the ePBP outperformed the conventional method in Env. 4. Because

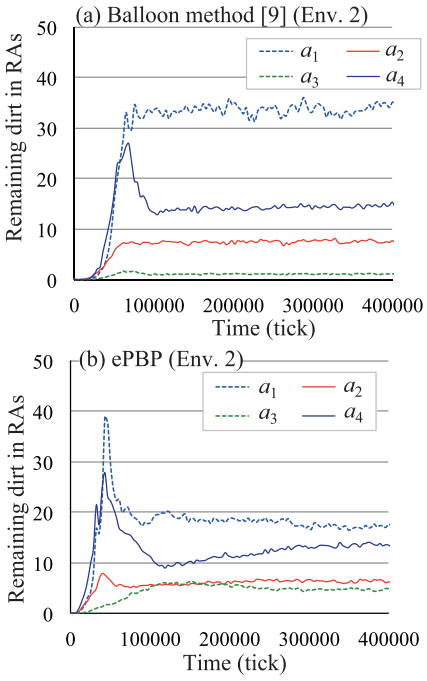

Fig. 11 Remaining Dirt in $|V_t^i|$ in Exp. 3.

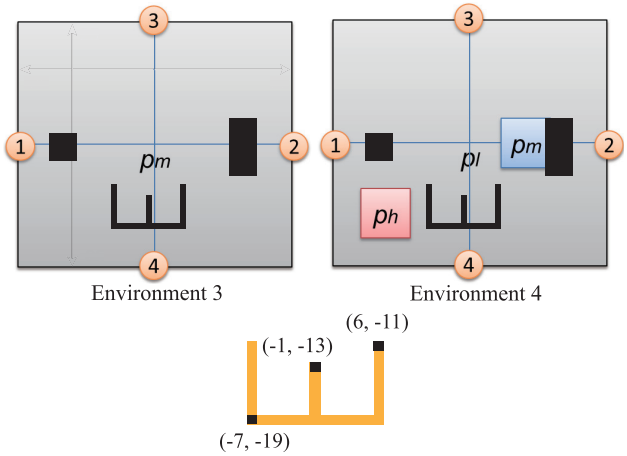

Fig. 12 Experimental environments.

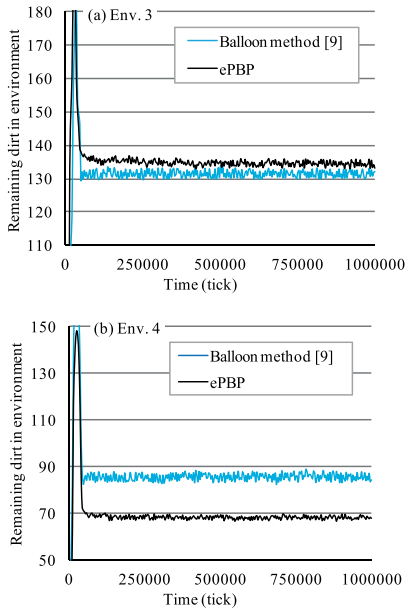

Fig. 13 Amount of Remaining Dirt, D, in Exp. 4.

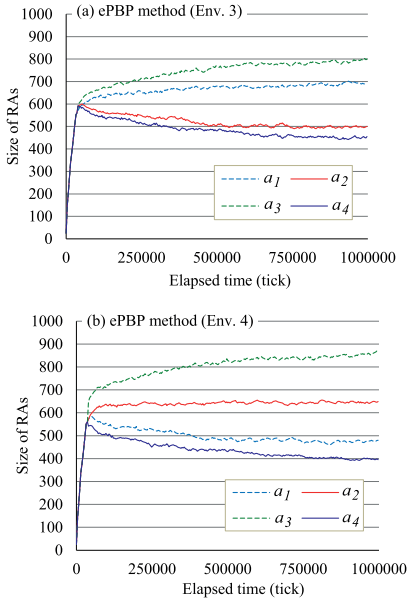

Fig. 14 Sizes of RAs, $|V_t^i|$, in Exp. 4.

Env. 3 is uniform except the three obstacles which hindered for the learning of the DAPs, the ePBP could not learn the DAP values efficiently. Figure 13 also shows that the values of D almost converged around 5,000 ticks. However, if we look at Fig. 13 (a) and (b) more carefully, the values of D decreased very slowly after that.

Figure 14 represents the size of RAs of agent a_i in both Envs. 3 and 4 using our proposed method. Note that the sizes of RAs excluded the nodes occupied by obstacles. Figure 14 (a) indicates that agents autonomously divided the areas on the basis of only the existence and the shapes of the obstacles since Env. 3 is uniform. For example, a_4 had the E-shaped obstacle that is more complex than others, and it took more ticks to reach the areas inside the E-shared obstacle. This results in the smaller a_4's RA than others. The RA of a_2 was also smaller because it had the rectangular obstacle which took slightly longer time to reach the nodes in the opposite side of the rectangle from the a_2's base, v_{base}^2. This situation is also similar for a_1 but the obstacle near v_{base}^1 was smaller, so the RA of a_1 was relatively larger.

On the other hand, because Env. 4 has a number of easy-to-dirty subareas, the area partition reflected both the obstacles and the characteristics of the environment. Figure 14 (b) indicates that because a_4 had both the E-shape obstacle and the easy-to-dirty subarea near the charging base, its RA was the smallest (about 400). In addition, the RA of a_3 was the largest (about 860) because it was neither easy-to-dirty subarea nor obstacles near its charging base. Of course, agents with the conventional method have equal-size RAs, thus the RA including the complex-shaped obstacle and easy-to-dirty region tended to have more remaining pieces of dirt.

Figure 15 (a) and (b) represents the amount of dirt left in the RAs in Env. 4 using the conventional and the proposed methods, respectively. Figure 15 (a) indicates that the differences in the amount of remaining dirt in RAs, D^i $(i = 1, 2, 3, 4)$, were large

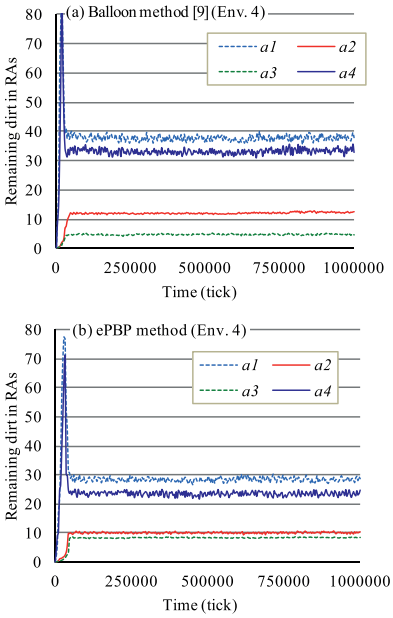

Fig. 15 Remaining Dirt in $|V_t^i|$ in Exp. 4.

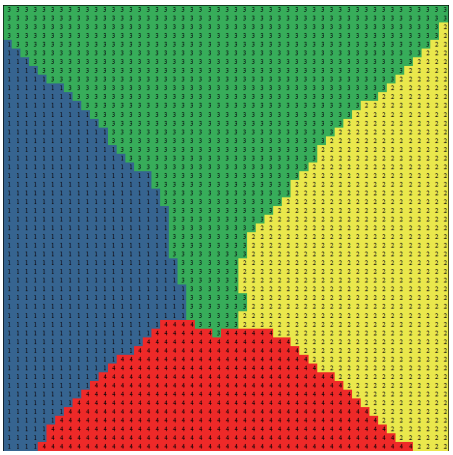

Fig. 16 Shape of RA in Env. 4.

but by using the proposed method, we can see from Fig. 15 (b) that agents could keep the values of D^i closer. This result shows that our proposed method could vacuum dirt in a more balanced manner.

We can observe two phenomena different from other experiments. First, if we compare the results of a_1 and a_4 in Figs. 14 (b) and 15 (b), we can see that the size of a_4's RA was smaller but the a_1's RA was dirtier. This indicates that because a_4 had E-shaped obstacle, a_1 cleaned the dirty subarea between v^1_{base} and v^4_{base} more than a_4.

Second, Fig. 14 obviously indicates that it took longer time to converge the sizes of RAs. We can consider two reasons for this (see also Fig. 4). First, agents required more time to reach and thereby learn the *DAPs* of the regions in the opposite side of obstacles, especially another side of the E-shaped obstacle. In addition, the exploring algorithm used in this experiment was too simple to clean effectively such a complex region. Second, the existence of obstacles let the speed of expansion of RAs slower because agents first try to expand them to the nearest nodes. This discussion suggests the limitation of the proposed method; i.e., we have to improve the learning speed, and we will address this issue next time.

How environment is partitioned is shown in **Fig. 16**. Note that we selected this result of partitioning randomly from 100 experimental trials we conducted, and we could see that other partitioning looked similar. We can see in Fig. 16 that a_4 had the smallest RA because an E-shape obstacle is next to its charging base, and the dirty area whose probability is p_h is also in its RA. Particularly, this obstacle made the cleaning difficult, and a_4 needed to spend longer time. Thus, a_4 decreased its RA. However, the RA of a_3 was the biggest because there is no obstacle nearby its charging base nor the dirty areas.

6. Discussion

From the results of our experiments, we can say that the proposed ePBP method can effectively partition the area in accordance with the differences in the environment and the performances of the agents in a cooperative cleaning task. The agents that: (a) use more efficient algorithms, (2) have high-capacity batteries, and/or (3) are deployed in regions that are relatively simpler and cleaner can handle larger areas, and thus, they try to expand their RAs by acquiring nodes from busier agents. Furthermore, although the ePBP method does not assume the information of dirty areas, i.e., the values of DAPs, it exhibits the performance comparable to the PBP [12]. However, a few things need to be considered.

The first thing to consider is the effect of the parameters used. Parameters R_1, R_2, and γ, which are used in Conditions (9) and (10) specify the situations in which agents start an AET. If these parameters are large, agents tend to expand their RAs only after they have sufficiently cleaned their current RAs. That is, they are conservative about expanding their RAs even if the adjacent agents lack the performance needed to clean their areas. If these parameters are small, the agents tend to start an AET even before their current RAs have been sufficiently cleaned, so AETs are started more frequently. This can result in frequent meaningless AETs. Parameter k_{inc} controls the number of nodes acquired in a single AET, and parameter k_{avoid} controls the number of fruitless AETs in which agents try to extend their RA towards the expense of agents with high expansion powers. The trade-off mentioned here is similar to the explore-or-exploit dilemma that occurs with learning algorithms. We think that the learning is needed to decide the values of these parameters: This is left to our future work.

Finally, as shown in Exp. 4, the convergence became slower when the environment had a number of obstacles. When its shape was complex, like the E-shaped obstacle, in particular, agents could rarely reach recessed areas inside the complex obstacle due to a number of reasons, and this resulted in the inefficient learning. First, the exploring algorithm used in our experiments was so simple to explore such recessed areas. Second, more importantly, agents had no information about the DAP and initially assumed that such recessed areas were not so dirty, so there was no motives to move there. For example, if the recessed areas were easy-to-dirty, agents gradually learned it and visited there more often. However, in our experiment, the recessed areas were not

easy to be dirty. This is also another issue that we should address in the future.

By using area expansion trial (AET), agents can adaptively expand their RAs. If the room is large, agents can expand their RA rapidly by adjusting the parameters used in the AET strategy. However, we cannot decide the maximum size of the cleaning area because it depends on the specifications and the number of agents. Note that most of the computational cost in our proposed method occurs in the calculation of the expansion power, and is $O(m)$, where m is the size of RA.

Our proposed method could partition the area/environment fairly and effectively by taking into account the characteristics of the environment and the capability of each agent. However, some additional issues such as map generation, path planning, identifying agents' locations, collision/obstacle avoidance, compensation for imperfect communication and how to identify the appropriate number of agents for efficient cleaning exist for the real applications of cleaning/sweeping domains. In particular, although the appropriate number of agents depends on the agent's specifications, it is important to introduce some mechanism, which contribute both efficiency in the cleaning and energy saving, to decide the appropriate number of the cleaning robots. For example, if the room is very dirty, then the number of agents should be increased. Yet, if some agents are redundant, the number of agents should be reduced by improving their specifications. This issue should be solved and is our next future work.

In addition, our work is not restricted to only the cleaning application. We can apply it to other real-world applications such as the security patrolling. Agents in this problem domain must visit/monitor locations in environment at different frequencies. For instance, continuous cleaning and security patrolling agents have to control robots so that they frequently visit regions that easy to accumulate dirt and those at high security levels. Thus, the cleaning task is just an example for our experiments described in Section 5.

7. Conclusion

We have presented a decentralized area-partitioning method for cleaning and patrolling tasks. It tries to uniformly keep clean/secure the given environment by allocating areas of responsibility in accordance with the characteristics of the environment and the performance of the exploration algorithms. We first modeled the environment, the agents, and the problem addressed here. Then, we described the proposed method in which agents try to expand their responsible areas and negotiate with adjacent agents to determine which agents should clean the identified boundary nodes while they learn what areas are easy to be dirty. Experimental results demonstrated that the proposed method can effectively divide an area into subareas (responsible areas) fairly and appropriately in accordance with the efficiency and capability of agents and the characteristics of the environments. As a result, unbalanced tasks are resolved, and the task is completed in a more balanced and efficient manner.

Our work mainly focused on the cleanliness of floor whose purpose is to minimize the amount of remaining dirt left in the whole environment after each cleaning. We think that energy consumption of the cleaning robot is important, but it has not been considered yet and must be related with the appropriate control of the number of agents and their operating time.

We can consider a number of future works to make our method practical as discussed in Section 6. Although applying our method to a new room relies on other methods to create the map of environment as discussed in Section 3.1, we believe that combining our method and a map creation seems better for actual application. Additionally, we plan to find a way to appropriately control the parameter values to enable more autonomous and intelligent activities and to speed up the convergence.

Acknowledgments This work was in part supported by JSPS KAKENHI grant number 25280087. The authors would like to thank anonymous reviewers for their valuable comments to improve this paper. We are also grateful to Mr. Keisuke Yoneda and Mr. Ayumi Sugiyama for their constructive comments and suggestions.

References

[1] Agmon, N. and Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots, *Proc. 15th Annual ACM-SIAM Symposium on Discrete algorithms* (*SODA '04*), pp.1070–1078, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (2004).

[2] Ahmadi, M. and Stone, P.: Continuous Area Sweeping: A Task Definition and Initial Approach, *Proc. 12th International Conference on Advanced Robotics*, pp.316–323 (2005).

[3] Ahmadi, M. and Stone, P.: A Multi-Robot System for Continuous Area Sweeping Tasks, *Proc. 2006 IEEE International Conference on Robotics and Automation*, pp.1724–1729 (2006).

[4] Breitenmoser, A., Schwager, M., Metzger, J.-C., Siegwart, R. and Rus, D.: Voronoi coverage of non-convex environments with a group of networked robots, *2010 IEEE International Conference on Robotics and Automation* (*ICRA*), pp.4982–4989 (2010).

[5] Chevaleyre, Y.: Theoretical Analysis of the Multi-agent Patrolling Problem, *Proc. Intelligent Agent Technology*, pp.302–308 (2005).

[6] Cortes, J., Martinez, S. and Bullo, F.: Spatially-Distributed Coverage Optimization and Control with Limited-Range Interactions, *ESAIM: Control, Optimisation and Calculus of Variations*, pp.691–719 (2004).

[7] Dutta, A., Dasgupta, P., Baca, J. and Nelson, C.: A Bottom-Up Search Algorithm for Dynamic Reformation of Agent Coalitions under Coalition Size Constraints, *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence* (*WI*) *and Intelligent Agent Technologies* (*IAT*), Vol.2, pp.329–336 (2013).

[8] Elmaliach, Y., Agmon, N. and Kaminka, G.A.: Multi-robot Area Patrol Under Frequency Constraints, *Annals of Mathematics and Artificial Intelligence*, Vol.57, No.3-4, pp.293–320 (2009).

[9] Elor, Y. and Bruckstein, A.M.: Multi-a(ge)nt Graph Patrolling and Partitioning, *Proc. 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technologies*, pp.52–57 (2009).

[10] Hahnel, D., Burgard, W., Fox, D. and Thrun, S.: An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements, *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS 2003*), Vol.1, pp.206–211 (2003).

[11] Hennes, D., Claes, D., Meeussen, W. and Tuyls, K.: Multi-Robot Collision Avoidance with Localization Uncertainty, *Proc. 11th International Conference on Autonomous Agents and Multiagent Systems* (*AAMAS 2012*) *– Volume 1*, *IFAAMAS*, pp.147–154 (2012).

[12] Kato, C. and Sugawara, T.: Decentralized Area Partitioning for a Cooperative Cleaning Task, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, Boella, G., Elkind, E., Savarimuthu, B., Dignum, F. and Purvis, M. (Eds.), Lecture Notes in Computer Science, Vol.8291, pp.470–477, Springer Berlin Heidelberg (2013).

[13] Kurabayashi, D., Ota, J., Arai, T. and Yoshida, E.: Cooperative Sweeping by Multiple Mobile Robots, *Proc. 1996 IEEE International Conference on Robotics and Automation* (*ICRA 96*), pp.1744–1749 (1996).

[14] McCaffrey, J.: Graph partitioning using a Simulated Bee Colony algorithm, *2011 IEEE International Conference on Information Reuse and Integration* (*IRI*), pp.400–405 (2011).

[15] Mead, R., Weinberg, J.B. and Croxell, J.R.: An implementation of

robot formations using local interactions, *Proc. 22nd National Conference on Artificial Intelligence - Volume 2*, *AAAI '07*, pp.1989–1990. AAAI Press (2007).
[16] Ranjbar-Sahraei, B., Weiss, G. and Nakisaee, A.: A Multi-robot Coverage Approach Based on Stigmergic Communication, *Multiagent System Technologies*, Timm, I. and Guttmann, C. (Eds.), Lecture Notes in Computer Science, Vol.7598, pp.126–138, Springer Berlin Heidelberg (2012).
[17] Sampaio, P.A., Ramalho, G. and Tedesco, P.: The Gravitational Strategy for the Timed Patrolling, *Proc. 2010 22nd IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, *ICTAI '10*, pp.113–120, Washington, DC, USA, IEEE Computer Society (2010).
[18] Schwager, M., Rus, D. and Slotine, J.-J.: Unifying Geometric, Probabilistic, and Potential Field Approaches to Multi-robot Deployment, *Int. J. Rob. Res.*, Vol.30, No.3, pp.371–383 (2011).
[19] Sea, V. and Sugawara, T.: Area Partitioning Method with Learning of Dirty Areas and Obstacles in Environments for Cooperative Sweeping Robots, *Proc. 4th IIAI International Congress on Advanced Applied Informatics* (*IIAI AAI 2015*), pp.523–529 (2015).
[20] Wolf, D.F. and Sukhatme, G.S.: Mobile Robot Simultaneous Localization and Mapping in Dynamic Environments, *Autonomous Robots*, Vol.19, No.1, pp.53–65 (2005).
[21] Yoneda, K., Sugiyama, A., Kato, C. and Sugawara, T.: Learning and relearning of target decision strategies in continuous coordinated cleaning tasks with shallow coordination, *Web Intelligence*, Vol.13, No.4, pp.279–294 (online), DOI: http://dx.doi.org/10.3233/WEB-150326 (2015).

Vourchteang Sea is a graduated student of Department of Computer Science and Engineering, Waseda University, Japan, since September, 2013. She received her B.E. degree in Computer Science from Royal University of Phnom Penh and M.S. of Engineering from Waseda University in 2012 and 2015, respectively. Currently, she is a Ph.D. student at Waseda University. Her research interests are related to artificial intelligence including multi-agent systems, machine learning and distributed systems.

Chihiro Kato is a graduated student of Department of Computer Science and Engineering, Graduate School of Fundamental Science and Engineering, Waseda University, Japan, since April, 2013. He received B.E. and M.S. degrees, in 2013 and 2015, respectively. His research interests are in coordination, multi-agent systems and machine learning.

Toshiharu Sugawara is a professor of Department of Computer Science and Communications Engineering, Waseda University, Japan, since April, 2007. He received his B.S. and M.S. degrees 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, machine learning, and distributed network management. He is a member of IEEE, ACM, AAAI, IPSJ, JSSST, and JSAI.