Move-optimal partial gathering of mobile agents without identifiers or global knowledge in asynchronous unidirectional rings [☆,☆☆]

Masahiro Shibata[a,*], Norikazu Kawata[b], Yuichi Sudo[b], Fukuhito Ooshita[c], Hirotsugu Kakugawa[d], Toshimitsu Masuzawa[b]

[a]*Graduate school of Computer Science and Systems Engineering, Kyushu Institute of Technology*
680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan
[b]*Graduate School of Information Science and Technology, Osaka University*
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
[c]*Graduate School of Science and Technology, NAIST*
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
[d]*Faculty of Science and Technology, Ryukoku University*
1-5 Seta, Otsu, Shiga, 520-2194, Japan

Abstract

In this paper, we consider the partial gathering problem of mobile agents in asynchronous unidirectional ring networks. The partial gathering problem is a generalization of the (well-investigated) total gathering problem, which requires that all the k agents distributed in the network terminate at a single node. The partial gathering problem requires, for a given positive integer $g\,(< k)$, that all the agents terminate in a configuration such that either

[☆]The conference version of this paper is published in the proceedings of 26th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2019).

[☆☆]This work was partially supported by JSPS KAKENHI Grant Number 17K19977, 18K18000, 18K18031, 18K11167, 19H04085, and 19K11826, and Japan Science and Technology Agency (JST) SICORP.

[*]Corresponding author. Tel.:+81-9-4829-7656.

Email addresses: `shibata@cse.kyutech.ac.jp` (Masahiro Shibata), `n-kawata@ist.osaka-u.ac.jp` (Norikazu Kawata), `y-sudou@ist.osaka-u.ac.jp` (Yuichi Sudo), `f-oosita@is.naist.jp` (Fukuhito Ooshita), `kakugawa@rins.ryukoku.ac.jp` (Hirotsugu Kakugawa), `masuzawa@ist.osaka-u.ac.jp` (Toshimitsu Masuzawa)

at least g agents or no agent exists at each node. The requirement for the partial gathering problem is strictly weaker than that for the total gathering problem, and thus it is interesting to clarify the difference on the move complexity between them. In this paper, we aim to solve the partial gathering problem for agents without identifiers or any global knowledge such as the number k of agents or the number n of nodes. We consider deterministic and randomized cases. First, in the deterministic case, we show that the set of unsolvable initial configurations is the same as that for the case of agents with knowledge of k. In addition, we propose an algorithm that solves the problem from any solvable initial configuration in a total number of $O(gn)$ moves. Next, in the randomized case, we propose an algorithm that solves the problem in a total number of $O(gn)$ moves in expectation from any initial configuration. Note that $g < k$ holds and agents require a total number of $\Omega(gn)$ (resp., $\Omega(kn)$) moves to solve the partial (resp., total) gathering problem. Thus, our algorithms can solve the partial gathering problem in asymptotically optimal total number of moves without identifiers or global knowledge, and the total number of $O(gn)$ moves is strictly smaller than that for the total gathering problem.

keywords: distributed system, mobile agent, gathering problem, partial gathering problem

---

1. Introduction

1.1. Background and related works

A *distributed system* consists of a set of computing entities (*nodes*) connected by communication links. As a promising design paradigm of distributed systems, (mobile) agents have attracted much attention [9, 2]. The agents can traverse the system, carrying information collected at visited nodes and execute tasks at each node using the information. In other words, agents can encapsulate the process code and data, which simplifies design of distributed systems [13, 3].

The *total gathering problem* (or rendezvous problem) is a fundamental problem for agents' coordination. When a set of k agents are arbitrarily placed at nodes, this problem requires that all the k agents in the system terminate at a single node in finite time. By meeting at a single node, all agents can share information or synchronize behaviors among them. The total gathering problem has been considered in various kinds of networks such as rings [12, 6, 8, 11, 14], trees [7, 1], tori networks [10], and arbitrary networks [4, 5]. Kranakis et al. [12] introduced the total gathering problem in ring networks for the first time, and they considered the problem for two agents. The number of agents is extended to any number in [6, 8]. Flocchini et al. [6] showed that the lower bound on memory space per agent to solve the total gathering problem is $\Omega(\log k + \log\log n)$, where n is the number of nodes. Later, Gasieniec et al. [8] give a space-optimal (i.e., $O(\log k+\log\log n)$ memory space per agent) algorithm that solves the problem. These results [12, 6, 8] are summarized by Kranakis et al. [11]. While the above results [12, 6, 8] considered deterministic algorithms, Ooshita et al. [14] considered a randomized algorithm for anonymous agents without any global knowledge and clarified solvability.

Recently, a variant of the total gathering problem, called the *g-partial gathering problem* [16], has been considered. This problem does not require all the k agents to meet at a single node, but allows the agents to meet partially at several nodes. Concretely, the problem requires, for a given positive integer $g\,(< k)$, that all the agents terminate in a configuration such that either at least g agents or no agent exists at each node (e.g., Fig. 1). From a practical point of view, the g-partial gathering problem is still useful especially for monitoring large-scale networks. For example, we assume that several areas in the network have some problems and they are distant from each other. When agents achieve total gathering and they meet at a single node near some area, they take a lot of time to treat the other areas. On the other hand, when agents achieve g-partial gathering so that k agents are partitioned into groups each of which has at least g agents and the groups are distributed uniformly in the network, they can treat each area quickly. Thus, meeting partially enables agents to monitor the network collaboratively and efficiently. The g-partial gathering problem is interesting also from a theoretical point of view. Clearly, if $k/2 < g < k$ holds, the g-partial gathering problem is equivalent to the total gathering problem. On the other hand, if $2 \leq g \leq k/2$ holds, the requirement for the g-partial gathering problem is strictly weaker than that for the total gathering problem. Thus, it is possible that the g-partial gathering problem can be solved with a strictly smaller total number of moves (i.e., lower costs) compared to the total gathering problem.

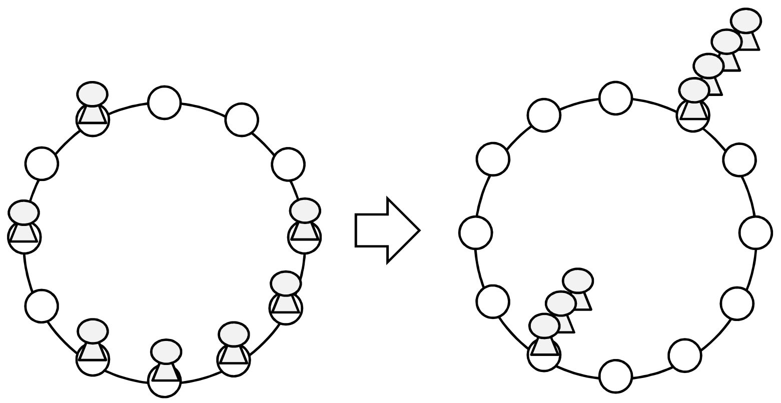


Figure 1: An example of the g-partial gathering problem ($g = 3$).

1.2. Previous works on partial gathering

So far, the g-partial gathering problem has been considered in rings [16], trees [18], and arbitrary networks [17]. In [16], Shibata et al. considered the g-partial gathering problem in unidirectional ring networks with whiteboards (or memory spaces that agents can read and write) at nodes. They considered two problem settings about agents: distinct agents (i.e., agents with distinct IDs) and anonymous agents (i.e., agents without IDs) with knowledge of k. For distinct agents, they give a deterministic algorithm that solves the g-partial gathering problem in a total number of $O(gn)$ moves. For anonymous agents with knowledge of k, they considered deterministic and randomized cases. In the deterministic case, they showed that there exist unsolvable initial configurations. In addition, they give an algorithm that solves the g-partial gathering problem from any solvable initial configuration in a total number of $O(kn)$ moves. In the randomized case, they give an algorithm that solves the g-partial gathering problem in a total number of $O(gn)$ moves in expectation. Note that in the above three results agents do not have any other global knowledge such as n. In addition, the g-partial (resp., the total) gathering problem in ring networks requires a total number of $\Omega(gn)$ (resp., $\Omega(kn)$) moves. Thus, the first and third results are asymptotically optimal in terms of a total number of moves, and the total number of $O(gn)$ moves is strictly smaller than that for the total gathering problem when $g = o(k)$.

In [18], Shibata et al. considered the g-partial gathering problem in tree networks. Since trees have lower symmetry than rings and some problems considered in trees are easily solved compared to rings, they considered the problem in weaker models than those for rings and clarified what condition is needed to achieve g-partial gathering with the same performance as that for rings. To do this, they considered agents that are anonymous and have no knowledge of k or n, and they considered three model variants. In the first two models, they consider nodes with no memory space (or whiteboards), but different multiplicity detection capability of agents are considered in the two models. Here, multiplicity detection capability means the capability of agents to detect the existence of agents staying at the same node, and whether the exact number of agents can be counted or not depends on the strength of the capability. In the first model, they consider agents with the weak multiplicity detection where each agent can detect whether another agent exists at the current node or not but cannot count the exact number of the agents. Then, they showed that, for asymmetric trees agents can solve the g-partial gathering problem in a total number of $O(kn)$ moves, and for symmetric trees agents cannot solve the g-partial gathering problem for the case of $g \geq 5$. In the second model, they consider agents with the strong multiplicity detection where each agent can count the exact number of agents staying at the current node. In this case, they showed that agents require a total number of $\Omega(kn)$ moves to solve the g-partial gathering problem from some initial configurations and they give a deterministic algorithm that solves the problem in a total number of $O(kn)$ (i.e., optimal) moves regardless of the initial configuration. In the third model, they consider agents with the weak multiplicity detection, and with removable identical tokens. In this token model, each agent represents its existence in the initial configuration using 1-bit memory on the whiteboard (this memory is called a token), and agents can remove any owner's token during the execution of the algorithm. In this case, they give a deterministic algorithm that solves the g-partial gathering problem in a total number of $O(gn)$ moves. This result shows that it is sufficient to use weak multiplicity detection and removable tokens to achieve g-partial gathering in a total number of $O(gn)$ moves, which is the strictly weaker assumption than the whiteboard model for rings.

In [17], Shibata et al. considered the g-partial gathering problem in arbitrary networks under the assumption that agents have distinct IDs and each node has a whiteboard. Then, they showed that agents require a total number of $\Omega(gn + m)$ moves, where m is the number of communication links, and give a deterministic algorithm that solves the problem in a total number of $O(gn + m)$ moves. Thus, they showed that agents can achieve move-optimal g-partial gathering also in arbitrary networks.

Table 1: Results of g-partial gathering for anonymous agents in unidirectional ring networks (n: #nodes, k: #agents)

	Previous results [16]		Results of this paper	
	Result 1	Result 2	Result 1 (Sec. 3)	Result 2 (Sec. 4)
Deter./Rand.	Deter.	Rand.	Deter.	Rand.
Knowledge of k	Available	Available	No	No
Unsolvable configurations	Exist	None	Exist	None
Agents can detect unsolvability	Yes	-	No	-
Total number of moves	$O(kn)$	$\Theta(gn)$	$\Theta(gn)$	$\Theta(gn)$

1.3. Our contribution

In this paper, for the case of $2 \leq g \leq k/2$, we consider the g-partial gathering problem in asynchronous unidirectional ring networks with whiteboards at nodes as in [16]. In this paper, we aim to solve the problem in weaker models than those of [16] while maintaining the optimality of move complexity. That is, while the previous work [16] achieved move-optimal g-partial gathering by the deterministic algorithm for agents with distinct IDs or the randomized algorithm for anonymous agents with knowledge of k, in this paper we aim to achieve this only using knowledge of g (i.e., they do not require distinct IDs or knowledge of k).

In Table 1, we compare our contributions with the results for anonymous agents in unidirectional ring networks [16]. We consider deterministic and randomized cases. First, in the deterministic case, we show that, unlike the case of [16], agents cannot detect whether the given initial configuration is one of unsolvable configurations shown in [16] or not. In addition, we propose an algorithm that solves the g-partial gathering problem from any solvable initial configuration in a total number of $O(gn)$ moves. Thus, we can show that the set of unsolvable configurations is the same as that of [16] (i.e., for the case of agents with knowledge of k). Next, in the randomized case, we propose an algorithm that solves the problem with probability 1 in a total number of $O(gn)$ moves in expectation from any initial configuration. Thus, our algorithms can solve the g-partial gathering problem in an asymptotically optimal total number of moves without distinct IDs or global knowledge. These results are improvements of previous results in [16].

2. Preliminaries

2.1. Network and agent models

We use almost the same model as that in [16]. A *unidirectional ring network* R is defined as 2-tuple $R = (V, E)$, where $V = \{v_0, v_1, \dots, v_{n-1}\}$ is a set of nodes and $E = \{e_0, e_1, \dots, e_{n-1}\}$ $(e_i = (v_i, v_{(i+1) \bmod n}))$ is a set of directed links. In the following, we call "unidirectional rings" simply "rings". We denote by $n\,(= |V|)$ the number of nodes. For simplicity, we denote $v_{(i+j) \bmod n}$ by v_{i+j} for any integers i and j. The *distance* from node v_i to v_j is defined to be $(j-i) \bmod n$. We define the direction from v_i to v_{i+1} (resp., v_{i-1}) as the *forward* (resp., *backward*) direction. We call node v_{i+1} the *next node* of v_i.

In this paper, we assume that nodes are anonymous, i.e., they do not have IDs. Every node $v_i \in V$ has a whiteboard that agents at node v_i can read from and write on.

Let $A = \{a_0, a_1, \dots, a_{k-1}\}$ be a set of $k\,(\leq n)$ anonymous agents. Agents can move through directed links, that is, they can move from v_i to v_{i+1} (i.e., move forward) for any i. Agents have knowledge of g but they do not have knowledge of k or n. In addition, agents cannot detect whether other agents exist at the current node or not (or no multiplicity detection capability).

We consider two models. In the first model, agents execute a deterministic algorithm. An agent a_i is a deterministic finite automaton (S, W, δ, $s_{initial}$, s_{final}, $w_{initial}$, $w'_{initial}$). The first element S is the set of all states of an agent, including two special states, initial state $s_{initial}$ and final state s_{final}. The second element W is the set of all states (contents) of a whiteboard, including two special initial states $w_{initial}$ and $w'_{initial}$. We explain the meanings of $w_{initial}$ and $w'_{initial}$ in the next subsection. The third element $\delta : S \times W \mapsto S \times W \times M$ is the state transition function that decides, from the current states of a_i and the current node's whiteboard, the next states of a_i and the whiteboard, and whether a_i moves to the next node or not. The last element $M = \{1, 0\}$ in δ represents whether a_i makes a movement or not. The value 1 represents movement to the next node and 0 represents stay at the current node. We assume that $\delta\,(s_{final}, w_i) = (s_{final}, w_i, 0)$ holds for any state $w_i \in W$, which means that a_i never changes its state, updates the contents of a whiteboard, or leaves the current node once it reaches state s_{final}. We say that an agent *terminates* when its state changes to s_{final}.

In the second model, we consider agents executing a randomized algorithm. An agent a_i in this model is a probabilistic automaton (S, W, R, δ,

$s_{initial}$, s_{final}, $w_{initial}$, $w'_{initial}$). The third element R is a set of random numbers. Since we treat a randomized algorithm, δ is a mapping $S \times W \times R \mapsto S \times W \times M$. If the state of a_i is s_{final}, then $\delta\left(s_{\mathit{final}}, w_i, r\right) = \left(s_{\mathit{final}}, w_i, 0\right)$ holds for any state $w_i \in W$ and any random number $r \in R$. The other elements in the automaton are the same as those in the deterministic model. Note that for both the models all the agents are modeled by the same state machine since they are anonymous.

2.2. System configuration and execution

In an agent system, a (global) *configuration* is defined as the Cartesian product $\mathcal{S} \times \mathcal{W} \times \mathcal{L}$, where $\mathcal{S} \in S^k$ represents the states of all the agents, $\mathcal{W} \in W^n$ represents the states (whiteboard' contents) of all the nodes, and $\mathcal{L} \in \{0, 1, \ldots n-1\}^k$ represents the current locations of agents. The locations of agents $\mathcal{L} = (l_0, l_1, \ldots l_{k-1})$ implies that each agent a_i is located at node v_{l_i}. We define C as a set of all configurations. In an initial configuration $c_0 \in C$, we assume that all the agents are in the same state $s_{initial}$ and deployed arbitrarily at mutually distinct nodes (or no two agents start at the same node), and the states of whiteboards are $w_{initial}$ or $w'_{initial}$ depending on the existence of an agent. That is, when an agent exists at node v, the initial state of v's whiteboard is $w_{initial}$. Otherwise, the state is $w'_{initial}$.

During an execution of the algorithm, we assume that agents move instantaneously, that is, agents always exist at nodes (do not exist on links). Each agent executes the following four operations in an *atomic step*: 1) reads the contents of its current node's whiteboard, 2) executes local computation (or changes its state), 3) updates the contents of the current node's whiteboard, and 4) moves to the next node or stays at the current node. A configuration changes to the next one when a *scheduler* activates some agents and the activated agents perform atomic steps as mentioned before. Concretely, in the deterministic case, letting A_i be a non-empty sequence of the activated agents, configuration c_i changes to c_{i+1} when each agent $a_j \in A_i$ performs an atomic step. If multiple agents at the same node are included in A_i, the agents perform atomic steps one by one following the order of sequence A_i. We denote the transition by $c_i \xrightarrow{A_i} c_{i+1}$. In the randomized case, in addition to A_i, a set R_i of k random numbers is used and the transition from configuration c_i to c_{i+1} is denoted by $c_i \xrightarrow{A_i, R_i} c_{i+1}$. We explain the detail of R_i in the next paragraph.

In the deterministic case, given a sequence of non-empty agent sequences

$\psi = A_0, A_1, \ldots$ such that $c_i \xrightarrow{A_i} c_{i+1}$ holds for any $i \geq 0$, a sequence of configurations $E_{\mathcal{A}}(c_0, \psi) = c_0, c_1, \ldots$ is called an *execution* of algorithm $\mathcal{A}$ from an initial configuration c_0 under ψ. Execution $E_{\mathcal{A}}(c_0, \psi)$ is infinite, or ends in final configuration c_{final} where every agent's state is s_{final}. In c_{final}, all the agents give the same output that whether the g-partial gathering problem is solved or it is not solvable from the given initial configuration. In the randomized case, define an infinite sequence $\phi = R_0, R_1, \ldots$, where $R_i = r^i_0, r^i_1, \ldots r^i_{k-1}$ is a sequence of k random numbers such that each r^i_j is chosen uniformly at random from R by agent a_j. Then, given $c_0 \in C$ and $\psi = A_0, A_1, \ldots$, a *probabilistic execution* of algorithm $\mathcal{A}$ is defined as $E_{\mathcal{A}}(c_0, \psi) = c_0, c_1, \ldots$ where $c_i \xrightarrow{A_i, R_i} c_{i+1}$ holds for any $i \geq 0$. That is, in the execution, each number in R_i is chosen randomly and configurations change deterministically. Note that the scheduler cannot choose ϕ but can choose ψ *depending on* ϕ in an adversarial way. In both the cases, we assume that the scheduler is *fair*, that is, each agent is activated infinitely often. In addition, when all the agents are activated and they perform steps in every configuration, that is, $A_i = A$ holds for every i, the execution is called *synchronous*. Otherwise, i.e., if $A_i = A$ may not necessarily hold, the execution is called *asynchronous*. In this paper, we consider asynchronous executions. Note that a synchronous execution is a special case of asynchronous executions.

Moreover, we define *periodic initial configurations* as follows. These definitions are used in Section 3. At first, we define the i-th ($i \neq 0$) forward (resp., backward) agent a' of agent a as the agent such that $i-1$ agents exist between a and a' in a's forward (resp., backward) direction. We call the a's 1-st forward and backward agents *nearest agents* of a. For convenience, we define the 0-th forward (or backward) agent of a as a itself. Then, for an initial configuration c_0 we assume that agents $a_0, a_1, \ldots, a_{k-1}$ exist in this order, that is, a_i is the i-th forward agent of a_0 in c_0. Then, we define the *distance sequence* of agent a_i in c_0 as $D_i(c_0) = (d^i_0(c_0), \ldots d^i_{k-1}(c_0))$, where $d^i_j(c_0)$ is the distance from the j-th forward agent of a_i to the $(j+1)$-st forward agent of a_i in c_0. Note that the k-th forward agent of a_i is a_i itself (i.e., $d^i_{k-1}(c_0)$ is the distance from $a_{(i-1) \bmod k}$ to a_i). In addition, we define the distance sequence $D(c_0)$ of configuration c_0 as the lexicographically minimum sequence among $\{D_i(c_0) | a_i \in A\}$. Moreover, let $\mathit{shift}(D, x) = (d_x, d_{x+1}, \ldots, d_{k-1}, d_0, d_1, \ldots, d_{x-1})$ for sequence $D = (d_0, d_1, \ldots, d_{k-1})$. Then, we say that a configuration c_0 is *periodic* if $D(c_0) = \mathit{shift}(D(c_0), x)$ holds for some x $(0 < x < k)$. Otherwise, we say c_0 is *aperiodic*. For an initial con-

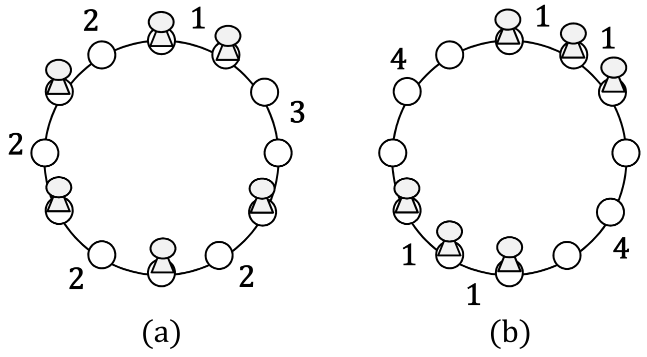


Figure 2: (a): An example of an aperiodic configuration, (b): An example of a periodic configuration

figuration c_0, we define the period $\mathit{peri}(c_0)$ of c_0 as the minimum positive integer x satisfying $D(c_0) = \mathit{shift}(D(c_0), x)$. Note that $\mathit{peri}(c_0) = k$ holds for any aperiodic initial configuration c_0. For example, Fig. 2 (a) is an aperiodic configuration with $D(c_0) = (1, 3, 2, 2, 2, 2)$ and $\mathit{peri}(c_0) = 6\ (= k)$, and Fig. 2 (b) is a periodic configuration with $D(c_0) = (1, 1, 4, 1, 1, 4)$ and $\mathit{peri}(c_0) = 3$.

2.3. Partial gathering problem

The requirement of the partial gathering problem is that, for a given integer $g\ (2 \le g \le k/2)$, agents terminate in a configuration such that either at least g agents or no agent exists at each node. Formally, we define the g-partial gathering problem as follows.

Definition 1. *Execution E solves the g-partial gathering problem when the following conditions hold:*

- *Execution E is finite (i.e., all agents terminate in state s_{final}).*
- *In the final configuration, at least g agents exist at any node where an agent exists.*

Definition 2. *A deterministic algorithm $\mathcal{A}$ solves the g-partial gathering problem if $E_{\mathcal{A}}(c_0, \psi)$ solves the problem for any c_0 and ψ.*

Definition 3. *A randomized algorithm $\mathcal{A}$ solves the g-partial gathering problem with probability 1 if $E_{\mathcal{A}}(c_0, \psi)$ solves the problem with probability 1 for any c_0 and ψ.*

In [16], the following lower bound on a total number of agent moves for the g-partial gathering problem in ring networks is shown. This theorem holds for both deterministic and randomized algorithms.

Theorem 1. [16] *The total number of agent moves required to solve the g-partial gathering problem in ring networks is $\Omega(gn)$ if $g \geq 2$.*

3. Deterministic g-partial gathering

In this section, we consider the deterministic case. First, we show that agents cannot detect whether the given initial configuration is one of unsolvable configurations shown in [16] (i.e., for the case of agents with knowledge of k) or not. Then, we propose an algorithm that solves the problem from any initial configuration other than the unsolvable initial configurations mentioned above. Thus, the set of unsolvable configurations is the same as that of [16].

3.1. Impossibility result

In this section, we show that agents cannot detect whether the given initial configuration is one of unsolvable configurations shown in [16] or not.

Theorem 2. *There exists no algorithm that solves the g-partial gathering problem from a periodic initial configuration c_0 with $peri(c_0)$ less than g. In addition, there exists no algorithm such that an agent terminates execution of the algorithm and detects whether the initial configuration has $peri(c_0)$ less than g or not.*

Proof. We can directly derive the former argument from [16]. In the proof of [16], agents are anonymous and they have knowledge of k. Intuitively, the argument holds as the following reason. In the case of anonymous agents, they can break symmetry only by using distance sequences of the initial configuration. However, in a (periodic) initial configuration with $peri(c_0)$, for each q $(0 \leq q \leq peri(c_0) - 1)$, agents $a_q, a_{q+peri(c_0)}, \ldots, a_{q+\ell\times peri(c_0)-1}$ $(\ell = k/peri(c_0))$ always execute the same action simultaneously in the synchronous execution (i.e., the execution such that the scheduler activates all the agents and makes them execute actions in each step). Then, they cannot break the symmetry (or periodicity) and any pair of them cannot gather at the same node. That is, there exist at least ℓ nodes where at least one agent exists in the finial configuration. Then, if $peri(c_0) < g$ holds, the value of ℓ is large and

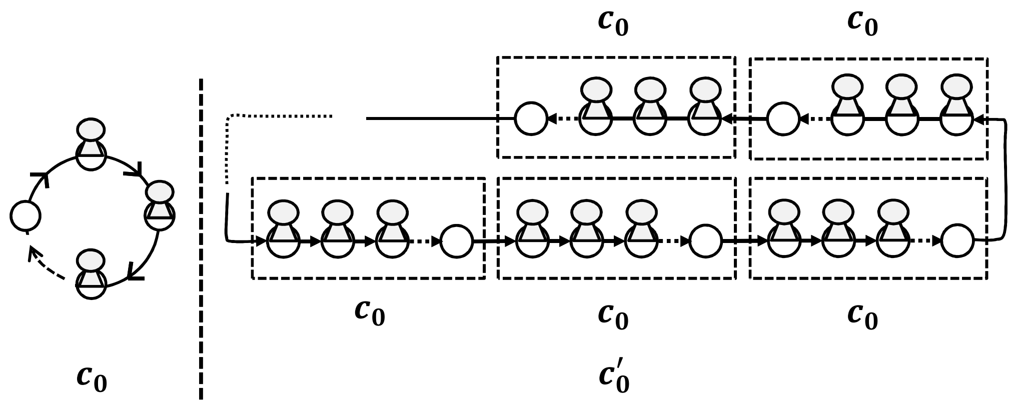


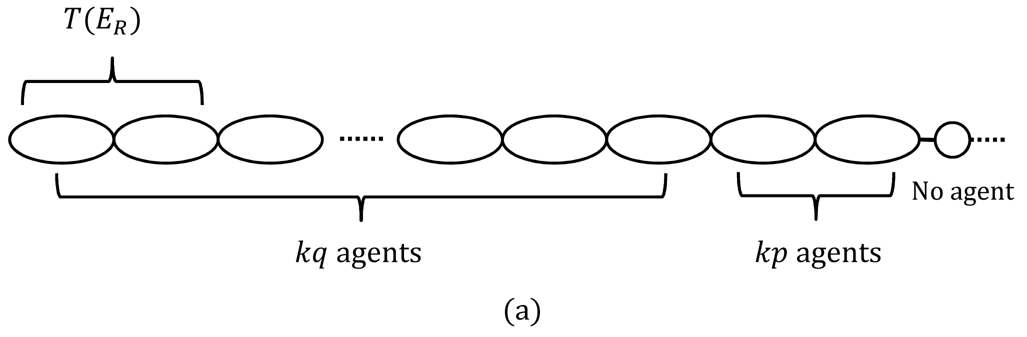
Figure 3: Simple examples of c_0 and c_0'.

there exist many nodes where at least one agent exists, and agents cannot g-partial gathering however the remaining ($= k - \ell$) agents behave.

In the following, we show that agents cannot detect whether the given initial configuration c_0 has $peri(c_0)$ less than g or not. Intuitively, this argument holds when we consider a periodic initial configuration c_0 with $peri(c_0) \leq g - 1$ and an aperiodic initial configuration c_0' (or a larger ring) with $peri(c_0') = k\,(> g)$ that consecutively includes c_0 sufficiently many times as a part of c_0' (see Fig. 3). Then, there exists some pair of agents a and a' such that a (resp., a') starts execution of the algorithm from c_0 (resp., c_0') and the algorithm execution of a' is the exactly same as that of a. Then, a' detects that c_0' has $peri(c_0') \leq g - 1$ and the g-partial gathering problem cannot be solved from c_0', nevertheless it is solvable from c_0'. Thus, agents cannot terminate execution of the algorithm with correctly detecting whether the given initial configuration c_0 has $peri(c_0)$ less than g or not.

From the two paragraphs, we show the concrete proof of the above idea. We prove it by contradiction. We use a similar idea to that in [14], which considers the total gathering problem and shows that agents without any knowledge cannot terminate execution of the algorithm with correctly detecting whether the total gathering problem can be solvable from the given initial configuration or not. For simplicity, we assume that agents move in a synchronous manner. At first, let us consider an n-node ring $R = (V, E)$ and a periodic initial configuration c_0 such that its $peri(c_0)$ is less than g. We assume that k agents $a_0, a_1, \ldots, a_{k-1}$ exist in this order. Let $V = \{v_0, v_1, \ldots, v_{n-1}\}$. Then, from the first argument of this theorem, the g-partial gathering problem cannot be solved from c_0, and agents can detect the fact by the hypothesis. Let E_R be the execution until all the agents detect unsolvability of the problem from c_0. In addition, we define $T(E_R)$ as the length of E_R and denote $E_R = c_0, c_1, \ldots, c_{T(E_R)}$. Then, the number of nodes where at least one agent exists in $c_{T(E_R)}$ is either of $k/peri(c_0), 2k/peri(c_0), \ldots, k$ because (i) for each $q\,(0 \leq q \leq peri(c_0) - 1)$, agents $a_q, a_{q+peri(c_0)}, \ldots,\ a_{q+\ell\times peri(c_0)-1}$ $(\ell = k/peri(c_0))$ always execute the same action simultaneously and any pair of them never gathers at the same node, and (ii) if the number of nodes where either of agents $a_0, a_1, \ldots, a_{peri(c_0)}$ exists in $c_{T(E_R)}$ is $h'\,(1 \leq h' \leq peri(c_0))$, the number of nodes where either of agents $a_{peri(c_0)\times\ell'}, a_{peri(c_0)\times\ell'+1}, \ldots, a_{peri(c_0)\times\ell'+peri(c_0)-1}$ $(1 \leq \ell' \leq k/peri(c_0) - 1)$ exists in $c_{T(E_R)}$ is also h' for each ℓ'. We assume that there exist $hk/peri(c_0)$ $(1 \leq h \leq peri(c_0))$ nodes where at least one agent exists in $c_{T(E_R)}$. Then, at most $peri(c_0) - (h - 1)$ agents exist at each node.

Next, let us consider an aperiodic initial configuration c'_0 such that (i) consecutively includes c_0 sufficiently many times as a part of c'_0 using $kq + kp$ agents for some large integer q and a relatively small integer p, and (ii) includes one node where no agents exists (this makes c'_0 aperiodic). An abstract expression of c'_0 is shown in Fig. 4 (a). Then, each of the kq agents executes the exactly same action as one of the k agents in c_0, there exist $hkq/peri(c_0)$ nodes where at least one of the kq agents exists in $c'_{T(E_R)}$, and at most $peri(c_0) - (h - 1)$ agents exist at each of the $hkq/peri(c_0)$ nodes. In this case, we show that, however the remaining kp agents behave, it is impossible that at least g agents gather at each of the $hkq/peri(c_0)$ nodes. More concretely, we consider the ring $R' = (V', E')$ consisting of $pn + qn + 1$ nodes, where p is the minimum integer satisfying $pn \geq T(E_R)$ and q is the minimum integer satisfying $p/q < (g - peri(c_0) - (h-1)) \times peri(c_0)/h$, respectively. The inequality $p/q < (g - peri(c_0) - (h-1)) \times peri(c_0)/h$ follows from $kp/(hkq/peri(c_0)) < g - (peri(c_0) - (h - 1))$. The construction of R' from R is shown in Fig. 4 (b). Let $V' = \{v'_0, v'_1, \ldots, v'_{pn+qn}\}$. We consider the initial configuration c'_0 such that $kq + kp$ agents $a'_0, a'_1, \ldots, a'_{kq-1}, a'_{kq}, \ldots a'_{kq+kp-1}$ exist in this order in R'. In addition, we define the initial position of each agent in R' as follows. Let $v_{f(i)}$ be the node where agent a_i initially stays in R. Then, we assume that agent a'_i initially stays at node $v'_{f(i \bmod k)+n\cdot\lfloor i/k \rfloor}$. That is, the initial positions for R are repeated from v'_0 to $v'_{pn+qn-1}$, and there is no agent at node v'_{pn+qn}. Then, we can show later that each agent $a'_i\,(0 \leq i \leq kq - 1)$ always executes the same action as agent a_i in R and terminates execution of the algorithm with recognizing that the value of $peri(c'_0)$



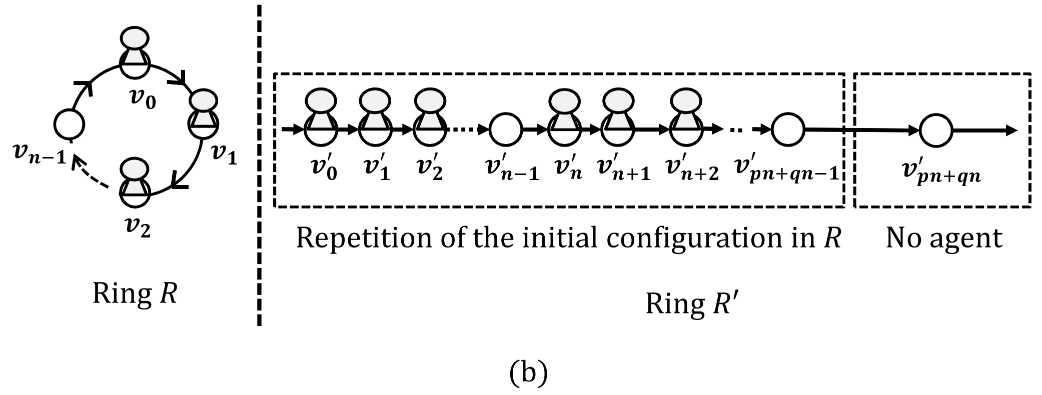


Figure 4: (a): Abstract expression of c'_0, (b): Construction of R from R'

is less than g and the g-partial gathering problem cannot be solved from c'_0. Actually, agents cannot achieve g-partial gathering after the kq agents execute such behaviors because (i) there exist $hkq/peri(c_0)$ nodes where at least one of the kq agents exists, (ii) at most $peri(c_0) - (h-1)$ agents exist at each of $hkq/peri(c_0)$ nodes, and (iii) however the remaining kp agents behave, it is impossible that at least g agents gather at each of the $hkq/peri(c_0)$ nodes since $p/q < (g - peri(c_0) - (h-1)) \times peri(c_0)/h$ holds. However, the g-partial gathering problem is solvable from c'_0 since c'_0 is an aperiodic initial configuration and the value of $peri(c'_0)$ is $kq + kp\,(> g)$, which is a contradiction.

To complete the proof, we need to show that each agent $a'_i\,(0 \le i \le kq-1)$ in R' always executes the same action as agent a_i in R. To show this, for each node v'_j in R', we define $c_v(v'_j) = v_{j \bmod n}$ as the corresponding node of v'_j in R. We also define the *local configuration* of node v as the 2-tuple that consists of the state of v and the states of all agents at v. Then, the following lemma holds.

Lemma 1. *Let us consider execution $E_{R'} = c'_0, c'_1, \ldots, c'_{T(E_R)}, \ldots$ for ring R'. We define $V'_t = \{v'_t, v'_{t+1}, \ldots, v'_{pn+qn-2}\}$. For any $t \leq T(E_R)$, configuration c'_t satisfies the following condition: for each $v'_j \in V'_t$, the local configuration of v'_j in c'_t is the same as that of $c_v(v'_j)$ in c_t.*

Proof. We prove the lemma by induction on t. For $t = 0$, Lemma 1 holds from the definition of R'. Next, we show that when Lemma 1 holds for $t\,(t < T(E_R))$, Lemma 1 holds also for $t + 1$.

From the hypothesis, for each $v'_j \in V'_{t+1}$ the local configurations of v'_{j-1} and v'_j in c'_t are the same as those of $c_v(v'_{j-1})$ and $c_v(v'_j)$ in c_t, respectively. Since agents have no knowledge of k or n, agents staying at v'_{j-1} and v'_j in c'_t execute the same actions as those at $c_v(v'_{j-1})$ and $c_v(v'_j)$ in c_t, respectively. Since only agents staying at nodes v'_{j-1} and v'_j can change the local configuration of v'_j in unidirectional rings[1], the local configuration of v'_j in c'_{t+1} is the same as that of $c_v(v'_j)$ in c_{t+1}. Therefore the lemma follows. □

Now we complete the proof of the theorem using Lemma 1. In $c'_{T(E_R)}$ local configuration of each node in $V^* = \{v'_{pn}, v'_{pn+1}, \ldots, v'_{qn+pn-2}\} \subseteq V'_{T(E_R)}$ is the same as that of the corresponding node in $c_{T(E_R)}$. That is, the states of agents staying at nodes in V^* is equal to those of the corresponding agents in V. Hence, every agent in V^* (kq agents in total) recognizes that the value of $peri(c'_0)$ is less than g and the g-partial gathering problem cannot be solved in configuration $c'_{T(E_R)}$, and it terminates execution of the algorithm. Then, there exist $hkq/peri(c_0)$ nodes where an agent exists. However, it is impossible that at least g agents gather at each of the $hkq/peri(c_0)$ nodes because $p/q < (g - peri(c_0) - (h-1)) \times peri(c_0)/h$ holds, which is a contradiction. Therefore the theorem holds. □

3.2. Proposed algorithm

In this section, we propose a deterministic algorithm that solves the g-partial gathering problem in a total number of $O(gn)$ moves from any initial configuration other than unsolvable configurations discussed in Section 3.1 (i.e., any initial configuration c_0 with $peri(c_0) \geq g$). The algorithm comprises two parts. In the first part, some agents are elected as leader agents by

[1]Recall that there is no in-transit agent in the synchronous execution we are considering.

executing a leader agent election algorithm partially. In the second part, the leader agents tell the other agents their meeting points, and the other agents move to that node.

3.2.1. The first part: leader election

The aim of this part is similar to [16], that is, to elect some leaders that satisfy the following properties: 1) At least one agent is elected as a leader, and 2) at least $g-1$ non-leader agents exist between two leaders. Each agent takes a status from the following three statuses:

- active: the agent is performing 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.

Initially, all agents are active. At first, we explain the idea of leader election in [16] because the proposed algorithm this time is based on the idea in [16]. In [16], the network is a unidirectional ring, agents have distinct IDs, and each node has a whiteboard. The leader election in [16] comprises several phases. In each phase, each active agent a_i moves in the ring until it observes IDs of two active agents by using whiteboards. Then, a_i observes IDs of three successive active agents including a_i itself, say id_1, id_2, id_3 in this order. Note that id_1 is the ID of a_i. Thereafter, a_i compares the value of id_2 and the values of id_1 and id_3. If id_2 is the largest among the three IDs, a_i remains active (as a candidate for leaders) in the next phase. Otherwise, it becomes inactive and drops out from the set of leader candidates. By this behavior, we can guarantee that, if a_i remains active, its nearest forward and backward active agents never remain active because of distinct IDs[2]. Hence, at least half of the current 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 [15], after executing j phases, at least $2^j - 1$ inactive agents exist between any two active agents. Thus, after executing

[2]There may exist several agents executing different phases since we consider an asynchronous execution. Agents treat this problem by controlling phases through whiteboards (the detail is described in the pseudocode part).

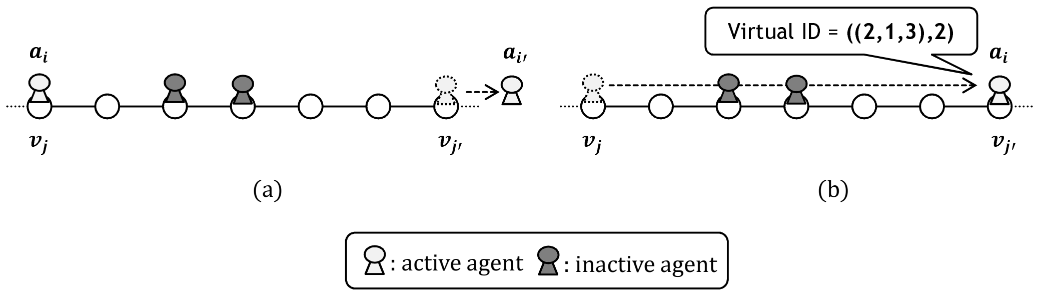


Figure 5: An example of a virtual ID.

$\lceil \log g \rceil$ phases, the following properties are satisfied: 1) At least one agent remains active, and 2) the number of inactive agents between any two active agents is at least $g - 1$. Therefore, all the remaining active agents become leaders.

In the following, we explain the way to apply the above leader election to anonymous agents. In this section, agents use *virtual IDs*. Then, similar to the above idea, in each phase active agents this time compare virtual IDs and determine whether they remain active in the next phase or not. To define the virtual ID, we introduce active nodes and inactive nodes. At the beginning of each phase, there exists an active agent, an inactive agent, or no agent at each node. Then, we call a node where an active (resp., an inactive) agent exists at the beginning of the phase an *active node* (resp., an *inactive node*). Note that it is not possible that some node is both an active node and an inactive node in some phase because agents occupy different nodes at the beginning of each phase. Then, a virtual ID is given in the form of $(disArray[\,], nInactive)$, where $disArray[\,]$ and $nInactive$ are a distance sequence and the number of inactive nodes between active nodes, respectively. Concretely, we assume that active agent a_i starts some phase at node v_j and $v_{j'}$ is v_j's forward active node. Here, we say that $v_{j'}$ is v_j's forward (resp., backward) active node if no active node exists from v_j to $v_{j'}$ (resp., $v_{j'}$ to v_j). In addition, let $v^1_{ina}, v^2_{ina}, \dots, v^\ell_{ina}$ be inactive nodes between v_j and $v_{j'}$. That is, nodes $v_j\,(= v^0_{ina}), v^1_{ina}, v^2_{ina}, \dots, v^\ell_{ina}, v_{j'}\,(= v^{\ell+1}_{ina})$ exist in this order. Then, when a_i moves from v_j to $v_{j'}$, it observes a distance sequence $(d_0, d_1, \dots, d_\ell)$, where $d_m\,(0 \le m \le \ell)$ is the distance from v^m_{ina} to v^{m+1}_{ina}. Then, a_i gets $disArray[\,] = (d_0, d_1, \dots, d_\ell)$ and $nInactive = \ell$ as its virtual ID. For example, in Fig. 5 (a), we assume that active agents a_i and $a_{i'}$ exist at node v_j and $v_{j'}$ at the beginning of some phase, respectively, and $a_{i'}$ already left $v_{j'}$. From

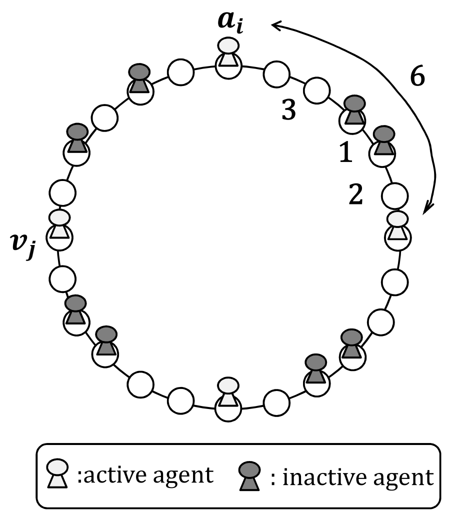


Figure 6: An example of comparison of virtual IDs.

(a) to (b), active agent a_i moves twice to visit an inactive node, moves once to visit the next inactive node, and moves three times to visit the next active node $v_{j'}$. Hence, it gets its virtual ID ((2,1,3),2). Note that each active agent can detect whether the current node is an active node, an inactive node, or another node using whiteboards. Each active agent moves until it observes such three virtual IDs. Note that multiple agents may have the same virtual ID since agents use distance sequences and numbers of inactive agents as virtual IDs. We explain the behavior in this case next.

After observing three virtual IDs id_1, id_2, id_3, each active agent a_i compares the virtual IDs by the lexicographical order and decides whether it remains active (as a candidate for leaders) in the next phase or not. Different from [16], multiple agents may have the same virtual ID. To treat this, when at least one virtual ID differs from other IDs, if $id_2 > \max(id_1, id_3)$ or $id_2 = id_3 > id_1$ holds, a_i remains active. Otherwise, a_i becomes inactive. When all the three virtual IDs are the same (i.e., $id_1 = id_2 = id_3$ holds), a_i compares the value of $nInactive$ with the value of g. If $nInactive \geq g - 1$ holds, it still remains active. Otherwise, it becomes inactive. Note that even if a_i becomes inactive, it does not happen that all the active agents become inactive because we consider an aperiodic initial configuration c_0 ($peri(c_0) = k$) or a periodic configuration with $peri(c_0) \geq g$ (the detail is described in Lemma 2). For example, in Fig. 6 when an active agent a_i moves to node v_j, it observes three virtual IDs and the value of the IDs are the

Table 2: Global variables used in the proposed algorithm

Variables for agent a_i

type	name	meaning	initial value
int	$a_i.id_1$ $a_i.id_2$ $a_i.id_3$	virtual ID	0
int	$a_i.phase$	phase number of a_i	0
int	$a_i.nInactive_1$ $a_i.nInactive_2$ $a_i.nInactive_3$	number of inactive agents in a virtual ID	0
array	$a_i.disArray_1[\,]$ $a_i.disArray_2[\,]$ $a_i.disArray_3[\,]$	distance sequence in a virtual ID	$\perp$

Variables for node v_j

type	name	meaning	initial value
int	$v_j.phase$	phase number of v_j	0
boolean	$v_j.initial$	existence of an agent at v_j in c_0	*true* if an agent exists in c_0 *false* otherwise
boolean	$v_j.inactive$	existence of an inactive agent at v_j (this value is set to true if an inactive agent exists at v_j)	*false*

same ((3,1,2),2). Then, a_1 remains active if the value 2 of *nInactive* is $g-1$ or larger (or $g \leq 3$), and becomes inactive if the value is smaller than $g-1$ (or $g \geq 4$).

To summarize, in each phase active agents move in the ring and get three virtual IDs, and determine whether they remain active in the next phase or not. They execute such a phase $\lceil \log g \rceil$ times and complete the leader election. Then, we can show that at least $g-1$ inactive agents exist between any two active agents (Lemma 2). Intuitively, this is because 1) when three IDs have different values, nearest active agents never remain active, and 2) when at least two IDs have the same value and two nearest active agents a_i and a_j remain active, at least $g-1$ inactive agents already exist between a_i and a_j. Thus, all the remaining active agents become leaders.

The pseudocode of the proposed algorithm in the first part is described in Algorithm 1. Global variables used in the algorithm are summarized in Table 2. In addition, in Algorithm 1, agents use procedure *NextActive*() to move to the next active node and get a virtual ID. The pseudocode of *NextActive*() is described in Procedure 1. In line 6 of Procedure 1, variables $a_i.pahse$ and $v_j.phase$ are used to treat asynchrony (i.e., to avoid agents from overtaking other agents). Note that, there exists no inactive agent in the first phase (i.e.,

Algorithm 1 Behavior of active agent a_i in the first part (v_j is the current node of a_i.)

Main Routine of Agent a_i

```
1:  while true do
2:      $a_i.phase := a_i.phase + 1, v_j.phase := a_i.phase$
3:      $(a_i.disArray_1[\,], a_i.nInactive_1) := NextActive()$
4:      $a_i.id_1 := (a_i.disArray_1[\,], a_i.nInactive_1)$
5:      $(a_i.disArray_2[\,], a_i.nInactive_2) := NextActive()$
6:      $a_i.id_2 := (a_i.disArray_2[\,], a_i.nInactive_2)$
7:      $(a_i.disArray_3[\,], a_i.nInactive_3) := NextActive()$
8:      $a_i.id_3 := (a_i.disArray_3[\,], a_i.nInactive_3)$
9:      if $(a_i.id_2 > \max(a_i.id_1, a_i.id_3)) \vee (a_i.id_2 = a_i.id_3 > a_i.id_1) \vee ((a_i.id_1 = a_i.id_2 = a_i.id_3) \wedge (a_i.nInactive_2 \geq g-1))$ then
10:         if $a_i.phase = \lceil \log g \rceil$ then terminate the first part and enter the second part with a leader status
11:     else
12:         $v_j.inactive := true$
13:         terminate the first part and enter the second part with an inactive status
14:     end if
15: end while
```

all the agent are active), and lines 13 – 16 are mainly executed in Procedure 1. Thus, when an active agent visits the next active node, lines 18 – 19 are executed and the agent gets a virtual ID $((d), 0)$, where d is the distance between two active nodes.

Concerning leader election, we have the following lemmas.

Lemma 2. *Algorithm 1 eventually terminates and satisfies the following two properties when it terminates.*

- *At least one leader agent exists.*
- *At least $g-1$ inactive agents exist between any two leader agents.*

Proof. Termination of Algorithm 1. When an active agent a_i does not satisfy the condition of line 9, it becomes inactive. If a_i satisfies the condition $\lceil \log g \rceil$ times, it becomes a leader. Thus, all the agents eventually change their statuses to an inactive status or a leader status, and Algorithm 1 eventually

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

```
Variables for Agent $a_i$
int $h = 0$;
int $distance = 0$;
int $nInactive = 0$;
array $disArray[\,]$;
Behavior of Agent $a_i$
1: let $l$ be the maximum integer such that $disArray[l] \neq 0$ holds
2: for (int $m = 0$, $m \leq l$; $m{+}{+}$) $disArray[m] := 0$
3: move to the next node (that becomes new $v_j$)
4: $distance := distance + 1$
5: while $(v_j.initial = false) \vee (v_j.inactive = true) \vee ((v_j.initial = true) \wedge (v_j.inactive = false) \wedge (v_j.phase \neq a_i.phase))$ do
6:   if $(v_j.initial = true) \wedge (v_j.inactive = false) \wedge (v_j.phase \neq a_i.phase)$ then wait at the current node $v_j$ until $(v_j.inactive = true) \vee (v_j.phase = a_i.phase)$ holds
7:   if $v_j.inactive = true$ then
8:     $disArray[h] := distance$
9:     $nInactive := nInactive + 1$
10:    $distance := 0$
11:    $h := h + 1$
12:  end if
13:  if $(v_j.initial = false) \vee (v_j.inactive = true)$ then
14:    move to the next node (that becomes new $v_j$)
15:    $distance := distance + 1$
16:  end if
17: end while
18: $disArray[h] := distance$
19: return ($disArray[\,]$, $nInactive$)
```

terminates.

Existence of at least one leader agent. In order to show this, we define $Y^1 = Y$ and $Y^{l+1} = Y^l \cdot Y$ (concatenation) for sequence Y. We show this first for the case that the initial configuration is aperiodic, and then the case that the initial configuration is periodic but solvable. If the initial configuration is aperiodic, $D(c_0) = shift(D(c_0), x)$ does not holds for any $x (0 < x < k)$ by the definition of aperiodic initial configurations. Thus, there exists no way to partition the whole distance sequence into subsequences with the same elements. Therefore, when at least two active agents exist in some phase, there exists an agent a_i such that $a_i.id_2 > \max(a_i.id_1, a_i.id_3)$ or $a_i.id_2 = a_i.id_3 > a_i.id_1$ holds and such an agent remains active. In addition, when there exists exactly one active agent in some phase, the active agent a_i observes three IDs such that $a_i.id_1 = a_i.id_2 = a_i.id_3$ and $a_i.nInactive = k-1 > g-1$ hold, and thus a_i remains active. The second equality follows from the fact that all the agents other than a_i are inactive in the phase. Thus, after executing $\lceil \log g \rceil$ phases, at least one active agent exists and it becomes a leader.

Next, we consider the case that the initial configuration c_0 is periodic and $peri(c_0) \geq g-1$ holds. In this case, $k = \ell \times peri(c_0)$ holds for some positive integer ℓ. Let D' be the aperiodic distance sequence such that $D(c_0) = (D')^{\ell}$ holds. Then, agent $a_{i'}$ $(0 \leq i' \leq peri(c_0)-1)$ and each of $a_{i'+q\times peri(c_0)}$ $(1 \leq q \leq \ell-1)$ execute the same action. Hence, without loss of generality, we consider the behavior of the agent set $A' = \{a_0, a_1, \dots, a_{peri(c_0)-1}\}$. Then, since D' is aperiodic, when at least two active agents exist among A' at the beginning of some phase, there exists at least one agent $a_{i'}$ such that $a_{i'}.id_2 > \max(a_{i'}.id_1, a_{i'}.id_3)$ or $a_{i'}.id_2 = a_{i'}.id_3 > a_{i'}.id_1$ holds since D' is aperiodic, and such an agent keeps an active status at the end of the phase. Thus, it does not happen that all the agents in A' become inactive. In addition, when exactly one agent $a_{i'}$ remains active among A' at the beginning of some phase, due to the periodicity of the initial configuration, it observes three IDs $a_{i'}.id_1, a_{i'}.id_2$, and $a_{i'}.id_3$ such that 1) $a_{i'}.id_1 = a_{i'}.id_2 = a_{i'}.id_3$ holds, and 2) $a_{i'}.nInactive$ of each ID is $peri(c_0) \geq g-1$. Thus, $a_{i'}$ still remains active at the end of the phase. Moreover, once $a_{i'}$ observes such IDs, it continues to observe the same ID until the leader election is completed. Thus, after executing $\lceil \log g \rceil$ phases, at least one agent remains active and such an agent becomes a leader.

Existence of at least $g-1$ inactive agents between two leaders. We show this for both the cases that the initial configuration is aperiodic and periodic (but solvable) together. Let $a_i.id_1, a_i.id_2, a_i.id_3$ be IDs that a_i observes in some phase. If a_i remains active in the phase, either of (i) $a_i.id_2 > \max(a_i.id_1, a_i.id_3)$, (ii) $a_i.id_2 = a_i.id_3 > a_i.id_1$, or (iii) $(a_i.id_1 = a_i.id_2 = a_i.id_3) \wedge (nInactive \geq g-1)$ holds. In the case of (iii), at least $g-1$ inactive agents already exist between two active agents. Hence, in the following we assume that a_i remains active by satisfying case (i) or (ii).

Let a_i be an active agent and $a_{i'}$ (resp., $a_{i''}$) be the a_i's backward (resp., forward) active agent. At first, we show that if a_i remains active in some phase, either of the following conditions is satisfied: (a) Both $a_{i'}$ and $a_{i''}$ drop out from the set of leader candidates, or (b) agent $a_{i'}$ drops out from the set of leader candidates, but $a_{i''}$ remains active in the phase and at least $g-1$ inactive agents already exist between a_i and $a_{i''}$. If $a_i.id_2 > \max(a_i.id_1, a_i.id_3)$ holds and a_i remains active (i.e., case (i)), $a_{i'}$ and $a_{i''}$ clearly drop out from the set of leader candidates and condition (a) is satisfied. In the following, we consider case (ii), that is, $a_i.id_2 = a_i.id_3 > a_i.id_1$ holds and a_i remains active. Let $a_{i'}.id_1, a_{i'}.id_2, a_{i'}.id_3$ be IDs observed by $a_{i'}$. Then, $a_{i'}.id_2 = a_i.id_1$ and $a_{i'}.id_3 = a_i.id_2$ hold. Since $a_{i'}.id_2 < a_{i'}.id_3$ holds, $a_{i'}$ does not satisfy the condition to remain active and it drops out from the set of leader candidates. Next, let $a_{i''}.id_1, a_{i''}.id_2, a_{i''}.id_3$ be IDs observed by $a_{i''}$. Then, $a_{i''}.id_1 = a_i.id_2$ and $a_{i''}.id_2 = a_i.id_3$ hold. If $a_{i''}.id_3 \neq a_{i''}.id_2$ holds, or $a_{i''}.id_3 = a_{i''}.id_2$ holds and the value of *nInactive* in each ID is less than $g-1$, $a_{i''}$ does not satisfy the condition to remain active and it drops out from the set of leader candidates. Hence, condition (a) is satisfied. If $a_{i''}.id_3 = a_{i''}.id_2$ holds and the value of *nInactive* in each ID is at least $g-1$, $a_{i''}$ satisfies the condition to remain active and then at least $g-1$ inactive agents already exist between a_i and $a_{i''}$. Hence, condition (b) is satisfied.

Now, we show that at least $g-1$ inactive agents exist between any two leader agents. We first show that after executing j phases, at least $2^j - 1$ inactive agents exist between any 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 any two active agents as mentioned above. For the case of $j = l\,(\leq \lceil \log g \rceil - 1)$, we assume that at least $2^l - 1$ inactive agents exist between any two active agents. Let a_i be an active agent and $a_{i'}$ (resp., $a_{i''}$) be the a_i's backward (resp., forward) active agent. After executing phase $l+1$, if a_i remains active by satisfying condition (a), $a_{i'}$ and $a_{i''}$ become inactive. Then, the number of inactive agents between any two active agents is at least $(2^l - 1) + 1 + (2^l - 1) = 2^{l+1} - 1$. If a_i remains active by satisfying condition (b), $a_{i'}$ similarly becomes inactive, and $a_{i''}$ also becomes inactive or remains active but at least $g-1$ inactive agents already exist between a_i and $a_{i''}$. Then, since $g > l$ holds, the number of inactive agents between any two active agents is at least $2^{l+1} - 1$. Thus, after executing j phases, at least $2^j - 1$ inactive agents exist between any two active agents. Therefore, after executing $\lceil \log g \rceil$ phases, at least $g-1$ inactive agents exist between any two leader agents. $\Box$

Lemma 3. *Algorithm 1 requires a total number of $O(n \log g)$ moves.*

Proof. From lines 3 – 8 in Algorithm 1, in each phase each active agent moves until it visits three active nodes and gets three IDs of active agents. This requires an $O(n)$ moves in total because each communication link is passed by three times. Since each agent executes such a phase $\lceil \log g \rceil$ times, the lemma follows. $\square$

3.2.2. The second part: leaders' instructions and agents' movement

After leader election, agents achieve g-partial gathering by leader agents' instructions. At the beginning of this part, there exists a leader agent, an inactive agent, or no agent at each node. We call a node where a leader agent exists a *leader node*. We use the same technique as in [16]. We assume that a leader agent a_i starts this part at leader node v_j, $v_{j'}$ is v_j's next leader node, and inactive nodes $v^1_{ina}, v^2_{ina}, \ldots v^{\ell}_{ina}$ exist between v_j and $v_{j'}$. That is, nodes $v_j, v^1_{ina}, v^2_{ina}, \ldots, v^{\ell}_{ina}, v_{j'}$ exist in this order. Then, in brief, leader agent a_i at v_j moves to the next leader node $v_{j'}$. During the movement, a_i marks node v^t_{ina} as a non-gathering node (resp., a gathering node) if $(t+1) \mod g \neq 0$ (resp., $(t+1) \mod g = 0$). That is, each leader continues to mark some consecutive $g-1$ inactive nodes as non-gathering nodes and mark the next inactive node as a gathering node until it visits the next leader node. Note that v_j and $v_{j'}$ are marked as non-gathering nodes by leader agents. Then, there exist at least $g-1$ non-gathering nodes between any two gathering nodes. Thus, agents can achieve g-partial gathering by the way that each agent moves to the nearest gathering node. The proof of correctness is shown in [16]. Intuitively, this is because 1) at least $g-1$ inactive agents exist between two leader agents by Lemma 2, and 2) after each leader moves in the ring with marking some consecutive $g-1$ nodes as non-gathering nodes and marking the next node as a gathering node, at least $g-1$ non-gathering nodes also exist between two gathering nodes. Also, this part can be achieved in a total number of $O(gn)$ moves since each link is passed by at most $2g$ times. By this fact and Lemma 3, we have the following theorem.

Theorem 3. *From any solvable initial configuration, the proposed algorithm solves the g-partial gathering problem in a total number of $O(gn)$ moves.*

4. Randomized g-partial gathering

In this section, for the case of $2 \leq g \leq k/2$, we propose a randomized algorithm that solves the g-partial gathering problem with probability 1 in a total number of $O(gn)$ (i.e., optimal) moves in expectation from any initial configuration. The basic idea is the same as that of the previous section, that is, agents elect multiple leaders by comparing (virtual) IDs. In this section, since we consider a randomized algorithm, agents can use random numbers as IDs. In addition, each agent compares $2g-1$ random IDs at one time instead of comparing three IDs as in the previous section[3]. By this behavior, if there exists a unique maximum ID among the consecutive $2g-1$ IDs, agents can make a configuration such that at least $g-1$ non-leader agents exist between two leaders. However, if two or more IDs have the same maximum value among the $2g-1$ IDs, agents cannot make such a configuration. Agents treat this case by additional behaviors explained in the following subsections.

The algorithm comprises two parts. In the first part, agents determine some candidate nodes for g-partial gathering using random IDs. In the second part, agents determine gathering nodes from the candidate nodes and achieve g-partial gathering.

4.1. The first part: determination of candidate nodes for g-partial gathering

In this part, agents determine candidate nodes for gathering using random IDs. In the following, we refer to "candidate nodes for gathering" as "candidate nodes". Each agent takes a status from the following three statuses:

- active: the agent moves in the ring to determine candidate nodes.
- leader: the agent elects the current node as a candidate node.
- waiting: the agent is staying at a candidate node and waiting for the next instruction.

Initially, all agents are active. This part comprises several (or possibly infinite) phases. At the beginning of each phase, each active agent a_i creates a $\lceil 7\log g \rceil$-bit random ID, and writes the ID and the current phase number on

[3]Although agents in the previous section can elect multiple leaders by comparing $2g-1$ virtual IDs at one time, we did not follow the same method because (i) simulating the leader election of [16] (i.e., agents compare three IDs at one time) is a simple way and (ii) it is complicated to treat the case that an agent observes multiple (four or more) same virtual IDs in a deterministic manner.

the current whiteboard. Thereafter, a_i moves until it observes $2g-1$ random IDs (including the one it creates) of the current phase. Let $id_1, id_2, \dots, id_{2g-1}$ be the IDs. Then, a_i compares id_g (or the ID at the middle of the ID sequence) with the other observed IDs. If id_g is the unique maximum ID among the $2g-1$ IDs, it becomes a leader. Different from the behavior of a leader in the previous section, a leader agent a_i in this section sets the flag $v_j.candi$ to declare that the current node v_j is a candidate node and waits at v_j until at least g agents gather at v_j. Note that a_i can check the number of agents staying at v_j by the whiteboard. The reason why leaders execute such a behavior is that, even if some two agents a_i and $a_{i'}$ become leaders, it is possible that only less than $g-1$ non-leader agents exist between a_i and $a_{i'}$ (such a case does not happen in Section 3.2). In this case, there exists another leader a_h such that at least $g-1$ non-leader agents exist between a_h and a_h's backward leader agent and a_h treats this situation (the detail is explained in the next part). If id_g is not the unique maximum ID, that is, id_g is not the maximum or id_g is the maximum but another ID has the same value as id_g, a_i additionally moves until it observes g IDs of the current phase to check whether a candidate node exists within this range or not. If a_i visits a candidate node v_j (or $v_j.candi = true$) during the movement, it enters a waiting status there. Otherwise, a_i executes the next phase. Each active agent repeats such a behavior until it becomes a leader and sets $v_j.candi = true$, or visits some candidate node v_j (or $v_j.candi = true$) and enters a waiting status.

An example is given in Fig. 7. Each number represents a random ID written on the whiteboard. For simplicity, we assume that there exists an agent at each node in the initial configuration and random IDs are already written in (a). We consider behaviors of agents a_1, a_2, a_3, and a_4. From (a) to (b), each active agent moves until it observes five $(= 2g-1)$ random IDs. Since agent a_3 observes random IDs (3,1,5,4,4) and the middle ID 5 is the unique maximum, it becomes a leader and sets a candidate flag at v_j. On the other hand, the other agents continue to move in the ring. From (b) to (c), agents a_1, a_2, and a_4 move to observe additional three $(= g)$ random IDs. During the movement, since agents a_1 and a_2 observe a candidate flag at v_j, they enter a waiting status there. On the other hand, since a_4 does not observe a flag, it updates a random ID and proceeds to the next phase (the four IDs from the rightmost node are similarly updated). From (c) to (d), since a_4 observes five random IDs (2,3,5,2,4) and the middle ID 5 is the unique maximum, it becomes a leader and sets the candidate flag at $v_{j'}$.

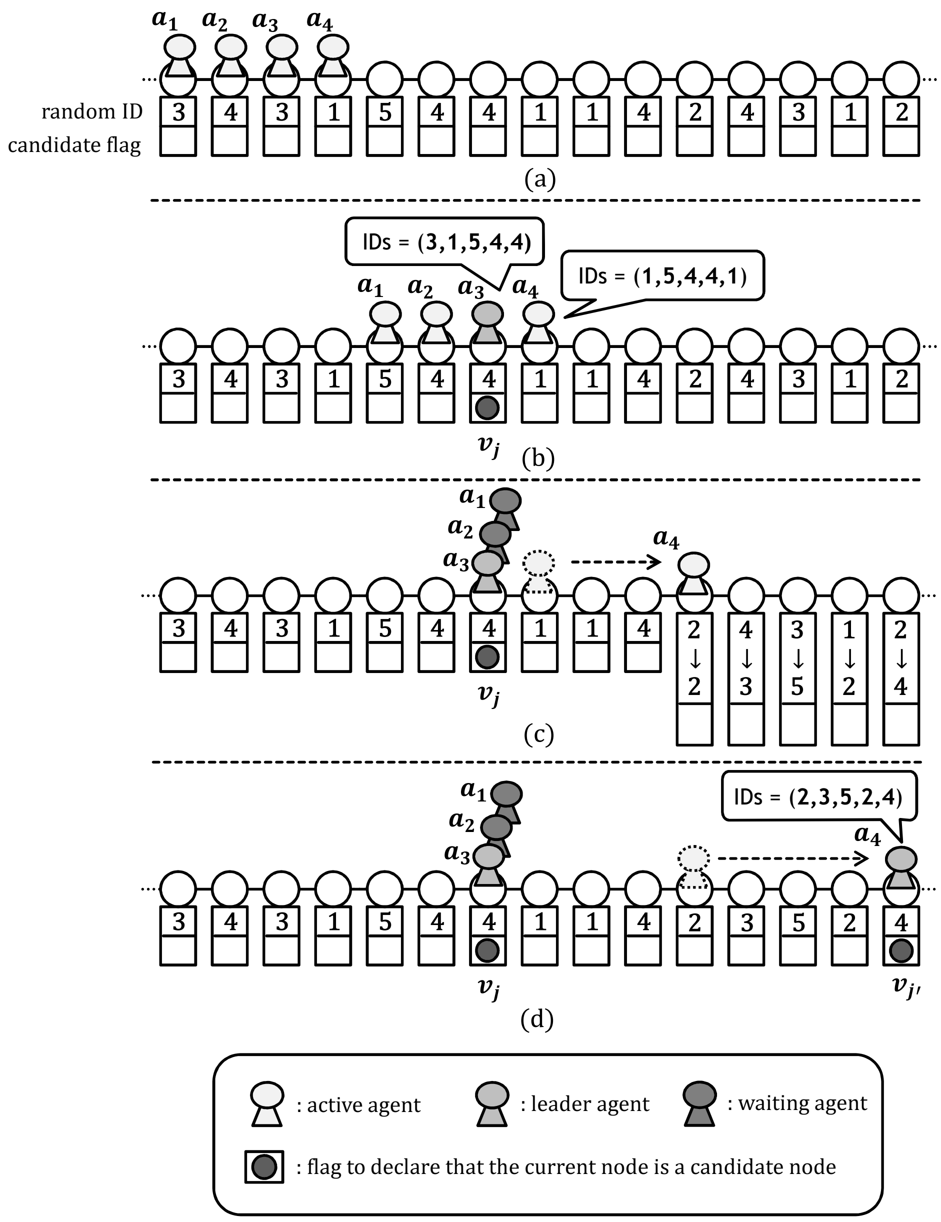


Figure 7: Behavior outlines of the first part for the case of $g = 3$.

However, no agent exists between v_j and $v_{j'}$. This situation is handled in the second part.

The pseudocode of active agents in the first part is described in Algo-

Table 3: Global variables used in the proposed algorithm

Variables for agent a_i

type	name	meaning	initial value
int	$a_i.phase$	phase number of a_j	0
int	$a_i.nIDs$	number of random IDs that a_i has observed	0
array	$a_i.id[\,]$	sequence of random IDs that a_i has observed	$\perp$
int	$a_i.nAgentsTemp$	number of agents that eventually visit and stay at the current node	0

Variables for node v_j

type	name	meaning	initial value
int	$v_j.phase$	phase number of v_j	0
int	$v_j.nAgents$	number of agents staying at v_j	0
int	$v_j.id$	random ID stored by v_j	$\perp$
int	$v_j.nVisited$	number of agents that visited v_j	0
boolean	$v_j.candi$	whether v_j is a candidate node or not	*false*
boolean	$v_j.activeExist$	existence of an active agent	*false*
boolean	$v_j.lVisited$	whether v_j is visited by a leader agent or not	*false*

rithm 2. Global variables used in the algorithm are summarized in Table 3. In addition, in Algorithm 2, agents use procedure *NextActive2*() to move to the next active node, and to enter a waiting status when visiting a candidate node v_j (or $v_j.candi = true$). The pseudocode of *NextActive2*() is described in Procedure 2. In lines 9 – 10 of Procedure 2, variables $a_i.nIDs$ and $v_j.nVisited$ are used to treat asynchrony (i.e., to avoid agents from overtaking other agents). In addition, node v_j has a boolean variable $v_j.activeExist$ representing whether an active agent exists at v_j or not. This variable is used to ensure that an agent executing the second part does not overtake an active agent that is still executing the first part (the detail is described in Section 4.2). When an active agent a_i completes the first part, it begins the second part by executing procedure *Realization*(), whose explanation is described in the next section. We have the following lemmas concerning Algorithm 2.

Lemma 4. *Algorithm 2 eventually terminates with probability 1 and all agents enter a leader status or a waiting status.*

Proof. We prove the lemma by contradiction, that is, we assume that some agent a_i retains the active status and continues to move in the ring. This implies that a_i does not visit a candidate node v_j (or $v_j.candi = true$) even if a_i continues to move in the ring. Thus, by Algorithm 2, all agents need to retain the active status and at least two random IDs need to have the same

Algorithm 2 Behavior of active agent a_i in the first part (v_j is the current node of a_i.)

Main Routine of Agent a_i

```
1: while true do
2:   $a_i.phase := a_i.phase + 1$, $v_j.phase := a_i.phase$
3:   $a_i.id[0] := random(\lceil 7 \log g \rceil)$, $v_j.id := a_i.id[0]$
4:   $a_i.nIDs := 1, v_j.nVisited := 1$
5:   while $a_i.nIDs < 2g - 1$ do
6:     $NextActive2()$
7:     $a_i.id[a_i.nIDs] := v_j.id$
8:     $a_i.nIDs := a_i.nIDs + 1$, $v_j.nVisited := v_j.nVisited + 1$
9:   end while
10:  if $\forall h \in [0, 2g-2] \setminus \{g-1\}; a_i.id[g-1] > a_i.id[h]$ then
11:    $v_j.activeExist := false$
12:    $v_j.candi := true$
13:    $v_j.nAgents := 1$
14:    terminate the first part and enter the second part by executing
       $Realization\ (leader)$
15:  else
16:    while $a_i.nIDs < 3g - 1$ do
17:      $NextActive2()$
18:      $a_i.nIDs := a_i.nIDs + 1$, $v_j.nVisited := v_j.nVisited + 1$,
19:    end while
20:  end if
21: end while
```

value among any consecutive $2g - 1$ random IDs. Otherwise, some active agent sets $v_j.candi = true$, and a_i observes the flag and enters a leader or waiting status. Note that, since $2 \leq g \leq k/2$ and $2g - 1 < k$ hold, when some two random IDs have the same value among some consecutive $2g - 1$ random IDs, it means that the IDs are created by mutually different agents. Since each active agent selects its random ID from $\lceil 7 \log g \rceil$-bits by line 2 in Algorithm 2, the probability that two random IDs have the same value is at most $(1/2)^{7\lceil \log g \rceil} \leq (1/g)^7$. Thus, the probability that at least two random IDs have the same value among some $2g - 1$ consecutive IDs is at most $\binom{2g-1}{2}(1/g)^7 < 2g^2/g^7 = 2/g^5$. Since $g \geq 2$ holds, the probability is at most $1/16$. This means that, by repeatedly executing phases, eventually no two IDs have the same value among some $2g-1$ consecutive IDs with probability

Procedure 2 void *NextActive*2() (v_j is the current node of a_i.)

```
Behavior of Agent $a_i$
1: $v_j.activeExist := false$
2: move to the next node (that becomes new $v_j$)
3: $v_j.activeExist := true$
4: while $v_j.initial = false$ do
5:     $v_j.activeExist := false$
6:     move to the next node (that becomes new $v_j$)
7:     $v_j.activeExist := true$
8: end while
9: if $(v_j.phase \neq a_i.phase) \wedge (v_j.candi = false)$ then wait until $(v_j.phase = a_i.phase) \vee (v_j.candi = true)$ holds
10: if $(a_i.nIDs \neq v_j.nVisited) \wedge (v_j.candi = false)$ then wait at $v_j$ until $a_i.nIDs = v_j.nVisited$ holds
11: if $v_j.candi = true$ then
12:     $v_j.activeExist := false$
13:     $v_j.nAgents := v_j.nAgents + 1$
14:     terminate the first part and enter the second part by executing Realization(waiting)
15: end if
```

1. However, this contradicts the assumption and the lemma follows. □

Lemma 5. *After all the agents complete execution of Algorithm 2, at least one candidate node has at least g agents.*

Proof. Let a_i be the first agent that becomes a leader and sets a candidate flag among all the agents, v_j be the node where a_i sets the flag, and $id_0, \ldots id_{2g-2}$ be the random $2g-1$ IDs observed by a_i. Then, by lines 10 – 14 in Algorithm 2, id_{g-1} is the unique maximum among the $2g-1$ IDs and thus a_i sets a candidate flag at v_j. We assume that agents $a_{i-(g-1)}, a_{i-(g-2)}, \ldots, a_{i-1}, a_i$ exist in this order. Then, agents $a_{i-(g-1)}, a_{i-(g-2)}, \ldots, a_{i-1}$ do not set a candidate flag since the value of id_h $(0 \leq h \leq g-2)$ is smaller than that of id_{g-1}, and they move until they observe g IDs or a candidate flag. During the movement, they observe a candidate flag at node v_j and enter waiting statuses there. Then, there exist one leader agent a_i and $g-1$ agents $a_{i-(g-1)}, a_{i-(g-2)}, \ldots, a_{i-1}$ at v_j. Thus, at least g agents exist at a candidate node v_j and the lemma follows. □

4.2. The second part: achievement of g-partial gathering

In this part, agents achieve g-partial gathering based on the candidate nodes. Each agent takes a status from the following three statuses:

- leader: the agent checks whether the current candidate node finally becomes a gathering node or not. If the node becomes a gathering node, the agent instructs waiting agents where they should move.
- waiting: the agent is waiting for the leader's instruction.
- moving: the agent moves to its gathering node.

We consider a situation such that all agents complete the first part, stay at some candidate nodes, and never move from the beginning of this part[4]. Then, there exist some (possibly one) candidate nodes, and at least one of them has at least g agents (Lemma 5) and some candidate nodes may have less than g agents each. Note that at each candidate node exactly one leader agent exists and the other agents are waiting agents. We denote a set of candidate nodes with at least g (resp., less than g) agents in the above situation by V_{candi}^{more} (resp., V_{candi}^{less}). Then, the basic movement of this part is as follows. Each leader agent a_i at a node in V_{candi}^{more} moves to the next candidate node $v_j \in V_{candi}^{more}$. During the movement, when a_i visits a candidate node $v_{j'} \in V_{candi}^{less}$, it sets a flag $v_{j'}.lVisited$ to declare that $v_{j'}$ is visited by a leader. Then, all the agents at $v_{j'}$ move to the nearest candidate node $v_{j''}$ such that the number of agents existing between $v_{j'}$ and $v_{j''}$ is at least g. After the movement, agents achieve g-partial gathering.

First, we explain the behavior of leader agents. Each leader agent a_i first waits at the current node v_j until either (i) at least g agents gather at v_j or (ii) $v_j.lVisited$ is set to true. Note that a_i can count the number of agents staying at v_j using variable $v_j.nAgents$ (i.e., a_i need not have the capability to count the number of agents staying at the same node). Then, (i) if at least g agents gather at v_j, a leader agent a_i moves to the next candidate node $v_{j'}$ (or $v_{j'}.candi = true$) and sets $v_{j'}.lVisited = true$. If at least $g-1$ waiting agents exist at $v_{j'}$ (i.e., $v_{j'} \in V_{candi}^{more}$), a_i terminates the algorithm at $v_{j'}$ because each of the at least $g-1$ agents at $v_{j'}$ eventually terminates the algorithm there and this guarantees that at least g agents gather at $v_{j'}$. If less

[4]We consider the situation for explanation, and it is possible that some agents are executing the second part while the other agents are still executing the first part.

than $g-1$ waiting agents exist at $v_{j'}$ (i.e., $v_{j'} \in V^{less}_{candi}$), a_i stores the number of the waiting agents at $v_{j'}$ to a variable $a_i.nAgentsTemp$. This operation means that $a_i.nAgentsTemp$ (possibly 0) agents move to the next candidate node. Thereafter, a_i moves to the next candidate node $v_{j''}$ and updates the number of agents that are to stay at $v_{j''}$ (or agents that will eventually gather at $v_{j''}$) using $a_i.nAgentsTemp$. If the updated number of agents at $v_{j''}$ is at least g, $v_{j''}$ eventually becomes a gathering node regardless of whether $v_{j''}$ is in V^{more}_{candi} or V^{less}_{candi}. In this case, a_i resets $a_i.nAgentsTemp$ to 0. If the updated number is less than g, $v_{j''}$ becomes a non-gathering node and a_i stores the number of agents that will eventually gather at $v_{j''}$ to $a_i.nAgentsTemp$. After updating the value of $a_i.nAgentsTemp$, a_i moves to the next candidate node. Each leader agent repeats such a behavior until it visits a candidate node v_j in V^{more}_{candi}. On the other hand, (ii) if $v_j.lVisited$ is set to true at node v_j where a leader agent a_i starts this part, this means that another leader agent $a_{i'}$ visits v_j and v_j is in V^{less}_{candi}. In this case, a_i enters a waiting status, whose behavior is described next.

Next, we explain the behaviors of waiting agents and moving agents. Each waiting agent a_i stays at the current node v_j until some leader agent visits v_j and sets $v_j.lVisited = true$. Then, it checks (from the whiteboard content) whether at least g agents will eventually gather at v_j or not. If at least g agents gather, a_i terminates the algorithm there. Otherwise, a_i enters a moving status. Each moving agent a_i moves to the next candidate node $v_{j'}$ and enters a waiting status there. Then, the number of agents that will eventually gather at $v_{j'}$ is the sum of agents that will visit $v_{j'}$ from v_j and agents that already stay at $v_{j'}$. Thus, if the number of agents that will eventually gather at $v_{j'}$ is at least g, a_i terminates the algorithm there. Otherwise, a_i enters a moving status again. Each moving agent repeats such a behavior until it visits a candidate node v_j such that at least g agents eventually gather at v_j. When all agents terminate the algorithm, the final configuration is a solution of the g-partial gathering problem.

An example is given in Fig. 8. For simplicity, we omit nodes where agents do not exist. From (a) to (b), since each number of agents at nodes v^1_{candi} and v^4_{candi} is $4\,(\geq g)$, leader agent a_1 (resp., a_4) moves to the next candidate node v^2_{candi} (resp., the next candidate node of v^4_{candi}). On the other hand, since each number of agents at v^2_{candi} and v^3_{candi} is less than g, leader agents a_2 and a_3 keep staying at the current nodes. Then, the system reaches the configuration of (c) and a flag declaring that the node is visited by a leader agent is set at v^2_{candi}. Since the number of agents at v^2_{candi} except for a_1

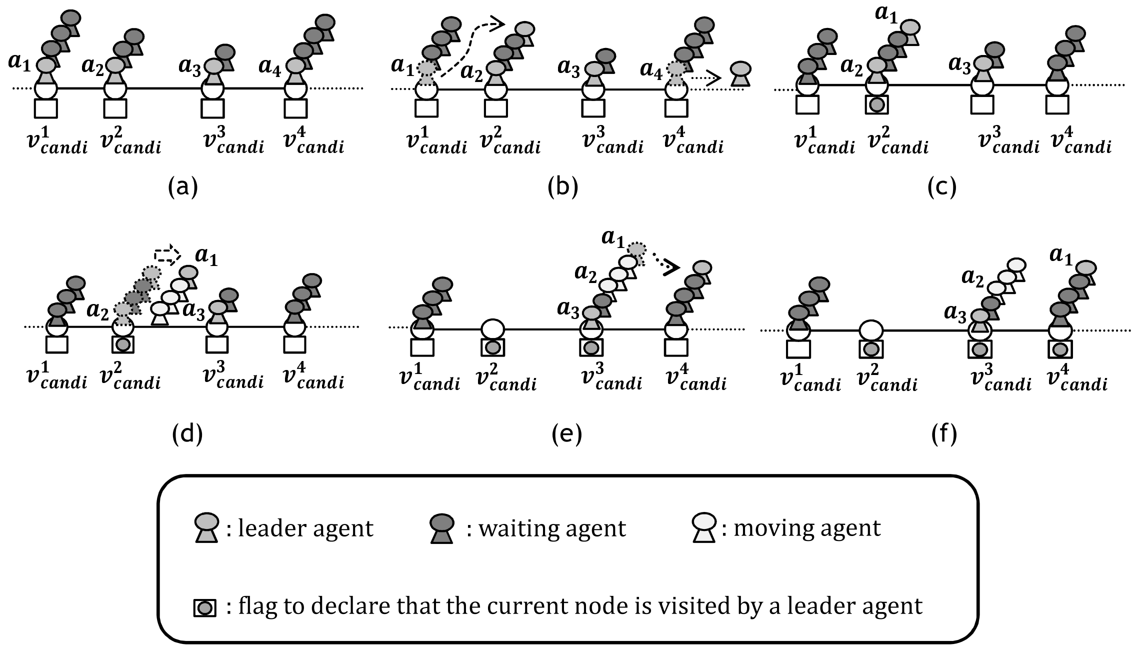


Figure 8: Behavior outlines of the second part for the case of $g = 4$.

is $3 < g$, all the agents (including a leader a_2) at v^2_{candi} move to the next candidate node v^3_{candi} (d). Then, the system reaches the configuration of (e) and a flag is set by a_1. Since the number of agents at v^3_{candi} except for a_1 is $5 > g$, a_1 moves to the next candidate node v^4_{candi} and the other agents at v^3_{candi} terminate the algorithm there. In (f), a_1 sets a flag at v^4_{candi}. Since the number of agents that set the flag (i.e., a_1) or already stay at v^4_{candi} is $4 = g$, all the agents at v^4_{candi} terminate the algorithm there.

Agents achieve the second part (or g-partial gathering) by procedure *Realization*(), whose pseudocode is described in Procedure 3. The behavior of leader agents is described in lines 1 – 23. Note that, since at least g agents gather to at least one candidate node by Lemma 5, at least one leader agent executes lines 6 – 23. In addition, in Procedure 3, if an active agent executing the first part exists at the current node v_j, a leader agent waits at v_j until the active agent leaves v_j (lines 9 – 18). This is realized by variable $v_j.ActiveExist$, whose usage is described in Procedure *NextActive*2(). The behavior of waiting (resp., moving) agents is described in lines 24 – 27 (resp., lines 28 – 32). We have the following lemmas for the proposed algorithm.

Lemma 6. *The proposed algorithm solves the g-partial gathering problem*

Procedure 3 void *Realization* (int: *status*) (v_j is the current node of a_i.)

Main Routine of Agent a_i

```
1: if $status = leader$ then
2:   wait at $v_j$ until $(v_j.lVisited = true) \vee (v_j.nAgents \geq g)$ holds
3:   if $v_j.lVisited = true$ then
4:     // another leader agent visits $v_j$
5:     enter a waiting status and execute $Realization$ ($waiting$)
6:   else
7:     // at least $g$ agents gather at $v_j$
8:     move to the next node (that becomes new $v_j$)
9:     if $v_j.activeExist = true$ then wait at $v_j$ until $v_j.activeExist = false$
       holds
10:    while $(v_j.candi = false) \vee (v_j.nAgents < g)$ do
11:      if $(v_j.candi = true) \wedge (v_j.nAgents < g)$ then
12:        // $a_i$ visits a candidate node in $V^{less}_{candi}$
13:        $v_j.lVisited := true$, $v_j.nAgents := v_j.nAgents + a_i.nAgentsTemp$
14:        if $v_j.nAgents \geq g$ then $a_i.nAgentsTemp := 0$
15:        else $a_i.nAgentsTemp := v_j.nAgents$
16:      end if
17:      move to the next node (that becomes new $v_j$)
18:      if $v_j.activeExist = true$ then wait at $v_j$ until $v_j.activeExist = false$
         holds
19:    end while
20:    // $a_i$ reaches a candidate node in $V^{more}_{candi}$
21:    $v_j.lVisited := true$
22:    terminate the algorithm
23:  end if
24: else if $status = waiting$ then
25:   wait at $v_j$ until $v_j.lVisited = true$ holds
26:   if $v_j.nAgents \geq g$ then terminate the algorithm
27:   else enter a moving status and execute $Realization$ ($moving$)
28: else if $status = moving$ then
29:   move to the next node (that becomes new $v_j$)
30:   while $v_j.candi = false$ do move to the next node (that becomes new
      $v_j$)
31:   enter a waiting status and execute $Realization$ ($waiting$)
32: end if
```

with probability 1 from any initial configuration.

Proof. By lemma 5, there exists at least one candidate node after executing Algorithm 2. Let $v^1_{candi}, v^2_{candi}, \dots v^\ell_{candi}$ be the candidate nodes. Note that exactly one leader agent stays at each $v^j_{candi}(1 \leq j \leq \ell)$ at the beginning of Algorithm 3. We assume that $v^1_{candi}, v^2_{candi}, \dots, v^\ell_{candi}$ exist in this order. In addition, let $na^j_{candi}(1 \leq j \leq \ell)$ be the number of agents staying at v^j_{candi}. When $na^j_{candi} \geq g$ holds (i.e., $v^j_{candi} \in V^{more}_{candi}$), leader agent a_i at v^j_{candi} moves to the next candidate node v^{j+1}_{candi} by lines 6 – 23 in Procedure 3. If $na^{j+1}_{candi} \geq g$ holds, at least g agents exist at v^{j+1}_{candi} including a_i, and all the agents at v^{j+1}_{candi} terminate the algorithm there. If $na^{j+1}_{candi} < g$ holds, a_i and all the agents (including the leader staying at v^{j+1}_{candi} at the beginning of Procedure 3) at v^{j+1}_{candi} move to v^{j+2}_{candi}. Leader agent a_i repeats such a behavior until it visits the next candidate node $v_{j'} \in V^{more}_{candi}$, and each of the other agents repeats such a behavior until it reaches a node $v^{j''}_{candi}$ such that $(\sum_{p=j+1}^{j''-1} na^p_{candi} < g) \wedge (\sum_{p=j+1}^{j''} na^p_{candi} \geq g)$ holds. Agents staying between $v^{j"+1}_{candi}$ and $v_{j'}$ behave in a similar manner. Hence, after the movement, all agents eventually terminate the algorithm and clearly at least g agents exist at each node where agents exist. Thus, the lemma follows. □

Lemma 7. *The expected total number of moves of the proposed algorithm is* $O(gn)$*.*

Proof. First, we analyze the expected total number of moves in the first part. In each phase, each active agent a_i moves until it observes $2g-1$ random IDs. If a_i does not become a leader, it additionally moves until it observes another g random IDs or visits some candidate node v_j (or $v_j.candi = true$). This requires a total number of $O(gn)$ moves since each link is passed by at most $3g$ times. In addition, by Lemma 4, the probability that at least two IDs have the same maximum value among some consecutive $2g-1$ IDs and agents proceed to the next phase is at most 1/16. Thus, letting $\mathrm{E}[M_{first}]$ be the expected total number of moves in the first part, we have the following inequality:

$$\mathrm{E}[M_{first}] \leq 3gn + \frac{1}{16}\mathrm{E}[M_{first}] \tag{1}$$

From the inequality, we have $\mathrm{E}[M_{first}] \leq (16/5)gn$.

Next, we analyze the total number of moves in the second part. Each leader agent a_i staying at a candidate node $v^j_{candi} \in V^{more}_{candi}$ moves to the nearest node in V_{candi}^{more}. This clearly requires a total number of n moves because no two leader agents traverse the same link. In addition, each agent staying at a candidate node $v_j \in V_{candi}^{less}$ moves to its gathering node or waits at v_j until at least g agents gather at v_j depending on the value of $v_j.nAgents$. Because of the behavior of leader agents explained above, this movement requires a total number of $O(gn)$ moves since each link is passed by at most $2g$ times. Hence, the total number M_{second} of moves in the second part is $O(gn)$. Therefore the lemma follows. □

By Lemmas 6 and 7, we have the following theorem.

Theorem 4. *The proposed algorithm solves the g-partial gathering problem with probability 1 in a total number of $O(gn)$ moves in expectation from any initial configuration.*

5. Conclusion

In this paper, we considered the g-partial gathering problem for anonymous agents without global knowledge in asynchronous unidirectional ring networks. We considered deterministic and randomized cases. First, in the deterministic case, we showed that there exist unsolvable initial configurations and agents cannot detect whether the initial configuration is an unsolvable one or not. In addition, we proposed an algorithm that solves the problem from any solvable initial configuration in a total number of $O(gn)$ moves. In the randomized case, we proposed an algorithm that solves the problem with probability 1 in a total number of $O(gn)$ moves in expectation from any initial configuration. Thus, our algorithms can solve the problem in the asymptotically optimal total number of moves without global knowledge.

Future works are as follows. First, we plan to investigate the possibility to solve the g-partial gathering problem without knowledge of k or n in a total number of $O(gn)$ moves not *in expectation* but *with high probability* (i.e., probability $1 - O(1/n)$). Lemma 4 guarantees that the proposed randomized algorithm this time achieves g-partial gathering in a total number of $O(gn)$ moves only with a constant probability. It is interesting and still open whether this algorithm (or some other algorithm) achieves g-partial gathering in a total number of $O(gn)$ moves with high probability or not. Second, we will consider improvements of the space complexity on whiteboards. In the deterministic proposed algorithm, each whiteboard requires $O(\log \log g)$ space to maintain phase numbers. In the randomized proposed algorithm, each whiteboard requires at least $\Omega(\log g)$ space to store random IDs and phase numbers that may infinitely increase. We will consider whether such space complexities can be shown to be asymptotically optimal (in the deterministic case) or can be reduced by introducing another randomization (in the randomized case). Finally, we will consider g-partial gathering from initial configurations such that two or more agents exist at the same node. We conjecture that agents can achieve g-partial gathering also from such configurations by maintaining the number of agents using whiteboards.

References

[1] D. Baba, T. Izumi, F. Ooshita, H. Kakugawa, and T. Masuzawa. Linear time and space gathering of anonymous mobile agents in asynchronous trees. Theoretical Computer Science, 478:118–126, 2013.

[2] J. Baumann, F. Hohl, K. Rothermel, and M. Straßer. Mole–concepts of a mobile agent system. World Wide Web, 1(3):123–137, 1998.

[3] G. Cabri, L. Leonardi, and F. Zambonelli. Mobile agent coordination for distributed network management. Journal of Network and Systems Management, 9(4):435–456, 2001.

[4] Y. Dieudonné and A. Pelc. Anonymous meeting in networks. Algorithmica, 74(2):908–946, 2016.

[5] Y. Dieudonné, A. Pelc, and V. Villain. How to meet asynchronously at polynomial cost. SIAM Journal on Computing, 44(3):844–867, 2015.

[6] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in a ring. Proceedings of Latin American Symposium on Theoretical Informatics, pages 599–608, 2004.

[7] P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory. Proceedings of International Symposium on Distributed Computing, pages 242–256, 2008.

[8] L. Gasieniec, E. Kranakis, D. Krizanc, and X. Zhang. Optimal memory rendezvous of anonymous mobile agents in a unidirectional ring. Proceedings of International Conference on Current Trends in Theory and Practice of Computer Science, pages 282–292, 2006.

[9] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'agents: Applications and performance of a mobile-agent system. Softw., Pract. Exper., 32(6):543–573, 2002.

[10] E. Kranakis, D. Krizanc, and E. Markou. Mobile agent rendezvous in a synchronous torus. Proceedings of Latin American Symposium on Theoretical Informatics, pages 653–664, 2006.

[11] E. Kranakis, D. Krozanc, and E. Markou. The Mobile Agent Rendezvous Problem in the Ring. Synthesis Lectures on Distributed Computing Theory, Vol. 1. 2010.

[12] E. Kranakis, N. Santoro, C. Sawchuk, and D. Krizanc. Mobile agent rendezvous in a ring. Proceedings of the 23rd International Conference on Distributed Computing Systems, pages 592–599, 2003.

[13] D.B. Lange and M. Oshima. Seven good reasons for mobile agents. Communications of the ACM, 42(3):88–89, 1999.

[14] F. Ooshita, S. Kawai, H. Kakugawa, and T. Masuzawa. Randomized gathering of mobile agents in anonymous unidirectional ring networks. IEEE Transactions on Parallel and Distributed Systems, 25(5):1289–1296, 2014.

[15] G. L. Peterson. An $O(n \log n)$ unidirectional algorithm for the circular extrema problem. ACM Transactions on Programming Languages and Systems, 4(4):758–762, 1982.

[16] M. Shibata, S. Kawai, F. Ooshita, H. Kakugawa, and T. Masuzawa. Partial gathering of mobile agents in asynchronous unidirectional rings. Theoretical Computer Science, 617:1–11, 2016.

[17] M. Shibata, D. Nakamura, F. Ooshita, H. Kakugawa, and T. Masuzawa. Partial gathering of mobile agents in arbitrary networks. IEICE Transactions on Information and Systems, 102(3):444–453, 2019.

[18] M. Shibata, F. Ooshita, H. Kakugawa, and T. Masuzawa. Move-optimal partial gathering of mobile agents in asynchronous trees. Theoretical Computer Science, 705:9–30, 2018.