Theoretical Computer Science 705 (2018) 9–30

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Move-optimal partial gathering of mobile agents in asynchronous trees ☆,☆☆

Masahiro Shibata [a,*], Fukuhito Ooshita [b], Hirotsugu Kakugawa [c], Toshimitsu Masuzawa [c]

[a] Department of Computer Science and Electronics, Kyushu Institute of Technology, 680-4, Kawatsu, Iizuka, Fukuoka, 820-8502, Japan
[b] Graduate School of Information Science, NAIST, 8916-5, Takayama, Ikoma, Nara 630-0192, Japan
[c] Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan

ARTICLE INFO

Article history:
Received 19 May 2015
Received in revised form 17 September 2017
Accepted 19 September 2017
Available online 27 September 2017
Communicated by D. Peleg

Keywords:
Distributed system
Mobile agent
Gathering problem
Partial gathering problem

ABSTRACT

In this paper, we consider the partial gathering problem of mobile agents in asynchronous tree networks. The partial gathering problem is a generalization of the classical gathering problem, which requires that all the agents meet at the same node. The partial gathering problem requires, for a given positive integer g, that each agent should move to a node and terminate so that at least g agents should meet at each of the nodes they terminate at. The requirement for the partial gathering problem is weaker than that for the (well-investigated) classical gathering problem, and thus, we clarify the difference on the move complexity between them. We consider two multiplicity detection models: weak multiplicity detection and strong multiplicity detection models. In the weak multiplicity detection model, each agent can detect whether another agent exists at the current node or not but cannot count the exact number of the agents. In the strong multiplicity detection model, each agent can count the number of agents at the current node. In addition, we consider two token models: non-token model and removable token model. In the non-token model, agents cannot mark the nodes or the edges in any way. In the removable-token model, each agent initially leaves a token on its initial node, and agents can remove the tokens. Our contribution is as follows. First, we show that for the non-token model agents require $\Omega(kn)$ total moves to solve the partial gathering problem, where n is the number of nodes and k is the number of agents. Second, we consider the weak multiplicity detection and non-token model. In this model, for asymmetric trees, by a previous result agents can achieve the partial gathering in $O(kn)$ total moves, which is asymptotically optimal in terms of total moves. In addition, for symmetric trees we show that there exist no algorithms to solve the partial gathering problem. Third, we consider the strong multiplicity detection and non-token model. In this model, for any trees we propose an algorithm to achieve the partial gathering in $O(kn)$ total moves, which is asymptotically optimal in terms of total moves. At last, we consider the weak multiplicity detection and removable-token model. In this model, we propose an algorithm to achieve the partial gathering in $O(gn)$ total moves. Note that in this model, agents require $\Omega(gn)$ total moves

☆ The conference version of this paper is published in the proceedings of 21th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2014).

☆☆ This work was supported by JSPS KAKENHI Grant Numbers 24500039, 24650012, 25104516, 26280022, and 26330084.

* Corresponding author.
E-mail addresses: shibata@cse.kyutech.ac.jp (M. Shibata), f-oosita@is.naist.jp (F. Ooshita), kakugawa@ist.osaka-u.ac.jp (H. Kakugawa), masuzawa@ist.osaka-u.ac.jp (T. Masuzawa).

https://doi.org/10.1016/j.tcs.2017.09.016

to solve the partial gathering problem. Hence, the second proposed algorithm is also asymptotically optimal in terms of total moves.

1. Introduction

1.1. Background and our contribution

A *distributed system* is a system that consists of a set of computers (*nodes*) and communication links. In recent years, distributed systems have become large and design of distributed systems has become complicated. As a way to design efficient distributed systems, (mobile) agents have attracted a lot of attention [1–5]. Agents can traverse the system and process tasks on each node, and hence they can simplify design of distributed systems.

The classical gathering problem is a fundamental problem for agents' cooperation [1,6–10]. This problem requires all agents to meet at a single node in finite time. The classical gathering problem is useful because, by meeting at a single node, all agents can share information or synchronize behaviors among them.

In this paper, we consider a variant of the classical gathering problem, called the *partial gathering problem* [11]. The partial gathering problem does not always require all agents to gather at a single node, but requires agents to gather partially at several nodes. More precisely, we consider the problem which requires, for a given positive integer g, that each agent should move to a node and terminate so that at least g agents should meet at each of the nodes they terminate at. We define this problem as the *g-partial gathering problem*. From a practical point of view, the g-partial gathering problem is still useful especially in large-scale network. When agents achieve the g-partial gathering, agents can share information and process tasks among at least g agents. In addition, while in the classical gathering agents meet at a single node, in the g-partial gathering agents meet at multiple nodes separately. This means that each group with at least g agents can partition the network and own its area that they should monitor efficiently. The g-partial gathering problem is interesting to investigate also theoretical point of view. Clearly, if $k/2 < g \leq k$ holds, the g-partial gathering problem is equivalent to the classical gathering problem. On the other hand, if $2 \leq g \leq k/2$ holds, the requirement for the g-partial gathering problem is weaker than that for the classical gathering problem. Thus there exists possibility that the g-partial gathering problem can be solved with fewer total moves (i.e., lower costs).

The g-partial gathering problem in unidirectional ring networks with whiteboards at nodes is studied in [11]. For distinct agents (i.e., agents having distinct IDs), the paper proposes a deterministic algorithm to solve the g-partial gathering problem in $O(gn)$ total moves without knowledge of k. For anonymous agents (i.e., agents having no IDs), the paper proposes a randomized algorithm to solve the g-partial gathering problem in $O(n \log k + gn)$ expected total moves. Since the classical gathering problem requires $\Omega(kn)$ total moves, these results show that the g-partial gathering problem can be solved in fewer total moves compared to the classical gathering problem. Moreover, since the g-partial gathering problem requires $\Omega(gn)$ total moves if $g \geq 2$, the paper showed that the deterministic algorithm is asymptotically optimal in terms of total moves.

In this paper, we consider the g-partial gathering problem for asynchronous tree networks for the case of $2 \leq g \leq k/2$. Since trees have lower symmetry than rings, we aim to solve the g-partial gathering problem in models weaker than the whiteboard model previously considered in the ring scenario. The contribution of this paper is summarized in Table 1. We consider two multiplicity detection models and two token models. Note that any combination of these multiplicity detection models and token models is weaker than the whiteboard model. First, we consider the non-token model. In this case, we show that agents require $\Omega(kn)$ total moves to solve the g-partial gathering problem even for the strong multiplicity detection model. We omit this result in Table 1. Next, we consider the case of the weak multiplicity detection and non-token model, where the weak multiplicity detection model assumes that each agent can detect whether another agent exists at the current node or not but cannot count the exact number of the agents. In this case, for asymmetric trees, by the result in [10] agents can achieve the g-partial gathering problem in $O(kn)$ total moves. From the lower bound of the total moves for non-token model, this algorithm is asymptotically optimal in terms of total moves. In addition, for that case that the tree is symmetric and $g \geq 5$ holds, we show that there exist no algorithms to solve the g-partial gathering problem. Hence, we need to relax the restriction of either the multiplicity detection or the token model. Next, we consider

Table 1
Results in each model.

	Model 1 (Section 4)		Model 2 (Section 5)	Model 3 (Section 6)
Token model	Non-token		Non-token	Removable-token
Multiplicity detection	Weak		Strong	Weak
Tree topology	Asymmetric	Symmetric	Arbitrary	Arbitrary
Solvability	Solvable	Insolvable ($g \geq 5$)	Solvable	Solvable
The total moves	$\Theta(kn)$ [10]	–	$\Theta(kn)$	$\Theta(gn)$

the case that the restriction of the multiplicity detection is relaxed: the strong multiplicity detection and non-token model, where the strong multiplicity detection model allows each agent to count the number of agents at the current node. In this case, we propose a deterministic algorithm to solve the g-partial gathering problem in $O(kn)$ total moves. From the lower bound of the total moves for the non-token model, this algorithm is also asymptotically optimal in terms of the total moves. Finally, we consider the case that the restriction of the token model is relaxed: the weak multiplicity detection and removable-token model. In this case, we propose a deterministic algorithm to solve the g-partial gathering problem in $O(gn)$ total moves. This result shows that the total moves can be reduced by using tokens. Note that in this model, agents require $\Omega(gn)$ total moves to solve the g-partial gathering problem. Hence, this algorithm is also asymptotically optimal in terms of the total moves.

1.2. Related works

Many fundamental problems for cooperation of mobile agents have been studied in literature. For example, the searching problem [2,5,12], the gossip problem [3], the election problem [13], the map construction problem [4], and the classical gathering problem [1,8–10,14] have been studied.

In particular, the classical gathering problem has received a lot of attention and has been extensively studied in many topologies, which include lines [15,16], trees [1,6–10,17], tori [1,18], arbitrary graphs [15,19,20] and rings [1,3,14,15,21]. Recently, the classical gathering problem for trees has been extensively studied because tree networks are utilized in a lot of applications.

For example, Fraigniaud and Pelc [6] considered the gathering problem in tree networks for the first time. This algorithm achieves the gathering for two synchronous agents with an arbitrary delay in starting time. The space complexity for each agent is $O(\log n)$ bits, which is asymptotically optimal [7]. Later, they considered the space complexity for the case that two synchronous agents start the algorithm at the same time [7]. In this case, they proposed an algorithm to achieve the gathering for $O(\log l + \log\log n)$ memory per agent, where l is the number of leaves.

The time complexity required for two agents' gathering in tree networks is considered in [8,9]. Czyzowicz et al. [8] considered the trade-off between time and space complexities for two synchronous agents' gathering for the case that each agent has $k \geq c \log n$ memory bits (c is some constant). In this case, they proposed an algorithm to solve the gathering problem in $O(n+n^2/k)$ time, which is asymptotically optimal. Elouasbi and Pelc [9] considered the time complexity trade-off between determinism and randomization. They proposed a deterministic algorithm for two synchronous agents' gathering in $O(n)$ time. On the other hand, when agents know the maximum degree of the tree and the upper bound of the initial distance between two agents, they proposed a randomized algorithm to achieve the two synchronous agents' gathering with high probability in $O(\log n)$ time.

Asynchronous gathering for two or more agents is considered in [10]. Baba et al. showed a lower bound of space complexity for time-optimal algorithms, that is, they showed that each agent requires $\Omega(n)$ memory bits to solve the gathering problem in $O(n)$ time. In addition, they proposed a space-optimal algorithm to solve the gathering problem on the condition that the time complexity is asymptotically optimal, that is, both the time complexity and the space complexity are $O(n)$.

1.3. Organization

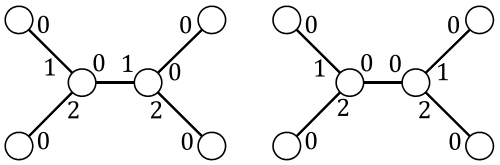
The paper is organized as follows. Section 2 presents the system model and the problem to be solved. In Section 3 we show the lower bound of total moves for the non-token model. In Section 4 we consider the first model, that is, the weak multiplicity detection and non-token model. In Section 5 we consider the second model, that is, the strong multiplicity detection and non-token model. In Section 6 we consider the third model, that is, the weak multiplicity detection and removable-token model. Section 7 concludes the paper.

2. Preliminaries

2.1. Network and agent model

A *tree network* T is a tuple $T = (V, L)$, where V is a set of nodes and L is a set of communication links. We denote by n $(= |V|)$ the number of nodes. Let d_v be the degree of v. We assume that nodes have no distinct IDs (i.e., are anonymous), but each link l incident to v is uniquely labeled at v with a label chosen from the set $\{0, 1, \ldots, d_v - 1\}$. We call this label *port number*. Since each communication link connects two nodes, it has two port numbers. However, port numbering is *local*, that is, there is no coherence between the two port numbers. The *path* $P(v_0, v_k) = (v_0, v_1, \ldots, v_k)$ with length k is a sequence of nodes from v_0 to v_k such that $\{v_i, v_{i+1}\} \in L$ $(0 \leq i < k)$ and $v_i \neq v_j$ if $i \neq j$. Note that, for any u, $v \in V$, $P(u, v)$ is unique in a tree. The *distance* from u to v is the length of the path from u to v. Next, we explain about center nodes. Let us consider the following sequence of trees constructed recursively as follows: $T_0 = T$ and T_{i+1} is obtained from T_i by removing all its leaves. Let j be the minimum value such that T_j has at most two nodes. Then, we call such nodes *center nodes*. We use the following theorem about center nodes later.

Theorem 1. [22] *There exist one or two center nodes in a tree. If there exist two center nodes, they are neighbors.* □



(a) An asymmetric tree (b) A symmetric tree

Fig. 1. Asymmetric and symmetric trees.

Next we define symmetry of trees, which is important to consider solvability in Section 4.

Definition 1. A tree T is symmetric iff there exists a function $\lambda : V \to V$ such that all the following conditions hold (see Fig. 1):

- For any $v \in V$, $v \neq \lambda(v)$ holds.
- For any $u, v \in V$, u is adjacent to v iff $\lambda(u)$ is adjacent to $\lambda(v)$.
- For any link $\{u, v\} \in L$, the port number assigned to $\{u, v\}$ at u is equal to the port number assigned to link $\{\lambda(u), \lambda(v)\}$ at $\lambda(u)$.

When tree T is symmetric, we say nodes u and v in T are symmetric if $u = \lambda(v)$ holds. When tree T is not symmetric, we say tree T is asymmetric. □

There exist k agents on tree T, and let $A = \{a_1, a_2, \ldots, a_k\}$ be the set of the agents. We assume that agents know neither n nor k. We consider the *strong multiplicity detection model* and the *weak multiplicity detection model*. In the strong multiplicity detection model, each agent can count the number of agents at the current node. In the weak multiplicity detection model, each agent can recognize whether another agent stays at the same node or not, but cannot count the number of agents at its current node. In both models, each agent cannot read the state of any other agent. Moreover, we consider the *non-token model* and the *removable-token model*. In the non-token model, agents cannot mark the nodes or the edges in any way. In the removable-token model, each agent initially leaves a token on its initial node at the beginning of the algorithm, and agents can remove any owner's token during the execution of the algorithm.

We assume that agents are anonymous (i.e., agents have no IDs) and execute a deterministic algorithm. We model an agent as a finite state machine $(S, \delta, s_{initial}, s_{final})$. The first element S is the set of all states of agents, which includes initial state $s_{initial}$ and final state s_{final}. When an agent changes its state to s_{final}, the agent terminates the algorithm. The second element δ is the state transition function. In the weak multiplicity detection and non-token model, δ is described as $\delta : S \times M_T \times R_A \to S \times M_T$. In the definition, set $M_T = \{\perp, 0, 1, \ldots, \Delta - 1\}$ represents the agent's movement, where Δ is the maximum degree of the tree. In the left side of δ, the value of M_T represents the port number assigned at the current node to the link the agent used in entering the current node (The value is $\perp$ in the first activation). In the right side of δ, the value of M_T represents the port number through which the agent leaves the current node to visit the next node. If the value is $\perp$, the agent does not move and stays at the current node. In this case, if the value of R_A (explained in the next sentence) changes from the previous movement, then the agent may change the value of M_T and leave the current node. In addition, $R_A = \{0, 1\}$ represents whether another agent stays at the current node or not. The value 0 represents that no other agents stay at the current node, and the value 1 represents that another agent stays at the current node.

In the strong multiplicity detection and non-token model, δ is described as $\delta : S \times M_T \times \{0, 1, \ldots, k-1\} \to S \times M_T$. In the definition, $\{0, 1, \ldots, k-1\}$ represents the number of other agents at the current node. In the weak multiplicity detection and removable-token model, δ is described as $\delta : S \times M_T \times R_A \times R_T \to S \times R_T \times M_T$. In the definition, in the left side of δ, $R_T = \{0, 1\}$ represents whether a token exists at the current node or not. The value 0 of R_T represents that there does not exist a token at the current node, and the value 1 of R_T represents that there exists a token at the current node. In the right side of δ, $R_T = \{0, 1\}$ represents whether the agent removes a token at the current node or not. If the value of R_T in the left side is 1 and the value of R_T in the right side is 0, it means that the agent removes a token at the current node. Otherwise, it means that an agent does not remove a token at the current node. Note that, in both models, we assume that each agent is not imposed any restriction on the memory.

During the execution of the algorithm, agents are located either on nodes or links. Each agent executes the following three operations in an atomic *step*: 1) Agent a_h reaches some node v, 2) agent a_h executes local computation at v, and 3) agent a_h leaves v or stays there. The local computation in the second action and the decision for the third action are determined by the state transition function δ. In the local computation, agent a_h executes the following operations: 1) Agent a_h obtains information about its local configuration (i.e., existence (resp. the number) of other agents at the current node v in the weak (resp. strong) multiplicity model and the token state at v in the removable-token model), 2) agent a_h executes some computation at v, 3) agent a_h decides whether a_h removes the token or not for the case of the removable-token model, 4) agent a_h decides whether a_h moves to the next node or not, and 5) agent a_h decides the port number to leave from (in the case that it decides to move). We assume a_h completes possible local computation at each step, that is, at

the end of a step, a_h either leaves v or decides to stay at v. If a_h decides to stay at v, after the decision a_h does nothing (i.e., does not change its state, does not remove the token at v, or does not leave v) unless other agents change a_h's local configuration. Note that the above atomic actions can be easily implemented if each node has a *buffer* that stores agents visiting the node and makes them execute processes in a FIFO order, and this assumption is very natural in a distributed system. In addition we assume that agents move in the tree network in a FIFO manner, that is, when agent a_h leaves some node v_j before another agent a_i leaves v_j through the same communication link as a_h, then a_h reaches v_j's neighboring node v'_j before a_i. Note that a FIFO assumption is known to be also natural in a distributed system.

2.2. System configuration

In the non-token model, (global) *configuration* c is defined as a product of states of agents, states of links, and locations of agents. Here, the state of link (v_j, v'_j) is a sequence of agents that are in transit from v_j to v'_j in this order. In the removable-token model, configuration c is defined as a product of states of agents, states of nodes (existence or nonexistence of tokens), states of links, and locations of agents. Note that in both models, the locations of agents are either on nodes or links. In addition, in the initial configuration c_0, we assume that node v_j has a token if there exists an agent at v_j, and v_j does not have a token if there exists no agent at v_j. Moreover in both models, we assume that no pair of agents stay at the same node in the initial configuration c_0, thus exactly k district nodes each have a token in c_0.

When configuration c_i changes to c_{i+1}, a *scheduler* activates a non-empty set of agents, say A_i, and each agent in A_i takes a step as mentioned before. We denote by such a transition $c_i \xrightarrow{A_i} c_{i+1}$. We assume that the scheduler is *fair*, that is, each agent is activated after a finite (unknown) amount of time and infinitely many times. In addition, we assume that if the scheduler activates some agent a_j that is 1) in a sequence of agents that are in transit in some link (v_l, v'_l), but 2) not in the head of the sequence, then a_j does not take a step (i.e., does not reach v'_l). Moreover, if several agents at the same node are included in A_i, the scheduler activates the agents one by one in an arbitrary order. When $A_i = A$ holds for every i, all agents take steps every time. This model is called the *synchronous model*. Otherwise, the model is called the *asynchronous model*. In this paper, we consider the asynchronous system.

If sequence of configurations $E = c_0, c_1, \ldots$ satisfies $c_i \xrightarrow{A_i} c_{i+1}$ $(i \geq 0)$, E is called an *execution* starting from c_0. We assume that any execution E is maximal in the sense that E is infinite, or ends in final configuration c_{final} where every agent's state is s_{final}.

2.3. Partial gathering problem

The requirement of the partial gathering problem is that, for a given positive integer g, each agent should move to a node and terminate so that at least g agents should meet at the node. Formally, we define the g-partial gathering problem as follows.

Definition 2. Execution E solves the g-partial gathering problem when the following conditions hold:

- Execution E is finite.
- In the final configuration, for any node v_j such that there exists an agent on v_j, there exist at least g agents on v_j. □

For the g-partial gathering problem, we have the following lower bound on the total number of agents moves. Notice that the lower bound result holds in any multiplicity detection model and any communication model (e.g., the removable-token model).

Theorem 2. *The total number of moves required to solve the g-partial gathering problem for tree networks is $\Omega(gn)$ if $g \geq 2$.*

Proof. We assume that $k = \lfloor n/2 \rfloor$ holds. Let us consider a line network such that $\lfloor g/2 \rfloor$ agents are placed at consecutive nodes starting from one endpoint and the other $k - \lfloor g/2 \rfloor$ agents are placed at consecutive nodes starting from the other endpoint. We call the $\lfloor g/2 \rfloor$ agents (resp., $k - \lfloor g/2 \rfloor$ agents) group G_1 (resp., G_2). Then, agents in G_1 need to meet at the same node with agents in G_2. Let $a_l \in G_1$ (resp., $a_r \in G_2$) be the agent located at the farthest node from the endpoint that an agent in G_1 (resp., G_2) exists. Then, the distance between a_l and a_r is at least $\lfloor n/2 \rfloor$, and hence they requires at least $\lfloor n/2 \rfloor$ moves to meet at the same node v'. Similarly, in order for one agent in G_1 and one agent in G_2 to meet at v', they require at least $\lfloor n/2 \rfloor$ moves. This requires $\lfloor g/2 \rfloor \times \lfloor n/2 \rfloor = \lfloor gn/4 \rfloor$ moves. Thus, we have the theorem. □

3. Lower bound of the total moves for the non-token model

For the non-token model, we have the following lower bound of the total moves. This result holds even for the strong-multiplicity detection model.

Theorem 3. *In the non-token model, agents require $\Omega(kn)$ total moves to solve the g-partial gathering problem even if agents know k.*

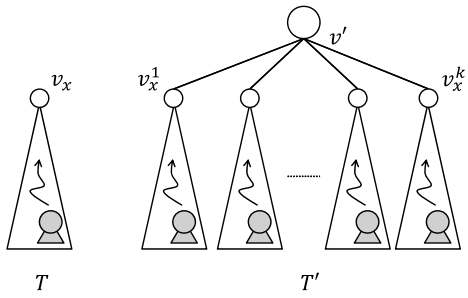


Fig. 2. T and T'.

Proof. For simplicity, we show the theorem for the case of the weak-multiplicity model, and the following proof can be similarly applied to the strong-multiplicity model. We show the theorem by contradiction, that is, we assume that there exists an algorithm A to solve the g-partial gathering problem in $o(kn)$ total moves. Let *a local configuration* of agent a staying at node v be a boolean value indicating whether another agent stays at v or not. Then, we define *a waiting state* of agents as follows: an agent a is in the waiting state at node v if a never leaves v before the local configuration of a changes. Concretely, there are two cases. The first case is that, when a visits node v and enters a waiting state at v, there exist no other agents at v. In this case, a neither changes its waiting state nor leaves v until another agent visits v. When the scheduler activates a and a observes such an agent, a can break its waiting state and leave v. The second case is that, when a visits v and enters a waiting state at v, there exists another agent at v. In this case, a neither changes its waiting state nor leaves v until there are no other agents at v. When the scheduler activates a and a detects such a situation, a can break its waiting state and can leave v.[1]

Let us consider the initial configuration c_0 such that k agents are placed in tree T with n nodes. We claim that some agent enters a waiting state in $o(n)$ moves without meeting other agents. Consider the execution that repeats a phase in which every agent not in a waiting state: 1) makes a movement, and 2) visits a node. Let a_i be the first agent that enters a waiting state in this execution. Then, a_i does not meet other agents until it enters a waiting state. This is because, unless each agent enters a waiting state, it moves in the tree and is never observed by other agents. If a_i makes $\Omega(n)$ moves before it enters a waiting state, each of the other agents makes $\Omega(n)$ moves. This implies the total number of moves is $\Omega(kn)$, which contradicts to the assumption of A. Hence, a_i enters a waiting state in $o(n)$ moves without meeting other agents. This implies there exists a node v_x which a_i does not visit before it enters a waiting state. Let v_w be the node where a_i is placed in the initial configuration c_0.

Next, we construct tree T' with $kn'+1$ nodes as follows: Let $T^1, \ldots, T^k$ be k trees with the same topology as T and v_x^j $(1 \le j \le k)$ be the node in T^j corresponding to v_x in T. Tree T' is constructed by connecting a new node v' to v_x^j for every j (Fig. 2). Let v_w^j $(1 \le j \le k)$ be the node in T^j corresponding to v_w in T. Consider the configuration c_0' such that k agents are placed at $v_w^1, v_w^2, \ldots, v_w^k$, respectively. Since agents do not have knowledge of n, each agent performs the same behavior as a_i in T (note that they do not visit v_x^j). Hence, each agent placed in T^j $(1 \le j \le k)$ enters a waiting state without moving out of T^j. Thus, each agent enters a waiting state at different nodes and does not resume its execution. Therefore, algorithm A cannot solve the g-partial gathering problem in T'. This is a contradiction. □

4. Weak multiplicity detection and non-token model

In this section, we consider the g-partial gathering problem for Model 1 in Table 1, that is, the weak multiplicity detection and non-token model. First, we consider the case for asymmetric trees. In this case, agents can achieve the classical gathering in $O(kn)$ total moves by the previous result in [10]. This result can be clearly applied to the g-partial gathering. Hence, we have the following theorem.

Theorem 4. *In the weak multiplicity detection and non-token model, agents solve the g-partial gathering problem in $O(kn)$ total moves for asymmetric trees.* □

Next, we consider the case that the tree is symmetric and agents are placed symmetrically in the initial configuration. In this case, we show that there exist no algorithms to solve the g-partial gathering problem if $g \ge 5$ holds. We consider the case such that in the initial configuration even agents are placed symmetrically in a symmetric tree, that is, if there exists an agent at node v, there also exists an agent at node v', where v and v' are symmetric. Then, we have the following theorem.

[1] The final state of an agent after gathering is a waiting state. Hence, the final state is a kind of the waiting state.

Theorem 5. *Let us consider the initial configuration such that agents are placed symmetrically in a symmetric tree. Then, in the weak multiplicity detection and non-token model, there exist no algorithms to solve the g-partial gathering problem if $g \geq 5$ holds.*

Proof. For contradiction, we assume that the g-partial gathering problem can be solved. We prove the theorem for the case that g is an odd number (we can also prove the theorem similarly for the case that g is an even number). We assume that the tree network is symmetric, and for any node v, we denote by v' the node symmetric to v. We consider the initial configuration c_0 such that $3g-1$ agents are placed symmetrically in the symmetric tree, that is, if there exists an agent at v, there also exists an agent at v'. For any agent a located at a node v in c_0, let a' denote the agent that is located at v' in c_0. Note that since $2g < k = 3g-1 < 3g$ holds, agents are allowed to meet at one or two nodes. Then, we have the following lemma [6].

Lemma 1. *Assume that each pair of nodes v_1 and v'_1, v_2 and v'_2, ... v_m and v'_m is symmetric in tree T. If agents a_i and a'_i $(1 \leq i \leq m)$ start an algorithm from v_i and v'_i, respectively, there exists an execution in which each pair acts in a symmetric manner even in an asynchronous model.* □

We consider a waiting state defined in Section 3. Then, the definition means that even when the local configuration of some waiting agent changes, the agent does not change its state unless the scheduler activates the agent. Note that, if an agent is staying at some node, then it is either in an initial state or a waiting state. Then, we have the following lemma about a waiting state.

Lemma 2. *At any node v_j where at least three waiting agents exist, at least two of the agents never leave v_j by the end of the algorithm.*

Proof. We assume that agents a^j_1, a^j_2, a^j_3 enter waiting states at v_j in this order. Since a^j_1 is the first agent that enters a waiting state at v_j, when a^j_2 enters a waiting state at v_j, the local configuration of a^j_1 changes, and a^j_1 can leave v_j. Since we consider the weak multiplicity detection model, even if a^j_1 leaves v_j, a^j_2 and a^j_3 cannot detect the fact and local configurations of a^j_2 and a^j_3 do not change. Thus, agents a^j_2 and a^j_3 never leave v_j. □

Let us consider a configuration such that there exist at least three nodes where there exist at least three waiting agents, respectively. We call such a configuration a *three-node three-waiting-agent configuration*. Then in three-node three-waiting-agent configurations, by Lemma 2 there exist at least three nodes where agents exist at the end of the algorithm execution. In addition since agents are allowed to meet at one or two nodes because of $k < 3g$, agents cannot solve the g-partial gathering problem when the system reaches a three-node three-waiting-agent configuration. This is the key idea of the proof. We consider an adversarial scheduler such that once some agent enters a waiting state, the scheduler never activates the agent until all agent enter waiting states. When all agents are in waiting states, we denote such a configuration by c_t. Note that c_t is the configuration such that all agents' states are waiting states and each agent enters a waiting state exactly once. Then, the outline of the proof is described as follows. At first, we construct configuration c_t by considering the adversarial scheduler. Then, we consider the placement of waiting agents in c_t and show the unsolvability in any placement. If c_t is a three-node three-waiting-agent configuration or a configuration such that there exists at most one waiting agent at each node, we can clearly show that agents cannot solve the g-partial gathering problem. Otherwise, we show that, in any placement of waiting agents in c_t, there exists an execution by an adversarial scheduler such that the system reaches either 1) a three-node three-waiting-agent configuration, 2) a configuration such that there exists at most one waiting agent at each node, or 3) a configuration such that there exist two nodes with agents but there exist at most $g-1$ waiting agents at one of them.

At first, we consider the execution until the system reaches the first configuration c_t such that all agents are in waiting states. We consider an execution E_t under the following fair scheduler α_t that makes agents' movements as follows. First, α_t activates all agents once. This makes all agents leave their initial nodes and be in transit; otherwise all agents enter waiting states at their initial nodes and cannot solve the g-partial gathering problem. Next, α_t selects an agent a among heads of FIFO sequences of transiting agents in links, and activates a and a' at the same time where a' is an agent whose initial node is symmetric to that of a. By the definition of an atomic step, after a and a' visit nodes and execute local computation, they enter waiting states or leave the nodes. Similarly, α_t continues to activate a pair of such symmetric transiting agents at the same time. Eventually, all agents enter waiting states and they reach c_t. Note that, in any algorithm, each agent necessarily enters a waiting state in finite time (otherwise, if an agent never enters a waiting state, the agent moves in the tree network forever). Hence, scheduler α_t is fair because the system reaches configuration c_t in finite number of agents' steps. Then, since agents are initially placed symmetrically and move symmetrically, it follows that if there exist l waiting agents at a node v in c_t, there also exist l waiting agents at node v'. Thus we can denote the nodes where agents exist in c_t by $v_1, \ldots, v_s, v'_1, \ldots, v'_s$. In addition, let N_l (resp., N'_l) be the number of waiting agents at v_l (resp., v'_l) in c_t. Clearly, $N_l = N'_l$ $(1 \leq l \leq s)$ and $N_1 + N_2 + \cdots + N_s = k/2$ hold. Without loss of generality, we assume that $N_1 \geq N_2 \geq \cdots \geq N_s$ holds. Moreover, we assume that agents $a^j_1, a^j_2, \ldots, a^j_{N_j}$ (resp., $a^{j'}_1, a^{j'}_2, \ldots, a^{j'}_{N'_j}$) enter waiting states at v_j (resp., v'_j) in this

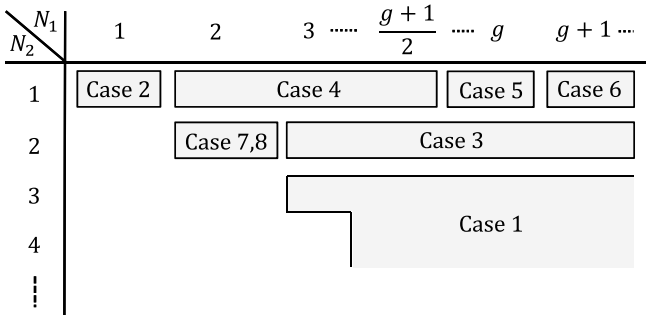


Fig. 3. Classification depending on values of N_1 and N_2 ($N_1 \geq N_2$).

order. We consider the following eight cases depending on values of $N_1, N_2, \ldots, N_s$ ($N'_1, N'_2, \ldots, N'_s$), and show that agents cannot solve the g-partial gathering problem in any case (contradiction). Fig. 3 represents the classification depending on values of N_1 and N_2. In addition, Case 7 considers $N_1 = N_2 = 2$ and $N_3 = 1$, and Case 8 considers $N_1 = N_2 = N_3 = 2$.

⟨Case 1: $N_2 \geq 3$ holds.⟩
In this case, there exist at least three waiting agents at each of v_1, v_2, v'_1 and v'_2 (three-node three-waiting-agent configuration). Hence from Lemma 2, there exist at least four nodes where agents exist at the end of algorithm execution. However, since $k = 3g - 1$ holds, agents are allowed to meet at one or two nodes. This contradicts the assumption that agents can solve the g-partial gathering problem.

⟨Case 2: $N_1 = N_2 = \cdots = N_s = 1$ holds.⟩
In this case, there exist no nodes where more than one agents exist in c_t. From the definition of a waiting state, the local configuration of each agent does not change and each agent never leaves the current node. This contradicts the assumption.

Before considering Case 3, we introduce the notion of elimination. Let us select a set of agents A_{elimi} such that both $|A_{elimi}| \leq g - 1$ and $A_{elimi} \subseteq \{a_i^j | 1 \leq j \leq s, 2 \leq i \leq N_j\} \cup \{a_i^{j'} | 1 \leq j' \leq s, 2 \leq i \leq N'_j\}$ hold. In addition, let c_0^{elimi} be the configuration obtained from c_0 by eliminating all agents in A_{elimi} in c_0. Moreover we define an execution E_t^{elimi} as follows: When in E_t the scheduler activates sets of agents $A_0, A_1, \ldots, A_{t-1}$ in this order and the system reaches c_t, then in E_t^{elimi} the scheduler activates sets of agents $A_0 - A_{elimi}, A_1 - A_{elimi}, \ldots, A_{t-1} - A_{elimi}$ in this order and the system reaches c_t^{elimi}. Then, we have the following lemma.

Lemma 3. *The locations and states of agents in $A - A_{elimi}$ in c_t^{elimi} are the same as those in c_t.*

Proof. We prove the lemma for the case of $|A_{elimi}| = 1$. Then, we can similarly prove the lemma for the case $|A_{elimi}| \geq 2$ by applying the following argument to each of A_{elimi} one by one. Let a_i^j ($2 \leq i \leq N_j$) be the unique agent in A_{elimi}. In this case, we show that the locations and states of agents in $A - A_{elimi}$ in c_l^{elimi} ($0 \leq l \leq t$) are equal to those in c_l. At first, we denote by c_p the configuration in E_t immediately after a_i^j enters a waiting state at v_j. Note that a_i^j enters a waiting state without being observed by any other agents. This is because 1) multiple agents do not exist at the same node in the initial configuration, 2) by the definition of scheduler α_t, a_i^j leaves its initial node before any other agents visit the node, and 3) until c_p, a_i^j reaches some node v, executes local computation, and leaves v in an atomic step, that is, a_i^j never waits at any node before c_p. In addition, in c_p there already exist waiting agents $a_1^j, \ldots, a_{i-1}^j$. Moreover, we denote by c_q ($p < q$) the configuration in which some agent a visits v_j for the first time after c_p.

Now let us consider E_t^{elimi}. First we can show that, except for a_i^j, the locations and states of agents in each of $c_0^{elimi}, c_1^{elimi}, \ldots c_p^{elimi}$ in E_t^{elimi} are the same as those in each of $c_0, c_1, \ldots, c_p$ in E_t. This is because in E_t, a_i^j moves without being observed by any other agents. Similarly, we can show that the locations and states of agents in each of $c_{p+1}^{elimi}, \ldots, c_{q-1}^{elimi}$ are the same as those except for a_i^j in each of $c_{p+1}, \ldots, c_{q-1}$. Next, we consider the locations and states of agents in c_q^{elimi}. In c_q^{elimi}, some agent a visits v_j and then there exist $i - 1$ waiting agents $a_1^j, \ldots, a_{i-1}^j$ at v_j. On the other hand in c_q, there exist waiting agents $a_1^j, \ldots, a_i^j$ at v_j. Then, agent a cannot distinguish the difference between c_q and c_q^{elimi} because $i \geq 2$ holds and we consider the weak multiplicity detection model. Thus, agent a behaves in the same way as in E_t and the locations and states of agents in c_q^{elimi} are the same as those in c_q, except for a_i^j.

In the following, we show by induction that the locations and states of agents in each of $c_{q+1}^{elimi}, \ldots, c_t^{elimi}$ are the same as those except for a_i^j in each of $c_{q+1}, \ldots, c_t$. We assume that the locations and sates of agents in each of c_r^{elimi} ($q+1 \leq r \leq t-1$) are the same as those except for a_i^j in each of c_r. Then, in c_t^{elimi} if there exists no agent that visits v_j, the locations and

states of agents in c_{r+1}^{elimi} in E_{r+1}^{elimi} are the same as those in each of c_{r+1} in E_t. This is because between c_r and c_{r+1} in E_t, a_i^j stays at v_j and it is never observed by agents except for agents already staying at v_j. In c_{r+1}^{elimi} if there exists some agent a that visits v_j, there exist i' ($i' \geq i$) waiting agents at v_j. Then, agent a cannot distinguish the difference between c_{r+1} and c_{r+1}^{elimi} because $i' \geq 2$ holds and we consider the weak multiplicity detection model. Hence, agent a behaves in the same way as in E_t and the locations and states of agents in c_{r+1}^{elimi} are the same as those in c_{r+1}, except for a_i^j. Thus, we can show that the locations and states of agents in each of $c_{q+1}^{elimi}, \ldots, c_t^{elimi}$ are also the same as those except for a_i^j in each of $c_{q+1}, \ldots, c_t$. Therefore, the locations and states of agents in $A - A_{elimi}$ in c_t^{elimi} are equal to those in c_t, and we have the lemma. □

By Lemma 3 and the fact that in c_t all agents are in waiting states, we can clearly show that in c_t^{elimi} all agents are in waiting states. We use this lemma to show the contradiction in the remaining cases.[2]

⟨Case 3: $N_1 \geq 3$ and $N_2 = 2$ hold.⟩
In this case, there exist three waiting agents a_1^1, a_2^1, and a_3^1 ($a_1^{1'}, a_2^{1'}$, and $a_3^{1'}$, respectively) at v_1 (v'_1), and agents a_2^1 and a_3^1 ($a_2^{1'}$ and $a_3^{1'}$, respectively) never leave v_1 (v'_1) by Lemma 2. Since $k = 3g - 1$ holds and agents are allowed to meet at one or two nodes, all agents must meet at v_1 or v'_1.

Now let us consider the initial configuration c_0^{elimi} obtained from c_0 by eliminating agents a_2^2 and $a_2^{2'}$. Then from Lemma 3, there exists an execution E_t^{elimi} from c_0^{elimi} to c_t^{elimi}, where there exists exactly one waiting agent a_1^2 ($a_1^{2'}$) at v_2 (v'_2) in c_t^{elimi}. In this configuration, agents a_1^2 and $a_1^{2'}$ need to meet at v_1 or v'_1. To do this, it is necessary that some agent enters a waiting state at v_2 and v'_2 in order to make a_1^2 and $a_1^{2'}$ observe changes of local configurations and leave there. We consider an execution E_x^{elimi} under the scheduler α_x^{elimi} deciding agents and their behavior as follows. Let $b_1, \ldots, b_h$ ($b'_1, \ldots, b'_h$) be the sequence of agents such that 1) b_1 (b'_1) is an agent that can leave the current node in c_t^{elimi}, 2) b_i (b'_i) ($2 \leq i \leq h-1$) is an agent in the waiting state at some node v_{bi} (v'_{bi}) where no other agents exist (note that b_i can leave v_{bi} when b_{i-1} arrives at v_{bi} and enters a waiting state), and 3) b_h (b'_h) is an agent in the waiting state at v_2 (v'_2), that is, $b_h = a_1^2$ ($b'_h = a_1^{2'}$). Then in α_x^{elimi}, agents b_j and b'_j ($1 \leq j \leq h-1$) are activated at the same time, and behave symmetrically. Finally, agents b_{h-1} and b'_{h-1} enter waiting states at v_2 and v'_2, respectively. We call such a configuration c_x^{elimi}. An example is shown in Fig. 4. In the figure, we assume that agents a_2^2 and $a_2^{2'}$ of the dotted lines are eliminated. In addition, the black agents $a_2^1, a_3^1, a_2^{1'}$, and $a_3^{1'}$ never leave the current nodes by the end of the algorithm. In Fig. 4, agents a_1^1 and $a_1^{1'}$ move symmetrically and enter waiting states at v_3 and v'_3, respectively (Fig. 4 (b)). After this, agents a_1^3 and $a_1^{3'}$ move symmetrically and enter waiting states at v_2 and v'_2, respectively (Fig. 4 (c) to Fig. 4 (d)).

Now, let us consider c_t. In c_t, there exist two waiting agents a_1^2 and a_2^2 ($a_1^{2'}$ and $a_2^{2'}$, respectively) at v_2 (v'_2). In addition, since a_1^2 ($a_1^{2'}$) is the first agent that enters a waiting state at v_2 (v'_2), a_1^2 ($a_1^{2'}$) can leave v_2 (v'_2). However we consider the execution E_x similarly to E_x^{elimi}, that is, agents b_1 and b'_1, b_2 and b'_2, ... , b_{h-1} and b'_{h-1} are activated and behave symmetrically in this order, while agents a_1^2 and $a_1^{2'}$ are not activated. Finally, agents b_{h-1} and b'_{h-1} enter waiting states at v_2 and v'_2, respectively. We call such a configuration c_x.[3] Then in c_x, there exist three waiting agents a_1^2, a_2^2, and b_{h-1} ($a_1^{2'}, a_2^{2'}$, and b'_{h-1}, respectively) at v_2 (v'_2), and agents a_2^2 and b_{h-1} ($a_2^{2'}$ and b'_{h-1}, respectively) never leave v_2 (v'_2) by Lemma 2. For example in Fig. 4, agents a_1^1 and $a_1^{1'}$ move symmetrically and enter waiting states at v_3 and v'_3, respectively (Fig. 4 (e) to Fig. 4 (f)). After this, agents a_1^3 and $a_1^{3'}$ move symmetrically and enter waiting states at v_2 and v'_2, respectively (Fig. 4 (g) to Fig. 4 (h)). Then there exist three waiting agents a_1^2, a_2^2, and a_1^3 ($a_1^{2'}, a_2^{2'}$, and $a_1^{3'}$, respectively) at v_2 (v'_2), and agents a_2^2 and a_1^3 ($a_2^{2'}$ and $a_1^{3'}$, respectively) never leave the current node by Lemma 2. Note that, agents $a_2^1, a_3^1, a_2^{1'}$ and $a_3^{1'}$ also never leave the current nodes v_1 and v'_1. Thus in c_x, there exist four nodes where agents exist and never leave the current nodes (three-node three-waiting-agent configuration), which is a contradiction.

From Case 4 to Case 6, we consider cases that there exist at least two waiting agents a_1^1 and a_2^1 ($a_1^{1'}$ and $a_2^{1'}$, respectively) at v_1 (v'_1), and there exists at most one waiting agent at the other nodes.

⟨Case 4: $2 \leq N_1 \leq (g+1)/2$ and $N_2 = 1$ hold.⟩
In this case, we consider the initial configuration c_0^{elimi} obtained from c_0 by eliminating agents $a_2^1, \ldots, a_{N_1}^1, a_2^{1'}, \ldots, a_{N'_1}^{1'}$. Note that, the number of eliminated agents $a_2^1, \ldots, a_{N_1}^1, a_2^{1'}, \ldots, a_{N'_1}^{1'}$ is $2N_1 - 2 \leq g - 1$ since $N_1 \leq (g+1)/2$ holds. Then from Lemma 3, there exists an execution E_t^{elimi} from c_0^{elimi} to c_t^{elimi}, where there exists at most one waiting agent at each node in c_t^{elimi}. This configuration is the same as the Case 2 and agents cannot solve the g-partial gathering problem.

[2] From Case 6 to Case 8, we consider a configuration obtained from c_0 by eliminating at least four agents, and we cannot apply this way for the case of $2 \leq g \leq 4$.

[3] Execution E_x is fair because the system reaches configuration c_x in finite number of agents' steps. Similarly, we can show that schedulers or executions we consider in the rest of this section are fair.

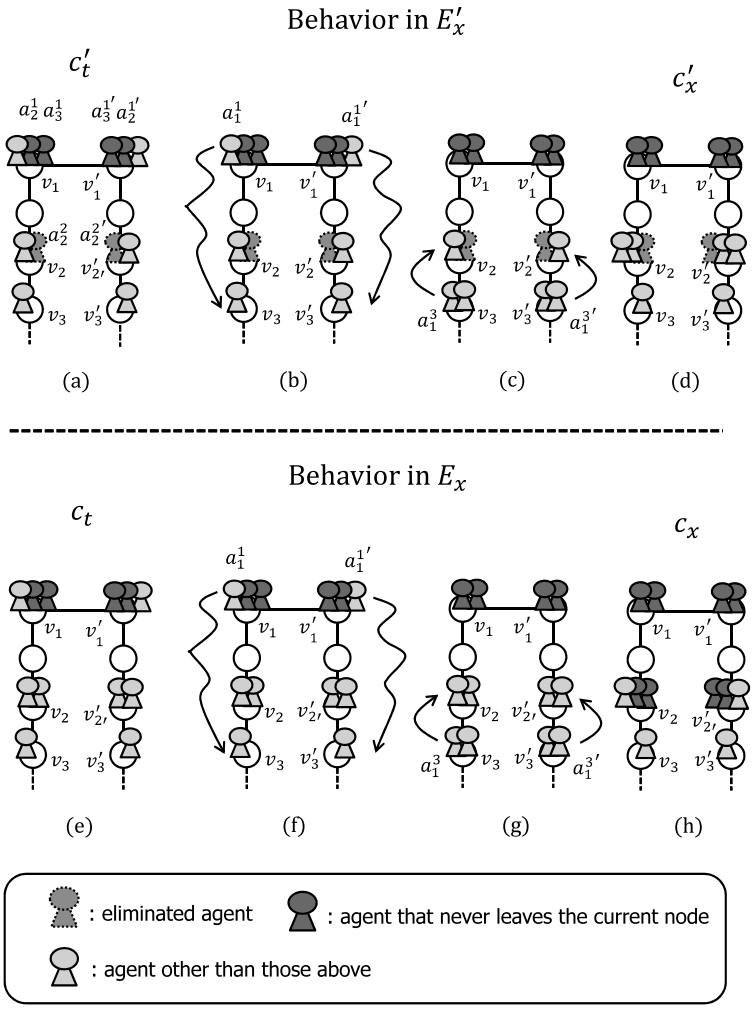


Fig. 4. An example of Case 3.

⟨Case 5: $(g+3)/2 \leq N_1 \leq g$ and $N_2 = 1$ hold.⟩
In this case, we consider the initial configuration c_0^{elimi} obtained from c_0 by eliminating agents $a_2^1, \ldots, a_{N_1}^1$. Note that, the number of eliminated agents $a_2^1 \ldots, a_{N_1}^1$ is $N_1 - 1 \leq g - 1$ since $N_1 \leq g$ holds. Then from Lemma 3, there exists an execution E_t^{elimi} from c_0^{elimi} to c_t^{elimi}, where there exist N'_1 waiting agents at v'_1 and at most one waiting agent at the other nodes in c_t^{elimi}. Since agents are allowed to meet at one or two nodes and only $a_1^{1'}$ can leave the current node v'_1 in this configuration, it is necessary that agent $a_1^{1'}$ firstly leaves v'_1 and enters a waiting state at some node where a waiting agent exists to make the waiting agent leave there. Without loss of generality, we assume that $a_1^{1'}$ enters a waiting state at v'_j where waiting agent $a_1^{j'}$ exists. We call such a configuration c_x^{elimi} and define E_x^{elimi} as an execution from c_t^{elimi} to c_x^{elimi}. Moreover after this, agents need to make the configuration such that some agent a' enters a waiting state at v_j in order to meet there or make agent a_1^j leave there. We call such a configuration c_y^{elimi} and define E_y^{elimi} as an execution from c_x^{elimi} to c_y^{elimi}. For example in Fig. 5, agent $a_1^{1'}$ moves and enters a waiting state at $v_{3'}$ (Fig. 5 (a) to Fig. 5 (b)), and after this, agent $a_1^{3'}$ moves and enters a waiting state at v_3 (Fig. 5 (c)).

Now let us consider c_t. In c_t, agents a_1^1 and $a_1^{1'}$ can leave the current nodes and the other agents cannot leave the current nodes. Then we consider an execution E_x under the fair scheduler α_x, where a_1^1 and $a_1^{1'}$ are activated at the same time, behave symmetrically and enter waiting states at v_j and v'_j, respectively. We call such a configuration c_x. Then, the local configurations of a_1^j and $a_1^{j'}$ change and they can leave v_j and v'_j, respectively. However, we consider the execution E_y similarly to E_y^{elimi}, that is, agent $a_1^{j'}$ leaves v'_j and some agent a' enters a waiting state at v_j, while a_1^j is not activated. Then there exist three waiting agents a_1^j, a_1^1, and a' at v_j, and agents a_1^1 and a' never leave v_j by Lemma 2. For example

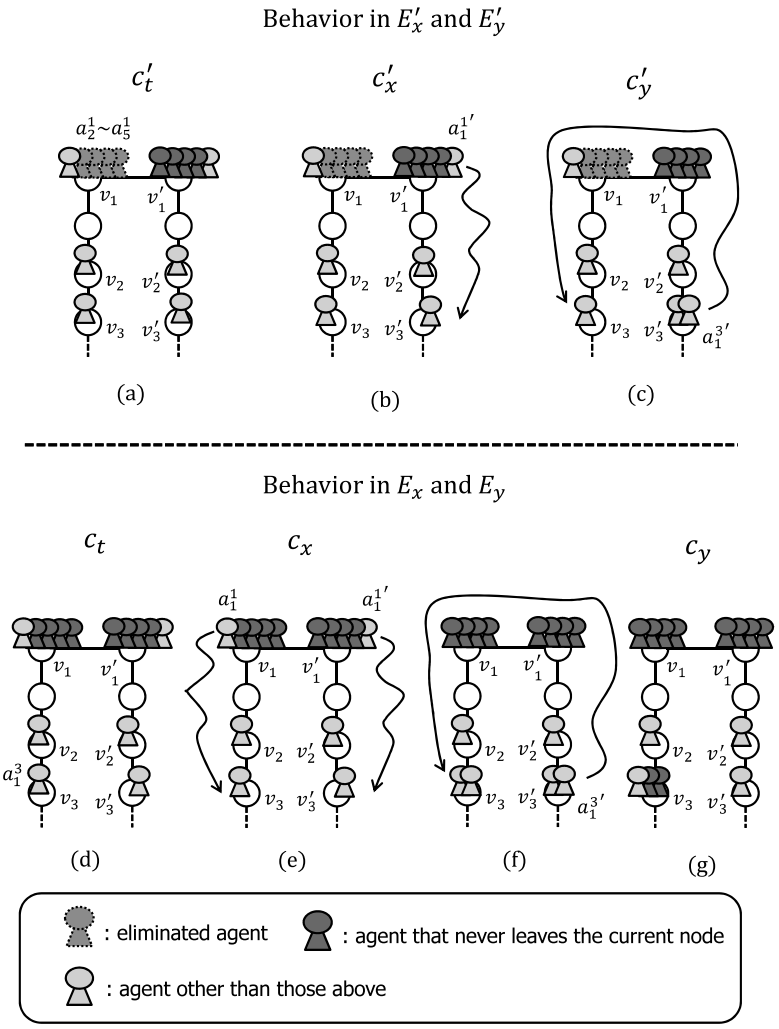


Fig. 5. An example of Case 5.

in Fig. 5, agents a_1^1 and $a_1^{1'}$ move and enter waiting states at v_3 and v'_3, respectively (Fig. 5 (d) to Fig. 5 (e)). After this, agent $a_1^{3'}$ leaves v'_3 and enters a waiting state at v_3 (Fig. 5 (f)). Then there exist three waiting agents a_1^3, a_1^1, and $a_1^{3'}$ at v_3, and agents a_1^1 and $a_1^{3'}$ never leave v_3. Note that, agents $a_2^1, a_3^1, a_2^{1'}$ and $a_3^{1'}$ also never leave the current nodes v_1 and v'_1. Thus in c_y, there exist three nodes where agents exist at the end of algorithm execution (three-node three-waiting-agent configuration), which is a contradiction.

⟨Case 6: $N_1 \geq g+1$ and $N_2 = 1$ hold.⟩
In this case, agents are allowed to meet at v_1 or v'_1. As a way to satisfy this, we consider an execution E_x from c_t to c_x, where each agent moves symmetrically until they enter waiting states at v_1 or v'_1 in c_x. Then, there exist $(3g-1)/2$ agents at v_1 and v'_1, respectively.

Now let us consider the initial configuration c_0^{elimi} obtained from c_0 by eliminating agents $a_4^1, \ldots, a_{4+(g+1)/2-1}^1$. Then from Lemma 3, there exists an execution E_t^{elimi} from c_0^{elimi} to c_t^{elimi}, where there exist $N_1 - (g+1)/2$ waiting agents at v_1, N'_1 $(= N_1)$ waiting agents at v'_1, and at most one waiting agent at the other nodes in c_t^{elimi}. Moreover we consider the execution E_x^{elimi} similarly to E_x, and we define c_x^{elimi} as the configuration that all agents meet at v_1 or v'_1. Then since $(g+1)/2$ agents $a_4^1, \ldots, a_{4+(g-1)/2-1}^1$ are eliminated, the number of agents that meet at v_1 is $(3g-1)/2-(g+1)/2 = g-1$. This contradicts that agents can solve the g-partial gathering problem.

In the Cases 7 and 8, we consider the case that there exist at most two waiting agents at each node.

⟨Case 7: $N_1 = N_2 = 2$ and $N_3 = 1$ hold.⟩
In this case, there are two waiting agents at v_1, v_2, v'_1, and v'_2, and at most one waiting agent at the other nodes in c_t. Now we consider the initial configuration c_0^{elimi} obtained from c_0 by eliminating agents $a_2^1, a_2^2, a_2^{1'}$, and $a_2^{2'}$. Then from Lemma 3,

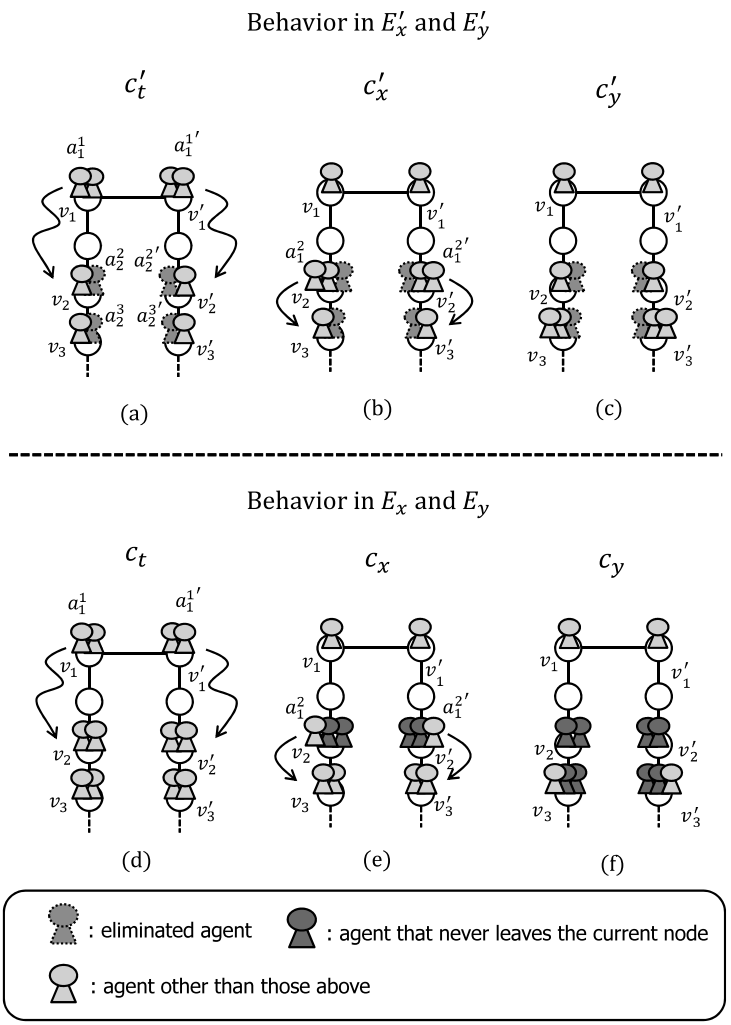


Fig. 6. An example of Case 8.

there exists an execution E_t^{elimi} from c_0^{elimi} to c_t^{elimi}, where there exists at most one waiting agent at each node in c_t^{elimi}. This configuration is the same as the Case 2 and agents cannot solve the g-partial gathering problem.

⟨Case 8: $N_1 = N_2 = N_3 = 2$ holds.⟩
In this case, there are two waiting agents at $v_1, v_2, v_3, v'_1, v'_2$, and v'_3 in c_t. Now we consider the initial configuration c_0^{elimi} obtained from c_0 by eliminating agents $a_2^2, a_2^3, a_2^{2'}$, and $a_2^{3'}$. Then from Lemma 3, there exists an execution E_t^{elimi} from c_0^{elimi} to c_t^{elimi}, where there exist two waiting agents a_1^1 and a_2^1 ($a_1^{1'}$ and $a_2^{1'}$, respectively) at v_1 (v'_1) and one waiting agent at v_2, v_3, v'_2, and v'_3, respectively. In this configuration, it is necessary that some agent enters a waiting state at v_2, v_3, v'_2 and v'_3 in order to meet there or to make the waiting agents leave the current nodes. Without loss of generality, we assume that at first some agents enter waiting states at v_2 and v'_2, respectively. After this, we assume that some agents enter waiting states at v_3 and v'_3, respectively. To do this, we consider an execution E_x^{elimi} under the scheduler α_x^{elimi} similarly to Case 3. That is, there exist the sequence of agents $b_1, \ldots, b_h$ ($b'_1, \ldots, b'_h$) such that agent b_h (b'_h) is in the waiting state at v_2 (v'_2). Then in α_x^{elimi}, agents b_j and b'_j ($1 \le j \le h-1$) are activated at the same time, behave symmetrically, and enter waiting states at $v_{b(j+1)}$ and $v'_{b(j+1)}$, respectively. Remind that at node $v_{b(j+1)}$, there exists a waiting agent $b_{(j+1)}$. Then, local configurations of agents b_{j+1} and b'_{j+1} change. Finally, agents b_{h-1} and b'_{h-1} enter waiting states at v_2 and v'_2, respectively. We call such a configuration c_x^{elimi}. Then, local configurations of a_1^2 and $a_1^{2'}$ change and they can leave the current nodes. For example in Fig. 6, agent a_1^1 ($a_1^{1'}$) leaves v_1 (v'_1) and directly enters a waiting state at v_2 (v'_2) (Fig. 6 (a) to Fig. 6 (b)). Moreover after c_x^{elimi}, we consider an execution E_y^{elimi} under the scheduler α_y^{elimi} similarly to α_x^{elimi}, that is, there exists the sequence of agents $d_1, \ldots, d_i$ ($d'_1, \ldots, d'_i$) such that agent d_i (d'_i) is in the waiting state at v_3 (v'_3). Then in α_y^{elimi}, agents d_j and d'_j ($1 \le j \le i-1$) are activated at the same time, behave symmetrically, and enter waiting states at $v_{d(j+1)}$ and

$v'_{d(j+1)}$, respectively. Note that at node $v_{d(j+1)}$, we assume that there exists a waiting agent d_{j+1}. Then, local configurations of agents d_{j+1} and d'_{j+1} change. Finally, agents d_{i-1} and d'_{i-1} enter waiting states at v_3 and v'_3, respectively. We call such a configuration c_y^{elimi}. For example in Fig. 6, agent a_1^2 ($a_1^{2'}$) leaves v_2 (v'_2) and directly enters a waiting state at v_3 (v'_3) (Fig. 6 (b) to Fig. 6 (c)).

Now let us consider c_t. In c_t, agents $a_1^1, a_1^2, a_1^3, a_1^{1'}, a_1^{2'}$ and $a_1^{3'}$ can leave the current nodes. However we consider the execution E_x similarly to E_x^{elimi}, that is, agents b_1 and b'_1, b_2 and b'_2, ... , b_{h-1} and b'_{h-1} are activated and behave symmetrically in this order, while agents a_1^2 and $a_1^{2'}$ are not activated. Finally, agents b_{h-1} and b'_{h-1} enter waiting states at v_2 and v'_2, respectively. We call such a configuration c_x. Then there exist three waiting agents a_1^2, a_2^2, and b_{h-1} ($a_1^{2'}, a_2^{2'}$, and b'_{h-1}, respectively) at v_2 (v'_2), and a_2^2 and b_{h-1} ($a_2^{2'}$ and b'_{h-1}, respectively) never leave the current node. For example in Fig. 6, agent a_1^1 ($a_1^{1'}$) leaves v_1 (v'_1) and directly enters a waiting state at v_2 (v'_2) (Fig. 6 (d) to Fig. 6 (e)). Then there exist three waiting agents a_1^2, a_2^2, and a_1^1 ($a_1^{2'}, a_2^{2'}$, and $a_1^{1'}$, respectively) at v_2 (v'_2), and a_2^2 and a_1^1 ($a_2^{2'}$ and $a_1^{1'}$, respectively) never leave the current node. Moreover after this, we consider the execution E_y similarly to E_y^{elimi}, that is, agents d_1 and d'_1, d_2 and d'_2, ... , d_{i-1} and d'_{i-1} are activated and behave symmetrically in this order, while agents a_1^3 and $a_1^{3'}$ are not activated. Finally, agents b_{i-1} and b'_{i-1} enter waiting states at v_3 and v'_3, respectively. We call such a configuration c_y. Then there exist three waiting agents a_1^3, a_2^3, and d_{i-1} ($a_1^{3'}, a_2^{3'}$, and d'_{i-1}, respectively) at v_3 (v'_3), and a_2^3 and d_{i-1} ($a_2^{3'}$ and d'_{i-1}, respectively) never leave the current node. For example in Fig. 6, agent a_1^2 ($a_1^{2'}$) leaves v_2 (v'_2) and directly enters a waiting state at v_3 (v'_3) (Fig. 6 (e) to Fig. 6 (f)). Then there exist three waiting agents a_1^3, a_2^3, and a_1^2 ($a_1^{3'}, a_2^{3'}$, and $a_1^{2'}$, respectively) at v_3 (v'_3), and a_2^3 and a_1^2 ($a_2^{3'}$ and $a_1^{2'}$, respectively) never leave the current node. Thus in c_y there exist four nodes where agents exist at the end of algorithm execution (three-node three-waiting-agent configuration). This contradicts that agents can solve the g-partial gathering problem.

Therefore, we have the theorem. □

5. Strong multiplicity detection and non-token model

In this section, we consider a deterministic algorithm to solve the g-partial gathering problem for Model 2 in Table 1, that is, the strong multiplicity detection and non-token model. We propose a deterministic algorithm to solve the g-partial gathering problem in $O(kn)$ total moves. Recall that, in the strong multiplicity detection model, each agent can count the number of agents at the current node.

At the beginning, each agent performs *a basic walk* [9]. In the basic walk, each agent a_h leaves the initial node through the port 0. Later, when a_h visits a node v_j through the port p of v_j, a_h leaves v_j through the port $(p+1) \mod d_{v_j}$. The basic walk allows each agent to traverse the tree in the DFS-traversal. Hence, when each agent visits nodes $2(n-1)$ times, it visits all the nodes and returns to the initial node. Remind that nodes are anonymous and agents do not know the number n of nodes. However, if an agent records the topology of the tree it ever visits, it can detect that it visits all the nodes and returns to the initial node. Concretely, in the DFS-traversal, every time each agent a_h visits a node for the first time, it obtains the port number used to enter and pushes it a stack. When a_h leaves the current node through the port p, it compares p with the number p' in the head of the stack. If $p = p'$ holds, a_h removes p' from the stack and this means that a_h moves closer to its initial node. Otherwise, it means that a_h moves further from its initial node. When a_h visits some node and the stack becomes empty, it means that a_h returns to its initial node. Moreover, if there exists no port p incident to its initial node such that a_h does not leave its initial node through p, it can detect that it observed all the nodes in the tree.

The idea of the algorithm is as follows: First, each agent performs the basic walk until it obtains the whole topology of the tree. Next, each agent computes a center node of the tree and moves there to meet other agents. If the tree has exactly one center node, then each agent moves to the center node and terminates the algorithm. If the tree has two center nodes, then each agent moves to one of the center nodes so that at least g agents meet at each center node. Concretely, agent a_h first moves to the closer center node v_j. If there exist at most $g-1$ agents except for a_h, then a_h terminates the algorithm at v_j. Otherwise, a_h moves to another center node $v_{j'}$ and terminates the algorithm.

The pseudocode is described in Algorithm 1. We have the following theorem.

Theorem 6. *In the strong multiplicity detection and non-token model, agents solve the g-partial gathering problem in $O(kn)$ total moves.*

Proof. At first, we show the correctness of the algorithm. From Algorithm 1, if the tree has one center node, agents go to the center node and agents solve the g-partial gathering problem obviously. Otherwise, each agent a_h first moves to one of the center nodes. If there already exist g or more agents at the center node, a_h moves to the other center node. Since $k \geq 2g$ holds, agents can solve the g-partial gathering problem.

Next, we analyze the total number of moves. At first, agents perform the basic walk and record the topology of the tree. This requires at most $2(n-1)$ total moves for each agent. Next, each agent moves to one of the center nodes, and terminates

Algorithm 1 The behavior of active agent a_h (v_j is the current node of a_h).

Main Routine of Agent a_h

```
1: perform the basic walk until it obtains the whole topology of the tree
2: if there exists exactly one center node then
3:     go to the center node via the shortest path and terminate the algorithm
4: else
5:     go to the closest center node via the shortest path
6:     if there exist at most g − 1 agents except for a_h then
7:         terminate the algorithm
8:     else
9:         move to the other center node
10:        terminate the algorithm
11:    end if
12: end if
```

the algorithm. This requires at most $\frac{n}{2}+1$ moves for each agent. Hence, each agent requires $O(n)$ total moves. Therefore, agents require $O(kn)$ total moves. □

6. Weak multiplicity detection and removable-token model

In this section, we consider the g-partial gathering problem for Model 3 in Table 1, that is, the weak multiplicity detection and removable-token model. We show that agents can achieve the g-partial gathering in asymptotically optimal total moves (i.e., $O(gn)$) by using only one removable token of each agent. Recall that, in the removable-token model, each agent has a token. In the initial configuration, each agent leaves a token at the initial node. We define *a token node* (resp., *a non-token node*) as a node that has a token (resp., does not have a token). In addition, when an agent visits a token node, the agent can remove the token.

The idea of the algorithm is similar to [11], which considers the g-partial gathering problem for distinct agents (i.e. having IDs) in unidirectional ring networks with whiteboards. The algorithm in [11] consists of two parts: the leader election and leaders' instructions. In the first part, agents execute the leader agent election partially using their IDs and whiteboards. Then, there exist at least $g-1$ non-leader agents between two leader agents. In the second part, each leader agent moves in the ring and instructs non-leader agents which node they should meet at by using whiteboards. After this, non-leader agents move to their gathering nodes by the instruction. When applying the above idea to the model in this section, there exist two problems. The first is the difference of network topology, that is, [11] considers unidirectional ring networks but in this paper we consider tree networks. The second is the difference of agents' and nodes' ability, that is, in [11] agents have distinct IDs and each node has a whiteboard but in this paper agents have no IDs and each node is allowed to only have at most one removable token. The first problem is solved by embedding the unidirectional ring in the tree network, and we explain this in the next paragraph. The second problem is solved by the combination of port numbers and removable-tokens, and we explain this in Section 6.1 and 6.2.

Now, we explain the way to embed the ring from the tree network. Agents perform the basic walk and embed a unidirectional ring network in the tree network by the Euler tour technique. Concretely, letting $v_0, v_1, \ldots, v_{2(n-1)}$ $(= v_0)$ be the node sequence such that agent a_h visits the nodes in this order in the basic walk starting at v_0, we can regard that a_h moves in the unidirectional ring network with $2(n-1)$ nodes. Later, we call this ring *the virtual ring*. In the virtual ring, we define the direction from v_i to v_{i+1} as a *forward* direction, and the direction from v_{i+1} to v_i as a *backward* direction. For simplicity in the virtual ring, operations to an index of a node assume calculation under modulo $2(n-1)$, that is, $v_{(i+1) \mod 2(n-1)}$ is simply represented by v_{i+1}. In addition in the virtual ring, we define *the neighboring agent* of a_h as the first agent in a_h's forward (backward) direction, i.e., there exist no agents between them. Moreover, when a_h visits a node v_j through a port p of v_j from a node v_{j-1} in the virtual ring, agents also use p as the port number of (v_{j-1}, v_j) at v_j. For example, let us consider a tree in Fig. 7 (a). Agent a_h performs the basic walk and visits nodes a, b, c, b, d, b in this order. Then, the virtual ring of Fig. 7 (a) is shown in Fig. 7 (b). Each number in Fig. 7 (b) represents the port number through which a_h visits each node in the virtual ring. Next, we define a token node in a virtual ring. At the beginning of the algorithm, each agent a_h leaves its token node through the port 0 in the basic walk. Thus, when a_h visits some token node in the tree such that a_h leaves there through the port 0 in the next movement, that is, when a_h visit some token node v_j through the port $(d_{v_j}-1)$, a_h regards the node as the token node in the virtual ring. In Fig. 7 (a), if nodes a and b are token nodes, then in Fig. 7 (b), nodes a and b'' are token nodes. By this definition, a token node in the tree network is mapped to exactly one token node in the virtual ring. Thus, by performing the basic walk, we can regard that all agents move in the same virtual ring although agents start the algorithm at different nodes. This is because the virtual ring starting at some node in the tree is actually represented by a port sequence P, and the virtual ring starting at other nodes in the same tree can be represented by the cyclic transformation of P. In Fig. 7, the virtual ring starting at a_h's initial node is represented by 001020. On the other hand, the virtual ring starting at another token node b is represented by 000102, and this sequence can be also represented by the cyclic transformation of 001020. Moreover, in the virtual ring, each agent also moves in a FIFO manner, that is, when an agent a_h leaves some node v_j before another agent a_i, a_h arrives at v_{j+1} before a_i.

In the following section, we explain the algorithm on the virtual ring. Note that we can show the asymptotical equivalence in terms of total moves between a tree and a virtual ring, because a tree with n nodes is regarded as a virtual

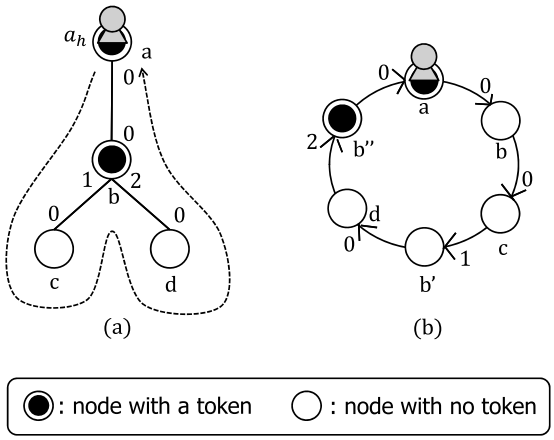


Fig. 7. An example of the basic walk.

ring with $2n-1$ nodes. The algorithm consists of two parts. In the first part, agents elect some leader agents by partially executing the leader agent election algorithm. In the second part, the leader agents instruct the other agents which node they should meet at, and the other agents move to the node.

6.1. The first part: leader election

In this section, we explain how to elect multiple leader agents. Note that, in this part no token is removed. In the leader agent election, each agent takes a state from the following three states:

- *active*: The agent is performing the leader agent election as a candidate for leaders.
- *inactive*: The agent has dropped out from the set of the leader candidates.
- *leader*: The agent has been elected as a leader.

The aim of the first part is similar to [11], that is, to elect some leaders and satisfy the following two properties: 1) At least one agent is elected as a leader, and 2) in the virtual ring, there exist at least $g-1$ inactive agents between two leader agents.

At first, we explain the idea of the leader election in [11] to adopt it in this paper. In [11], the network is a unidirectional ring, each agent is distinct, and each node has a whiteboard. First, we explain the idea under the assumption that the ring is bidirectional for intuitively understanding. Later, we apply the idea to the unidirectional ring. The algorithm consists of several phases. In each phase, each active agent compares its own ID with IDs of its forward and backward neighboring active agents. More concretely, each active agent writes its ID on the whiteboard of its current node, and then moves forward and backward to observe IDs of the forward and backward active agents. If its own ID is the smallest among the three agents, the agent remains active (as a candidate for leaders) in the next phase. Otherwise, the agent drops out from the candidate for the set of leader candidates and becomes inactive. Note that, in each phase, neighboring active agents never remain as candidates for leaders. Hence at least half of the currently active agents become inactive in each phase, that is, the number of inactive agents between two active agents at least doubles in each phase. Then from [23], after executing j phases, there exists at least 2^j-1 inactive agents between two active agents. Thus, after executing $\lceil \log g \rceil$ phases, the following properties are satisfied: 1) At least one agent remains as a candidate for leaders, and 2) the number of inactive agents between two active agents is at least $g-1$. Therefore, all the remaining active agents become leaders.

Next, we implement the above algorithm in asynchronous unidirectional rings by using a traditional approach [23]. Let us consider active agent a_h. In unidirectional rings, a_h cannot move backward or observe the ID of its backward active agent. Instead, a_h moves forward until it observes IDs of two active agents. Then, a_h observes IDs of three successive active agents. We assume a_h observes id_1, id_2, id_3 in this order. Note that id_1 is the ID of a_h. Here this situation is similar to that in which the active agent with ID id_2 observes id_1 as its backward active agent and id_3 as its forward active agent in a bidirectional ring. For this reason, a_h behaves as if it would be an active agent with ID id_2 in the bidirectional ring. That is, if id_2 is the smallest among the three IDs, a_h remains active as a candidate for leaders. Otherwise, a_h drops out from the set of leader candidates and becomes inactive.

In the following, we explain the way to apply the above leader election to anonymous agents in the weak multiplicity detection and removable-token model. First, we explain the treatment about IDs. For explanation, let *active nodes* be nodes where active agents start execution of each phase. In this section, agents use *virtual IDs* in the virtual ring. Concretely, when agent a_h moves from an active node v_j to v_j's forward active node $v_{j'}$, a_h observes port sequence $p_1, p_2, \dots p_l$, where p_m is the port number at v_{j+m} through which a_h visits the m-th node v_{j+m} after leaving v_j. In this case, a_h uses this port sequence $p_1, p_2, \dots p_l$ as its virtual ID. For example, in Fig. 7 (b), when a_h moves from a to b'', a_h observes the port

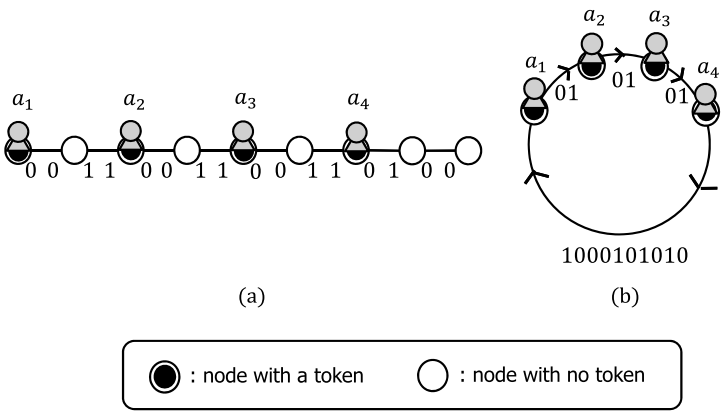


Fig. 8. An example that agents observe the same port sequence.

numbers $0, 0, 1, 0, 2$ in this order. Hence, a_h uses 00102 as a virtual ID from a to b''. Similarly, a_h uses 0 as a virtual ID from b'' to a. Note that, multiple agents may have the same virtual IDs, and we explain the behavior in this case later. Next, we explain the treatment of whiteboards by using removable tokens. Fortunately, we can easily overcome this problem if agents can detect active nodes. Concretely, each active agent a_h moves until a_h visits three active nodes. Then, a_h observes its own virtual ID, the virtual ID of a_h's forward active agent a_i, and the virtual ID of a_i's forward active agent a_j. Thus, a_h can obtain three virtual IDs id_1, id_2, id_3 without using whiteboards. Therefore, agents can use the above approach for a unidirectional ring, that is, a_h behaves as if it would be an active agent with ID id_2 in a bidirectional ring. In the rest of this paragraph, we explain how agents detect active nodes. In the beginning of the algorithm, each agent starts the algorithm at a token node and all token nodes are active nodes. After each agent a_h visits three active nodes, a_h decides whether a_h remains active or drops out from the set of leader candidates at the active (token) node. If a_h remains active, then a_h starts the next phase and leaves the active node. Thus, in some phase, when some active agent a_h visits a token node v_j where no agents exist, a_h knows that a_h visits an active node and the other nodes are not active in the phase.

After observing three virtual IDs id_1, id_2, id_3, each active agent a_h compares virtual IDs by the lexicographical order and decides whether a_h remains active (as a candidate for leaders) in the next phase or not. Different from [11], multiple agents may have the same IDs. To treat this case, if $id_2 < \min(id_1, id_3)$ or $id_2 = id_3 < id_1$ holds, then a_h remains active as a candidate for leaders. Otherwise, a_h becomes inactive and drops out from the set of leader candidates. For example, let us consider the initial configuration of Fig. 8 (a). In the figure, black nodes are token nodes and the numbers near communication links are port numbers. The virtual ring of Fig. 8 (a) is shown in Fig. 8 (b). For simplicity, we omit non-token nodes in Fig. 8 (b). The numbers in Fig. 8 (b) are virtual IDs. Each agent a_h continues to move until a_h visits three active nodes. By the movement, a_1 observes three virtual IDs $(01, 01, 01)$, a_2 observes three virtual IDs $(01, 01, 1000101010)$, a_3 observes three virtual IDs $(01, 1000101010, 01)$, and a_4 observes three virtual IDs $(1000101010, 01, 01)$, respectively. Thus, a_4 remains as a candidate for leaders, and a_1, a_2, and a_3 drop out from the set of leader candidates. Note that, like Fig. 8, if an agent observes the same virtual IDs three times, it drops out from the set of leader candidates. This implies, if all active agents have the same virtual IDs, all agents become inactive. However, we can show that, when there exist at least three active agents, it does not happen that all active agents observe the same virtual IDs. Thus in each phase, at least the half of active agents become inactive, and we show this later (Lemma 5). Moreover, if there are only one or two active agents in some phase, then the agents notice the fact during the phase. In this case, the agents immediately become leaders. By executing $\lceil \log g \rceil$ phases, agents complete the leader agent election.

Pseudocode. The pseudocode to elect leaders is given in Algorithm 2. All agents start the algorithm with active states. The pseudocode describes the behavior of active agent a_h, and v_j represents the node where agent a_h currently stays. If agent a_h becomes inactive or a leader, a_h immediately moves to the next part and executes the algorithm for an inactive state or a leader state in Section 6.2. In Algorithm 2, a_h uses the following variables:

- id_1, id_2, and id_3 are variables for storing three virtual IDs.
- *phase* is a variable for storing its own phase number.

In Algorithm 2, each active agent a_h moves until a_h observes three virtual IDs and decides whether a_h remains active as a candidate for leaders or not on the basis of the virtual IDs. Note that, since each agent moves in a FIFO manner, it does not happen that some active agent passes another active agent in the virtual ring, and each active agent correctly observes three neighboring virtual IDs in the phase. In Algorithm 2, a_h uses procedure *NextActive*(), by which a_h moves to the next active node and returns the port sequence as a virtual ID. The pseudocode of *NextActive*() is described in Procedure 1. In *NextActive*, a_h uses the following variables:

Algorithm 2 The behavior of active agent a_h (v_j is the current node of a_h).

Variables for Agent a_h
int $phase = 0$;
int id_1, id_2, id_3;
Main Routine of Agent a_h
1: $phase = phase + 1$
2: $id_1 = NextActive()$
3: $id_2 = NextActive()$
4: $id_3 = NextActive()$
5: **if** the number of active agents in the tree is two or less **then**
6: change its state to a leader state
7: break Algorithm 2
8: **end if**
9: **if** $(id_2 < min(id_1, id_3)) \vee (id_2 = id_3 < id_1)$ **then**
10: **if** $(phase = \lceil \log g \rceil)$ **then**
11: change its state to a leader state
12: break Algorithm 2
13: **else**
14: go to line 1
15: **end if**
16: **else**
17: change its state to an inactive state
18: **end if**

Procedure 1 int *NextActive*() (v_j is the current node of a_h).

Variables for Agent a_h
array $port[\ ]$;
int $move$;
Behavior of Agent a_h
1: $move = 0$
2: leave v_j through the port 0
// arrive at the forward node and v_j is updated
3: let p be the port number through which a_h visits v_j
4: $port[move] = p$
5: $move = move + 1$
6: **while** (there does not exist a token) $\vee$
$(p \neq d_{v_j} - 1) \vee$ (there exists another agent) **do**
7: leave v_j through the port $(p + 1) \mod d_{v_j}$
// arrive at the forward node and v_j is updated
8: let p be the port number through which a_h visits v_j
9: $port[move] = p$
10: $move = move + 1$
11: **end while**
12: return $port[\ ]$

- *port* is an array for storing a virtual ID.
- *move* is a variable for storing the number of nodes it visits.

During the basic walk, each active agent visits active node v_j through the port $(d_{v_j} - 1)$. Thus, when agent a_h leaves active node v_j, it always uses the port 0 (line 2 in Procedure 1).

Note that, if there exist only one or two active agents in some phase, then the agent travels once around the virtual ring before getting three virtual IDs. In this case, the active agent knows that there exist at most two active agents in the phase and they become leaders (lines 5 to 8 in Algorithm 2). To do this, agents record the topology every time they visit nodes, but we omit the description of this behavior in Algorithm 2 and Procedure 1.

First, we show the following lemma to show that at least one agent remains active or becomes a leader in each phase.

Lemma 4. *When there exist three or more active agents, there exist two active agents having different virtual IDs.*

Proof. To show the lemma, we use the theorem from [5]. Let $t[1..q]$ be a port sequence that an agent observes in visiting q nodes by performing the basic walk. In our algorithm, $t[1..q]$ corresponds to $port[\ ]$ (described in Procedure 1) and represents a virtual ID that the agent gets in traverse from an active node to the next active node. Moreover, $(t[1..q])^k$ denotes the concatenation of k copies of $t[1..q]$. If $t[1..q] = (t[1..q'])^k$ holds some positive integers q' and k $(q = q'k)$, we call $t[1..q]$ is periodic. Otherwise, we call $t[1..q]$ is aperiodic. In addition, the *length* of an n-node tree T is the length of its Euler tour, that is, $2(n-1)$. Then, we use the following theorem.

Theorem 7. [5] *Let T be a tree of length at least $q \geq 1$. Assume that $t[1..q]$ is aperiodic and $t[1..kq] = (t[1..q])^k$ for some $k \geq 3$. Then one of the following three cases must hold.*

1. *The length of T is q.*
2. *The length of T is $2q$.*
3. *The length of T is greater than kq.* □

We show the lemma by contradiction, that is, assume that there exist $k' \geq 3$ active agents in some phase and all the k' active agents have the same virtual IDs. Let x be the virtual ID. Then, $t[1..|x|] = x$ holds. In addition, when each active agent moves in the tree and observes one virtual ID x, each link in the virtual link is passed by exactly once. Hence, $t[(\ell|x|+1)..(\ell+1)|x|] = x$ holds $(0 \leq \ell \leq k'-1)$ and $t[1..k'|x|] = (t[1..|x|])^{k'}$ holds. Moreover, in this case the total number of their moves (i.e., $k'|x|$) is equal to the length of the tree. If x is aperiodic, the length of the tree is $k'|x|$. However from Theorem 7, the length of the tree is never $k'|x|$, which is a contradiction. If x is periodic, $t[1..|x|] = (t[1..|x'|])^s$ holds for some x' and s (x' is aperiodic). Then, $t[1..k'|x|] = t([1..|x'|])^{k's}$ holds and the length of the tree is $k's|x'|(= k'|x|)$. However, from Theorem 7, the length of the tree is never $k's|x'|$, which is also a contradiction. □

Next, we have the following lemmas about Algorithm 2.

Lemma 5. *Algorithm 2 eventually terminates, and satisfies the following two properties.*

- *There exists at least one leader agent.*
- *In the virtual ring, there exist at least $g-1$ inactive agents between two leader agents.*

Proof. We show the lemma in the virtual ring. Obviously, Algorithm 2 eventually terminates. In the following, we show the above two properties.

At first, we show that there exists at least one leader agent. From lines 5 to 7 of Algorithm 2, when there exist only one or two active agents in some phase, the agents become leaders. We assume that in some phase, active agent a_h observes three IDs $a_h.id_1, a_h.id_2$, and $a_h.id_3$ in this order. When there are three or more active agents in some phase, if $a_h.id_2 < \min(a_h.id_1, a_h.id_3)$ or $a_h.id_2 = a_h.id_3 < a_h.id_1$ holds, agent a_h remains as a candidate for leaders, and otherwise a_h drops out from the set of leader candidates. Thus, unless all agents observe the same virtual IDs, at least one agent remains active as a candidate for leaders. From Lemma 4, it does not happen that all agents observe the same virtual IDs. Therefore, there exists at least one leader agent.

Next, we show that there exist at least $g-1$ inactive agents between two leader agents in the virtual ring. At first, we show that in each phase, at least half of active agents become inactive. In each phase, if $a_h.id_2 < \min(a_h.id_1, a_h.id_3)$ or $a_h.id_2 = a_h.id_3 < a_h.id_1$ holds, a_h remains as a candidate for leaders. If the agent a_h satisfies $a_h.id_2 < \min(a_h.id_1, a_h.id_3)$, then the a_h's backward and forward active agents drop out from the set of leader candidates. In the following, we consider the case that agent a_h satisfies $a_h.id_2 = a_h.id_3 < a_h.id_1$. Let $a_{h'}$ be a a_h's backward active agent and $a_{h''}$ be a a_h's forward active agent. Agent $a_{h'}$ observes three virtual IDs $a_{h'}.id_1, a_{h'}.id_2, a_{h'}.id_3$, and both $a_{h'}.id_2 = a_h.id_1$ and $a_{h'}.id_3 = a_h.id_2$ hold. Hence, $a_{h'}.id_2 > a_{h'}.id_3$ holds, and $a_{h'}$ drops out from the set of leader candidates. Next, $a_{h''}$ observes three virtual IDs $a_{h''}.id_1, a_{h''}.id_2, a_{h''}.id_3$, and both $a_{h''}.id_1 = a_h.id_2$ and $a_{h''}.id_2 = a_h.id_3$ hold. Since $a_{h''}.id_1 = a_{h''}.id_2$ holds, $a_{h''}$ does not satisfy the condition to remain as a candidate for leaders and drops out from the candidate. Thus in each phase, at least half of active agents drop out from the set of leader candidates and become inactive. Now, we show that there exist at least $g-1$ inactive agents between two leader agents. We firstly show that after executing j phases, there exist at least $2^j - 1$ inactive agents between two active agents. We show this by induction. For the case of $j = 1$, there exists at least $2^1 - 1 = 1$ inactive agent between two active agents as mentioned above. For the case of $j = k$, we assume that there exist at least $2^k - 1$ inactive agents between two active agents. After executing $k+1$ phases, since at least one of neighboring active agents becomes inactive, the number of inactive agents between two active agents is at least $(2^k - 1) + 1 + (2^k - 1) = 2^{k+1} - 1$. Hence, after executing j phases, there exist at least $2^j - 1$ inactive agents between two active agents. Therefore, after executing $\lceil \log g \rceil$ phases, there exist at least $g-1$ inactive agents between two leader agents in the virtual ring. □

Lemma 6. *Algorithm 2 requires $O(n \log g)$ total moves.*

Proof. In the virtual ring, each active agent moves until it observes three virtual IDs in each phase. This requires at most $O(n)$ total moves because each communication link of the virtual ring is passed by at most three agents (including the same agent if only one or two active agents exist) and the length of the ring is $2(n-1)$. Since agents execute $\lceil \log g \rceil$ phases, we have the lemma. □

6.2. The second part: leaders' instruction and agents' movement

In this section, we explain the second part, i.e., an algorithm to achieve the g-partial gathering by using the elected agents. Let leader nodes (resp., inactive nodes) be the nodes where agents become leaders (resp., inactive agents). Note that all leader nodes and inactive nodes are token nodes. In this part, each agent takes one of the following three states:

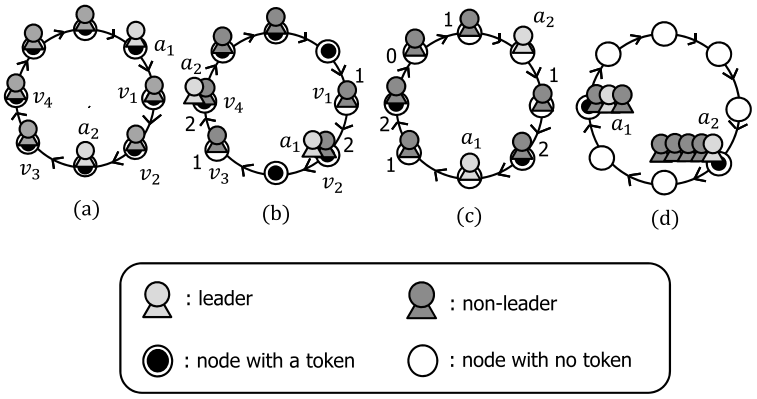


Fig. 9. Partial gathering in the removable-token model for the case of $g = 3$ (a_1 and a_2 are leaders, and black nodes are token nodes).

- *leader*: The agent instructs inactive agents where they should move.
- *inactive*: The agent waits for the leader's instruction.
- *moving*: The agent moves to its gathering node.

We explain the idea of the algorithm in the virtual ring. The basic movement is also similar to [11], that is, to divide agents into groups each of which consists of at least g agents. While in [11], each node has a whiteboard, in this section each node is allowed to only have a removable token. Each leader agent a_h moves to the next leader node, and during the movement a_h repeats the following behavior: a_h removes tokens of inactive nodes $g-1$ times consecutively and then a_h does not remove a token of the next inactive node. The behavior guarantees that at least $g-1$ agents exist between any two token nodes when all the leaders complete the behavior. After that, agents move to the nearest token nodes, which guarantees that at least g agents meet at each token node.

First, we explain the behavior of leader agents. Whenever leader agent a_h visits an inactive node v_j, it counts the number of inactive nodes (including the current node) that a_h has visited. If the number plus one is not a multiple of g, a_h removes a token at v_j. Otherwise, a_h does not remove the token and continues to move. Agent a_h continues this behavior until a_h visits the next leader node $v_{j'}$ (Later, explain how a_h detects whether it visits the next leader node $v_{j'}$ or not). After that, a_h removes a token at $v_{j'}$. When all the leaders complete this behavior, there exist at least $g-1$ inactive agents between two token nodes. Hence, agents solve the g-partial gathering problem by moving to the nearest token node (This is done by changing their states to moving states). For example, let us consider the configuration of Fig. 9 (a) ($g = 3$). We assume that a_1 and a_2 are leader agents and the other agents are inactive agents. In Fig. 9 (b), a_1 visits node v_2 and a_2 visits node v_4, respectively. The number near each node represents the number (modulo g) of inactive nodes that a_1 or a_2 has ever visited. Then, agents a_1 and a_2 remove tokens at v_1 and v_3, and do not remove tokens at v_2 and v_4, respectively. After that, a_1 and a_2 continue this behavior until they visit the next leader nodes. At the leader nodes, they remove the tokens (Fig. 9 (c)).

When a token at v_j is removed, an inactive agent at v_j changes its state to a moving state and starts to move. Concretely, each moving agent moves to the nearest token node v_j. Note that, since each agent moves in a FIFO manner, it does not happen that a moving agent passes a leader agent and terminates at some token node before the leader agent removes the token. After all agents complete their own movements, the configuration changes from Fig. 9 (c) to Fig. 9 (d) and agents can solve the g-partial gathering problem. Note that, since each agent moves in the same virtual ring in a FIFO manner, it does not happen that an active agent executing the leader agent election passes a leader agent and that a leader agent passes an active agent.

Pseudocode. In the following, we show the pseudocode of the algorithm. The pseudocode of leader agents is described in Algorithm 3. Variable $tCount$ is used to count the number of inactive nodes a_h has ever visited. When a_h visits a token node v_j where another agent exists, v_j is an inactive node because an inactive agent becomes inactive at a token node and agents move in a FIFO manner. Whenever each leader agent a_h visits an inactive node, a_h increments the value of $tCount$. At inactive node v_j, a_h removes a token at v_j if $tCount \neq g-1$ (does not remove a token otherwise) and continues to move (lines 5 to 9). This guarantees that, if a token at inactive node v_j is not removed, at least g agents meet at v_j. When a_h removes a token at v_j, an inactive agent at v_j changes its state to a moving state (line 7). When a_h visits a token node $v_{j'}$ where no agents exist, $v_{j'}$ is the next leader node. This is because token nodes are leader nodes or inactive nodes, and from an atomicity of the execution there exist no agents at each leader node. Note that also from an atomicity of the execution, it does not happen that some leader agent visits a leader node v such that another agent becomes a leader at v but still stays at v. When leader agent a_h moves to the next leader node $v_{j'}$, a_h removes a token at $v_{j'}$ and changes its state to a moving state. In Algorithm 3, a_h uses the procedure *NextToken*() to move to the next token node. The pseudocode of *NextToken*() is described in Procedure 2. In Procedure 2, a_h performs the basic walk until a_h visits a token node v_j through the port $(d_{v_j} - 1)$.

The pseudocode of inactive agents is described in Algorithm 4. Inactive agent a_h waits at v_j until either a token at v_j is removed or a_h observes another agent. If the token is removed, a_h changes its state to a moving state (lines 4 to 6). If a_h observes another agent, the agent is a moving agent and terminates the algorithm at v_j (lines 7 to 9). This means v_j is selected as a token node where at least g agents meet at the end of the algorithm. Hence, a_h terminates the algorithm at v_j.

The pseudocode of moving agents is described in Algorithm 5. In the virtual ring, each moving agent a_h moves to the nearest token node by using *NextToken*().

Algorithm 3 The behavior of leader agent a_h (v_j is the current node of a_h).

```
Variable in Agent a_h
int tCount = 0;
Main Routine of Agent a_h
 1: NextToken()
 2: while there exists another agent at v_j do
 3:    //this is an inactive node
 4:    tCount = (tCount + 1) mod g
 5:    if tCount ≠ g − 1 then
 6:       remove a token at v_j
 7:       //an inactive agent at v_j changes its state to a moving state
 8:    end if
 9:    NextToken()
10: end while
11: remove a token at v_j // this is a leader node
12: change its state to a moving state
```

Procedure 2 void *NextToken*() (v_j is the current node of a_h).

```
1: leave v_j through the port 0
   // arrive at the forward node and v_j is updated
2: let p be the port number through which a_h visits v_j
3: while (there does not exist a token) ∨ (p ≠ d_{v_j} − 1) do
4:    leave v_j through the port (p + 1) mod d_{v_j}
      // arrive at the forward node and v_j is updated
5:    let p be the port number through which a_h visits v_j
6: end while
```

Algorithm 4 The behavior of inactive agent a_h (v_j is the current node of a_h).

```
Main Routine of Agent a_h
1: while (there does not exist another agent at v_j)∨(there exists a token at v_j) do
2:    wait at v_j
3: end while
4: if there exists another agent at v_j then
5:    terminate the algorithm
6: end if
7: if there does not exist a token then
8:    change its state to a moving state
9: end if
```

Algorithm 5 The behavior of moving agent a_h (v_j is the current node of a_h).

```
Main Routine of Agent a_h
1: NextToken()
2: terminate the algorithm
```

We have the following lemma about the algorithms.

Lemma 7. *After the leader agent election, agents solve the g-partial gathering problem in $O(gn)$ total moves.*

Proof. We show the lemma in the virtual ring. At first, we show the correctness of the proposed algorithms. Let $v_0^g, v_1^g, \ldots, v_l^g$ be inactive nodes that still have tokens after all leader agents complete their behaviors, and we call these nodes *gathering nodes*. From Algorithm 3, each leader agent a_h removes the tokens at the consecutive $g-1$ inactive nodes and does not remove the token at the next inactive node. By this behavior and Lemma 5, there exist at least $g-1$ moving

agents between v_i^g and v_{i+1}^g. Moreover, these moving agents move to the nearest gathering node v_{i+1}^g. Therefore, agents solve the g-partial gathering problem.

In the following, we evaluate the total number of moves required for the algorithms. At first, let us consider the total number of moves required for leader agents to move to the next leader nodes. This requires $2(n-1)$ total moves since all leader agents travel once around the virtual ring. Next, let us consider the total number of moves required for moving (inactive) agents to move to the nearest token nodes (For example, the total number of moves form Fig. 9 (c) to Fig. 9 (d)). From Algorithm 5, each moving agent moves to the nearest gathering node. In the following, we show that the number of moving agents between some gathering node v_i^g and its forward gathering node v_{i+1}^g is $O(g)$. From Algorithm 3, the moving agents between v_i^g and v_{i+1}^g consist of inactive agents and leader agents between v_i^g and v_{i+1}^g. Since there exists at least one gathering node between two leader nodes, there exists at most one leader node between v_i^g and v_{i+1}^g. If there exist no leader node between v_i^g and v_{i+1}^g, then clearly there exist $g-1$ inactive nodes between v_i^g and v_{i+1}^g. If there exists one leader node v_l between v_i^g and v_{i+1}^g, there exist at most $g-1$ inactive nodes between v_i^g and v_l, and at most $g-1$ inactive nodes between v_l and v_{i+1}^g, respectively. Thus, there exist at most $O(g)$ moving agents between gathering nodes v_i^g and v_{i+1}^g, and the total number of moves required for moving (inactive) agents to move to the nearest gathering nodes is at most $O(gn)$ since each communication link is passed by at most $O(g)$ times.

Therefore, we have the lemma. □

From Lemma 6 and Lemma 7, we have the following theorem.

Theorem 8. *In the weak multiplicity detection and the removable-token model, our algorithm solves the g-partial gathering problem in $O(gn)$ total moves.* □

7. Conclusion

In this paper, we considered the g-partial gathering problem in asynchronous tree networks. At first, in the non-token model we showed that agents require $\Omega(kn)$ total moves to solve the g-partial gathering problem. After this, we considered three model variants. First, in the weak multiplicity detection and non-token model, for asymmetric trees agents can solve the g-partial gathering problem in $O(kn)$ total moves by the previous result in [10], and we showed that there exist no algorithms to solve the g-partial gathering problem for symmetric trees. Second, in the strong multiplicity detection and non-token model, we proposed a deterministic algorithm to solve the g-partial gathering problem in $O(kn)$ total moves. Finally, in the weak multiplicity detection and removable-token model, we proposed a deterministic algorithm to solve the g-partial gathering problem in $O(gn)$ total moves.

Open problems are as follows. The first is to consider the weak multiplicity detection and non-token model for symmetric trees (the same model as in Section 4) when the locations of agents are asymmetric or $2 \leq g \leq 4$ holds. We conjecture that, even in this case, the g-partial gathering problem is not solvable by the similar discussion in Section 4. The second is to consider the g-partial gathering problem in general networks. We conjecture that in this model, the g-partial gathering problem can be solved efficiently in terms of total moves by the similar way to Section 6, that is, agents execute the leader agent election partially and elected leaders instruct non-leaders gathering nodes.

References

[1] E. Kranakis, D. Krozanc, E. Markou, The Mobile Agent Rendezvous Problem in the Ring, Synthesis Lectures on Distributed Computing Theory, vol. 1, 2010, pp. 1–122.
[2] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile search for a black hole in an anonymous ring, Algorithmica 48 (1) (2007) 67–90.
[3] T. Suzuki, T. Izumi, F. Ooshita, H. Kakugawa, T. Masuzawa, Move-optimal gossiping among mobile agents, Theoret. Comput. Sci. 393 (1) (2008) 90–101.
[4] J. Chalopin, S. Das, A. Kosowski, Constructing a map of an anonymous graph: applications of universal sequences, in: Proc. of the 12th International Conference on Principles of Distributed Systems, in: LNCS, vol. 6490, 2010, pp. 119–134.
[5] L. Gasieniec, A. Pelc, T. Radzik, X. Zhang, Tree exploration with logarithmic memory, in: Proc. of the 18th Annual ACM–SIAM Symposium on Discrete Algorithms, 2007, pp. 585–594.
[6] P. Fraigniaud, A. Pelc, Deterministic rendezvous in trees with little memory, in: Proc. of the 22nd International Symposium on Distributed Computing, in: LNCS, vol. 6950, 2008, pp. 242–256.
[7] P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, ACM Trans. Algorithms 9 (2) (2013) 17.
[8] J. Czyzowicz, A. Kosowski, A. Pelc, Time vs. space trade-offs for rendezvous in trees, in: Proc. of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, 2012, pp. 1–10.
[9] S. Elouasbi, A. Pelc, Time of anonymous rendezvous in trees: determinism vs. randomization, in: Proc. of the 19th International Colloquium on Structural Information and Communication Complexity, in: LNCS, vol. 7355, 2012, pp. 291–302.
[10] D. Baba, T. Izumi, F. Ooshita, H. Kakugawa, T. Masuzawa, Linear time and space gathering of anonymous mobile agents in asynchronous trees, Theoret. Comput. Sci. 478 (2013) 118–126.
[11] M. Shibata, S. Kawai, F. Ooshita, H. Kakugawa, T. Masuzawa, Algorithms for partial gathering of mobile agents in asynchronous rings, in: Proc. of the 16th International Conference on Principles of Distributed Systems, in: LNCS, vol. 7702, 2012, pp. 254–268.
[12] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Multiple agents rendezvous in a ring in spite of a black hole, in: Proc. of the 8th International Conference on Principles of Distributed Systems, in: LNCS, vol. 3144, 2004, pp. 34–46.
[13] L. Barriere, P. Flocchini, P. Fraigniaud, N. Santoro, Rendezvous and election of mobile agents: impact of sense of direction, Theory Comput. Syst. 40 (2) (2007) 143–162.

[14] S. Kawai, F. Ooshita, H. Kakugawa, T. Masuzawa, Randomized rendezvous of mobile agents in anonymous unidirectional ring networks, in: Proc. of the 19th International Colloquium on Structural Information and Communication Complexity, in: LNCS, vol. 7355, 2012, pp. 303–314.
[15] G.D. Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vaccaro, Asynchronous deterministic rendezvous in graphs, Theoret. Comput. Sci. 355 (3) (2005) 315–326.
[16] S. Guilbault, A. Pelc, Gathering asynchronous oblivious agents with restricted vision in an infinite line, in: Proc. of the 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems, in: LNCS, vol. 8255, 2013, pp. 296–310.
[17] A. Collins, J. Czyzowicz, L. Gasieniec, A. Kosowski, R. Martin, Synchronous rendezvous for location-aware agents, in: Proc. of the 25th International Symposium on Distributed Computing, in: LNCS, vol. 6950, 2011, pp. 447–459.
[18] E. Kranakis, D. Krizanc, E. Markou, Mobile agent rendezvous in a synchronous torus, in: Proc. of the 8th Latin American Theoretical Informatics, in: LNCS, vol. 3887, 2006, pp. 653–664.
[19] S. Guilbault, A. Pelc, Asynchronous rendezvous of anonymous agents in arbitrary graphs, in: Proc. of the 32nd International Symposium on Distributed Computing, in: LNCS, vol. 7109, 2011, pp. 421–434.
[20] Y. Dieudonne, A. Pelc, D. Peleg, Gathering despite mischief, in: Proc. of the 23nd Annual ACM–SIAM Symposium on Discrete Algorithms, 2012, pp. 527–540.
[21] P. Flocchini, E. Kranakis, D. Krizanc, F.L. Luccio, N. Santoro, C. Sawchuk, Mobile agents rendezvous when tokens fail, in: Proc. of the 11th International Colloquium on Structural Information and Communication Complexity, in: LNCS, vol. 3104, 2004, pp. 161–172.
[22] N. Santoro, Determining topology information in distributed networks, in: Proc. of the 11th Southeaster Conference on Combinatorics, Graph Theory and Computing, 1980, pp. 869–878.
[23] G.L. Peterson, An $O(n \log n)$ unidirectional algorithm for the circular extrema problem, ACM Trans. Program. Lang. Syst. 4 (4) (1982) 758–762.