Uniform deployment of mobile agents in asynchronous rings

Uniform deployment of mobile agents in asynchronous rings[☆,☆☆]

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

[a] *Kyushu Institute of Technology, 680-4, Kawadu, 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 Information Science, NAIST Takayama 8916-5, Ikoma, Nara 630-0192, Japan*

Abstract

In this paper, we consider the uniform deployment problem of mobile agents in asynchronous unidirectional rings, which requires the agents to uniformly spread in the ring. The uniform deployment problem is in striking contrast to the rendezvous problem which requires the agents to meet at the same node. While rendezvous aims to break the symmetry, uniform deployment aims to attain the symmetry. It is well known that the symmetry breaking is difficult in distributed systems and the rendezvous problem cannot be solved from some initial configurations. Hence, we are interested in clarifying what difference the uniform deployment problem has on the solvability and the number of agent moves compared to the rendezvous problem. We consider two problem settings, with knowledge of k (or n) and without knowledge of k or n where k is the number of agents and n is the number of nodes. First, we consider agents with knowledge of k (or n since k and n can be easily obtained if one of them is given). In this case, we propose two algorithms.

[☆]The conference version of this paper is published in the proceedings of 29th ACM Symposium on Principles of Distributed Computing (PODC 2016).

[☆☆]This work was supported by JSPS KAKENHI Grant Numbers 24500039, 26280022, 26330084, 15H00816, and 16K00018.

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

Email addresses: shibata@cse.kyutech.ac.jp (Masahiro Shibata), f-oosita@is.naist.jp (Fukuhito Ooshita), kakugawa@ist.osaka-u.ac.jp (Hirotsugu Kakugawa), masuzawa@ist.osaka-u.ac.jp (Toshimitsu Masuzawa)

Preprint submitted to Journal of Parallel and Distributed Computing February 19, 2018

The first algorithm solves the uniform deployment problem with termination detection. This algorithm requires $O(k \log n)$ memory space per agent, $O(n)$ time, and $O(kn)$ total moves. The second algorithm also solves the uniform deployment problem with termination detection. This algorithm reduces the memory space per agent to $O(\log n)$, but uses $O(n \log k)$ time, and requires $O(kn)$ total moves. Both algorithms are asymptotically optimal in terms of total moves since there are some initial configurations such that agents require $\Omega(kn)$ total moves to solve the problem. Next, we consider agents with no knowledge of k or n. In this case, we show that, when termination detection is required, there exists no algorithm to solve the uniform deployment problem. For this reason, we consider the relaxed uniform deployment problem that does not require termination detection, and we propose an algorithm to solve the relaxed uniform deployment problem. This algorithm requires $O((k/l) \log(n/l))$ memory space per agent, $O(n/l)$ time, and $O(kn/l)$ total moves when the initial configuration has symmetry degree l. This means that the algorithm can solve the problem more efficiently when the initial configuration has higher symmetric degree (i.e., is closer to uniform deployment). Note that all the proposed algorithms achieve uniform deployment from any initial configuration, which is a striking difference from the rendezvous problem because the rendezvous problem is not solvable from some initial configurations.

Keywords: distributed system, mobile agent, uniform deployment, ring networks, token, symmetry degree

---

1. Introduction

1.1. Background and motivation

A *distributed system* consists of a set of computers (*nodes*) connected by communication links. As a promising design paradigm of distributed systems, (mobile) agent systems have attracted a lot of attention [1, 2]. Agents can traverse the system carrying information collected at visiting nodes and process tasks on each node using the information. In other words, agents can encapsulate the process code and data, which simplifies design of distributed systems [3, 4].

In this paper, we consider the *uniform deployment* (or *uniform scattering*) *problem* as a fundamental problem for coordination of agents. This problem requires all agents to spread uniformly in the network. From a practical

point of view, uniform deployment is useful for the network management. In a distributed system, it is necessary that regularly each node gets software updates and is checked whether some application installed on the node is running correctly or not [5, 6]. Hence, considering agents with such services, uniform deployment guarantees that agents visit each node at short intervals and provide services. Uniform deployment might be useful also for a kind of the load balancing. That is, considering agents with large-size database replicas, uniform deployment guarantees that not all nodes need to store the database but each node can quickly access the database [7, 8]. Hence, we can see the uniform deployment problem as a kind of the resource allocation problem.

1.2. Related works

There are several researches considering the uniform deployment problem in the *Look-Compute-Move* model. In this model, agents are assumed to be oblivious (or memoryless) but be able to observe multiple agents (and nodes in graph environments) within its visibility range. In the look phase, an agent takes a snapshot and gets the positions of all agents (and nodes in graph environments) within the visibility range. In the compute phase, based on the snapshot, the agent decides where to go in the next movement. In the move phase, the agent moves to the destination. Agents repeat such cycles until the given task is completed. In the Look-Compute-Move model, Flocchini et al. [9] considered the uniform deployment problem in cycle environments of length m (m is a real number). They considered two types of uniform deployment: *exact* and *ϵ-approximate*. In the exact uniform deployment, agents move in the ring so that the distance between any two consecutive agents is the same, say d. In the ϵ-approximate uniform deployment, agents move in the cycle so that the distance should be between $d-\epsilon$ and $d+\epsilon$. They showed that if agents do not have common sense of direction, agents cannot solve the exact uniform deployment problem even if agents have unlimited memory and visibility range. If agents have common sense of direction, they proposed an algorithm to solve the exact uniform deployment problem for agents with knowledge of d. In addition, for any $\epsilon > 0$ they proposed an algorithm to solve the ϵ-approximate uniform deployment problem for agents without knowledge of d.

Elor et al. [10] considered uniform deployment in ring networks. They considered agents without knowledge k or n, where k is the number of agents and n is the number of nodes, but with visibility range VR. They considered

a semi-synchronous model, that is, a subset of agents execute a behavior in each round. They showed that, if $VR < \lfloor n/k \rfloor$ holds, agents cannot solve the uniform deployment problem. If $VR \geq \lfloor n/k \rfloor$ holds, they proposed an algorithm to solve the balanced uniform deployment problem without quiescence. That is, agents eventually satisfy the condition of uniform deployment and continue to move in the ring satisfying the condition. In addition, they proposed an algorithm to solve the semi-balanced uniform deployment problem with quiescence. That is, agents eventually terminate the algorithm satisfying the condition such that the distance between any two adjacent agents is between $n/k - k/2$ and $n/k + k/2$.

While [9] and [10] considered uniform deployment in ring networks, Barriere et al. [11] considered uniform deployment in grid networks and proposed an algorithm to achieve uniform deployment in $O(n/d)$ time, where d is the interval of uniform deployment.

1.3. Our contributions

In this paper, we focus on uniform deployment on asynchronous unidirectional rings. Although ring networks might seem so restricted in practice, it is known that the idea for ring networks is fundamental one and can be applied to other networks by embedding a ring in the network [12, 13]. Different from [9, 10, 11], we consider agents that have memory but cannot observe nodes except for the currently located node. To the best of our knowledge, this is the first research considering uniform deployment for such agents. In addition to the fact that uniform deployment is useful from a practical point of view as mentioned before, it is interesting to investigate also from a theoretical point of view. The problem exhibits a striking contrast to the *rendezvous problem*. The rendezvous problem, one of the most investigated problem, requires all agents to meet at a single node [14], and by doing this agents can share information or synchronize behaviors among them [15, 16, 17, 18, 19]. While rendezvous aims to break the symmetry and requires all the agents to meet at the single node, uniform deployment aims to attain the symmetry of agent locations and requires agents to spread uniformly. It is well known that the symmetry breaking is difficult (and sometimes impossible) in distributed systems, and the rendezvous problem cannot be solved from some initial configurations. Hence, it is interesting to clarify what difference the uniform deployment problem has on the solvability and the number of agent moves compared to the rendezvous problem.

Table 1: Results in each model

	Result 1 (Section 3.1)	Result 2 (Section 3.2)	Result 3 (Section 4.1)	Result 4 (Section 4.2)
Knowledge of k	Available	Available	Not Available	Not Available
Termination detection	Required	Required	Required	Not Required
Solvable / Unsolvable	Solvable	Solvable	Not Solvable	Solvable
Agent memory	$O(k \log n)$	$O(\log n)$	-	$O((k/l) \log(n/l))$
Time complexity	$O(n)$	$O(n \log k)$	-	$O(n/l)$
Total agent moves	$O(kn)$	$O(kn)$	-	$O(kn/l)$

n: number of nodes, k: number of agents, l: symmetry degree of the initial configuration

Contributions of this paper are summarized in Table 1. We assume that each agent initially has a token and can release it on a node that it is visiting. After a token is released at some node, agents cannot remove the token. In addition, we assume that agents can send a message of any size to agents staying at the same node. We consider two problem settings. First, we consider agents with knowledge of k (or n since k and n can be easily obtained if one of them is given). In this case, we propose two algorithms. The first algorithm solves the uniform deployment problem with termination detection. This algorithm requires $O(k \log n)$ memory space per agent, $O(n)$ time, and $O(kn)$ total moves. The second algorithm also solves the uniform deployment problem with termination detection. This algorithm reduces the memory space per agent to $O(\log n)$, but uses $O(n \log k)$ time, and requires $O(kn)$ total moves. Note that, from some initial configurations agents require $\Omega(kn)$ total moves to solve the problem. Hence, both algorithms are asymptotically optimal in terms of total moves.

Next, we consider agents with no knowledge of k or n. In this case, we show that, when termination detection is required, there exists no algorithm to solve the uniform deployment problem. Intuitively, it is due to impossibility of finding k or n when the initial configuration has sufficient number of repetitions of an agent location pattern: when an agent misestimates these at smaller numbers than actual ones, it prematurely terminates and uniform deployment cannot be achieved.

For this reason, we consider the relaxed uniform deployment problem that does not require termination detection, and we propose an algorithm to solve the relaxed uniform deployment problem. In this algorithm, each agent

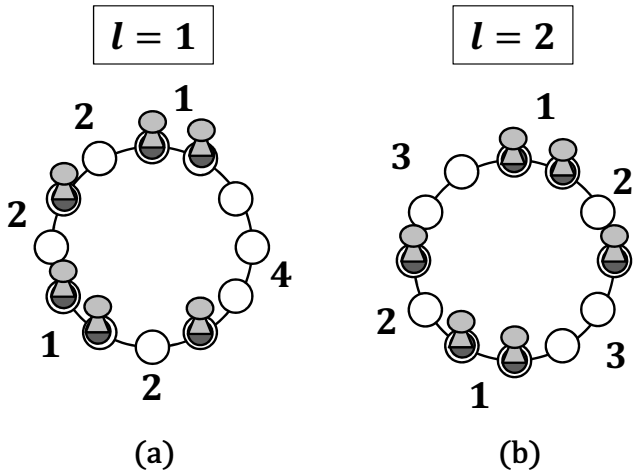

Figure 1: An example of the symmetry degree

estimates k and n (possibly at smaller values than actual ones) and behaves based on the estimation. Thus, the efficiency of the algorithm depends on the estimation. To evaluate the efficiency, we introduce the following parameter l to denote by the symmetry degree of an initial configuration: we say that an initial configuration has symmetry degree l when its distance sequence can be represented as l-times repetition of some aperiodic sequence. For example, the initial configuration in Fig. 1 (a) has symmetry degree 1 since its whole distance sequence (1,4,2,1,2,2) is aperiodic, and the initial configuration in Fig. 1 (b) has symmetry degree 2 since its whole distance sequence (1,2,3,1,2,3) is represented as 2-times repetition of aperiodic sequence (1,2,3). Hence, the symmetry degree becomes larger for a more highly symmetric initial configuration. Note that if the initial configuration is already uniform, $l = k$ holds, and hence $1 \leq l \leq k$ holds. Agents cannot know l but the efficiency depends on it. Using the symmetry degree parameter l, the efficiency of the algorithm is denoted as follows: this algorithm requires $O((k/l)\log(n/l))$ memory space per agent, $O(n/l)$ time, and $O(kn/l)$ total moves. At first glance, the upper bound $O(kn/l)$ of the total moves seem to violate the lower bound $\Omega(kn)$ of the total moves. However, the lower bound is for the worst case of initial configurations. Initial configuration with $l \geq 2$ are closer to the uniformly-deployed configuration and agents require less than $\Omega(kn)$ total moves to solve the problem. Hence, from such

initial configurations agents can adaptively solve the problem in less than $\Omega(kn)$ total moves. Thus, this algorithm achieves uniform deployment more efficiently when the initial configuration has a higher symmetry degree. This is a natural but interesting property. For example, for an asymmetric initial configuration this algorithm requires $O(k \log n)$ memory space per agent, $O(n)$ time, and $O(kn)$ total moves. However, when l is $\omega(1)$, this algorithm requires $o(k \log n)$ memory space per agent, $o(n)$ time, and $o(kn)$ total moves. When l is $\Omega(k)$, this algorithm requires $O(\log(n/k))$ memory space per agent, $O(n/k)$ time, and $O(n)$ total moves.

Note that, from any initial configuration such that all agents are in the initial state and placed at the distinct nodes, all proposed algorithms achieve uniform deployment, which is a striking difference from the rendezvous problem because the rendezvous problem is not solvable from some initial symmetric configurations.

1.4. Organization

The paper is organized as follows. Section 2 presents the system model and the problem to be solved. In Section 3 we consider agents with knowledge of k. In Section 4 we consider agents with no knowledge of k or n. Section 5 concludes the paper.

2. Preliminaries

2.1. System model

A *unidirectional ring network* R is defined as 2-tuple $R = (V, E)$, where V is a set of anonymous nodes and E is a set of unidirectional links. We denote by $n\,(= |V|)$ the number of nodes. Then, we define $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E = \{e_0, e_1, \ldots, e_{n-1}\}\,(e_i = (v_i, v_{(i+1) \bmod n}))$. For simplicity, operations on an index of a node assume calculations modulo n, that is, $v_{(i+1) \bmod n}$ is simply represented by v_{i+1}. We define the direction from v_i to v_{i+1} as the *forward* direction. In addition, we define the j-th forward agent a' of agent a as the agent such that $j-1$ agents exist between a and a' in the forward direction of a. For convenience, we define the 0-th forward agent of a as a itself. Moreover, the *distance* from node v_i to $v_j\,(0 \leq i, j \leq n-1)$ is defined to be $(j-i) \bmod n$.

An agent is a state machine having an *initial state*. Let $A = \{a_0, a_1, \ldots, a_{k-1}\}$ be a set of $k\,(\leq n)$ anonymous agents. For simplicity, operations on an index of an agent assume calculations modulo k. Since the ring is

unidirectional, agents staying at v_i can move only to v_{i+1}. We consider two problem settings: agents with knowledge of k (or n since k and n can be easily obtained if one of them is given) and agents with no knowledge of k or n. We assume that each agent initially has a *token* and can release it on a node that it is visiting. The token on an agent or a node can be realized by one bit memory that denotes existence of the token, and thus, the token cannot carry any additional information. Note that if agents are not allowed to have tokens, they cannot mark nodes in any way and this means that uniform deployment problem cannot be solved. This is because if all agents move in a synchronous manner, they cannot get any information of other agents. After a token is released at some node, agents cannot remove the token. Note that since agents are anonymous, they cannot recognize the owner of the token. In addition, we assume that agents can send a message of any size to agents staying at the same node. We assume that agents move through a link in a FIFO manner, that is, when agent a_p leaves v_i after agent a_q, a_p reaches v_{i+1} after a_q. Note that such a FIFO assumption is natural because 1) agents are implemented as messages in practice, and 2) the FIFO assumption of messages are natural and can be easily realized in distributed systems.

Each agent a_i executes the following five operations in an atomic action: 1) The agent reaches a node v (when a_i is in transit toward v), or it starts operations at v (when a_i is at v), 2) the agent receives all the messages (if any), 3) the agent executes local computation, 4) the agent broadcasts a message to all the agents staying at the same node v (if any) if it decides to send a message, and 5) the agent leaves v if it decides to move. After taking an atomic action, a_i has no message. Note that these assumptions of atomic actions are also natural because they can be implemented locally at a node if each node has an incoming *buffer* that stores agents about to visit the node and makes them execute actions in a FIFO order. We consider an *asynchronous* system, that is, the time for each agent to transit to the next node and to wait until execution of the next action (when staying at a node) is finite but unbounded.

A (global) *configuration* C is defined as a 5-tuple $C = (S, T, M, P, Q)$ and the correspondence table is given in Table 2. The first element S is a k-tuple $S = (s_0, s_1, \ldots, s_{k-1})$, where s_i is the state (including the state to denote whether it holds a token or not) of agent $a_i\,(0 \leq i \leq k-1)$. The second element T is an n-tuple $T = (t_0, t_1, \ldots, t_{n-1})$, where t_i is the state (i.e., the number of tokens) of node $v_i\,(0 \leq i \leq n-1)$. The third element M is a

Table 2: Meaning of each element in configuration $C = (S, T, M, P, Q)$

Element	Meaning and example
$S = (s_0, s_1, \ldots, s_{k-1})$	Set of agent states (s_i: the state of agent a_i)
$T = (t_0, t_1, \ldots, t_{n-1})$	Set of node states (t_i: the state of node v_i)
$M = (m_0, m_1, \ldots, m_{k-1})$	Set of message sequences (m_i: a sequence of massages sent to a_i and not received by a_i)
$P = (p_0, p_1, \ldots, p_{n-1})$	Set of agents staying at nodes (p_i: a set of agents staying at node v_i)
$Q = (q_0, q_1, \ldots, q_{n-1})$	Set of agent sequences residing on links (q_i: a sequence of agents in transit from v_{i-1} to v_i)

k-tuple $M = (m_0, m_1, \ldots m_{k-1})$, where m_i is a sequence of messages reached a_i but not consumed yet by a_i. The remaining elements P and Q represent the positions of agents. The element P is an n-tuple $P = (p_0, p_1, \ldots, p_{n-1})$, where p_i is a set of agents staying at node v_i $(0 \leq i \leq n-1)$. The element Q is an n-tuple $Q = (q_0, q_1, \ldots, q_{n-1})$, where q_i is a sequence of agents residing in the FIFO queue corresponding to link (v_{i-1}, v_i) $(0 \leq i \leq n-1)$. Hence, agents in q_i are those in transit from v_{i-1} to v_i.

We denote by $\mathcal{C}$ the set of all possible configurations. In *initial configuration* $C_0 \in \mathcal{C}$, all agents are in the initial state (where each has a token) and placed at distinct nodes, and no node has any token. In addition, in C_0 the node where agent a is located is called the *home node* of a and denoted by $v_{HOME}(a)$. We assume that in C_0 agent a is stored at the incoming buffer of its home node $v_{HOME}(a)$. This assures that agent a starts the algorithm at $v_{HOME}(a)$ before any other agent visits $v_{HOME}(a)$, that is, a is the first agent that takes an action at $v_{HOME}(a)$.

In addition, we define *periodic rings* and the symmetry degree l. For 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, in C_0 we define the *distance sequence* of agent a_i as $D_i(C_0) = (d_0^i(C_0), \ldots, d_{k-1}^i(C_0))$, where $d_j^i(C_0)$ is the distance from the j-th forward agent of a_i to the $(j+1)$-th forward agent of a_i in C_0. In addition, we define the distance sequence of configuration C_0 as the lexicographically minimum sequence among $\{D_i(C_0) | a_i \in A\}$, and we denote it by $D(C_0)$. 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, when $D(C_0) = \mathit{shift}(D(C_0), x)$ holds for some x $(0 < x < k)$, we say the ring

is *periodic*. For a periodic ring, letting x be the minimum integer satisfying $D(C_0) = \mathit{shift}(D(C_0), x)$, the symmetric degree l of C_0 is defined to be $l = k/x$. Otherwise, we say the ring is *aperiodic* and we define $l = 1$.

A *schedule* is an infinite sequence of agents. A schedule $X = \rho_1, \rho_2, \ldots$ is fair if every agent appears in X infinitely often. An infinite sequence of configurations $E = C_0, C_1, \ldots$ is called an *execution* from C_0 if there exists a fair schedule $X = \rho_1, \rho_2, \ldots$ that satisfies the following conditions for each h $(h > 0)$:

- If $\rho_{h-1} \in p_i$ holds for some i in configuration C_{h-1}, the states of ρ_{h-1} and v_i in C_{h-1} are changed to those in C_h by local computation of ρ_{h-1}. Let $a_j = \rho_{h-1}$. If $m_j \neq \emptyset$, all messages in m_j are delivered to a_j and consumed, that is, m_j becomes $\emptyset$. In addition, if ρ_{h-1} sends a message, the message is appended to the tail of m_l for each agent a_l staying at v_i. Moreover, if ρ_{h-1} releases its token at v_i, the value of t_i increases by one. After this, if ρ_{h-1} decides to move to v_{i+1}, ρ_{h-1} is removed from p_i and is appended to the tail of sequence q_{i+1}. If ρ_{h-1} decides to stay, ρ_{h-1} remains in p_i. The other elements in C_h are the same as those in C_{h-1}.

- If ρ_{h-1} is at the head of q_i for some i in configuration C_{h-1}, ρ_{h-1} moves to v_i, that is, ρ_{h-1} is removed from q_i. Then, the states of ρ_{h-1} and v_i in C_{h-1} are changed to those in C_h by local computation of ρ_{h-1}. If ρ_{h-1} sends a message, the message is appended to the tail of m_l for each agent a_l staying at v_i. In addition, if ρ_{h-1} releases its token at v_i, the value of t_i increases by one. After this, if ρ_{h-1} decides to move to v_{i+1}, ρ_{h-1} is appended to the tail of sequence q_{i+1}. If ρ_{h-1} decides to stay, ρ_{h-1} is inserted in p_i. The other elements in C_h are the same as those in C_{h-1}.

2.2. The uniform deployment problem

The uniform deployment problem in a ring network requires $k\,(\geq 2)$ agents to spread uniformly in the ring, that is, all the agents are located at distinct nodes and the distance between any two *adjacent agents* should become identical like Fig. 2. Here, we say two agents are adjacent when there exists no agent between them. However, we should consider the case that n is not a multiple of k. In this case, we aim to distribute the agents so that the distance d of any two adjacent agents should be $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$.

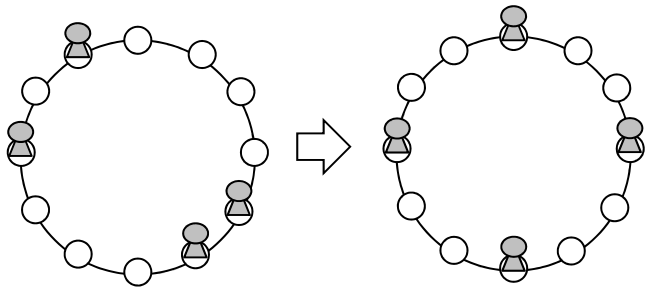

Figure 2: An example of uniform deployment ($n = 16, k = 4, d = 3$)

We consider two kinds of the uniform deployment problem: *with termination detection* and *without termination detection*. At first, we define the uniform deployment problem with termination detection. In this case, a unique *halt state* is defined as follows: when agent a_i enters the halt state, it terminates the algorithm, that is, a_i neither changes its state nor leaves the current node even if another agent sends a message to a_i. Hence if an agent enters the halt state, it can detect its termination. Now, we define the uniform deployment problem with termination detection as follows.

Definition 1. *An algorithm solves the uniform deployment problem* with *termination detection if any execution satisfies the following conditions.*

- *All agents change their states to the halt state in finite time.*
- *When all agents are in the halt state, $q_j = \emptyset$ holds for any $q_j \in Q$ and the distance of each pair of adjacent agents is $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$.* □

Next, we define the uniform deployment problem without termination detection. In this case, *suspended states* are defined as follows: when agent a_i enters a suspended state, it neither changes its state nor leaves the current node unless another agent sends a message to a_i. If a_i receives a message, it can resume its behavior and leave the current node. Different from the halt state, the suspended states are not uniquely defined since an agent can resume its behavior from a suspended state; the suspended state should contain the information necessary to resume the behavior. The uniform deployment problem without termination detection allows all agents to stop in the suspended states, which is also known as communication deadlock.

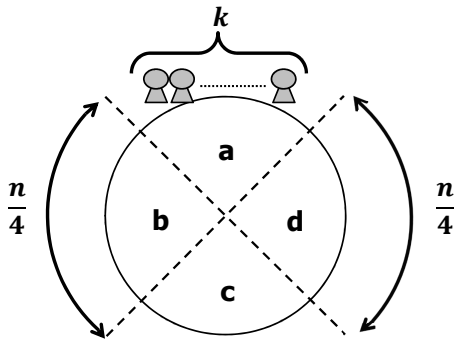

Figure 3: The initial configuration to derive a lower bound $\Omega(kn)$ of the total moves

Definition 2. *An algorithm solves the uniform deployment problem* without *termination detection if any execution satisfies the following conditions.*

- *All agents change their states to the suspended states in finite time.*
- *When all agents are in the suspended states, $m_i = \emptyset$ holds for each agent a_i, $q_j = \emptyset$ holds for any $q_j \in Q$, and the distance of each pair of adjacent agents satisfies $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$.* □

For the uniform deployment problem, we have the following lower bound of total moves. This lower bound holds even if agents have knowledge of k.

Theorem 1. *When $k \leq pn$ holds for some constant $p\,(p < 1)$, a lower bound of the total moves to solve the uniform deployment problem (with or without termination detection) is $\Omega(kn)$ even if agents have knowledge of k.*

Proof. We assume for simplicity that $k \leq n/4$ holds and consider the initial configuration such that all agents stay in a quarter part of the ring like Fig. 3. In such an initial configuration, $l = 1$ holds (i.e. the ring is aperiodic). Then, the ring is divided into four quarter parts, and in the initial configuration, all agents are in the part a. To achieve uniform deployment, $k/4$ agents need to move to the part c, the opposite part of a, and each of them must move at least $n/4$ times. Thus, the total number of moves is at least $(k/4) \times (n/4) = kn/16$. This argument can be easily extended to any constant $p\,(p < 1)$ satisfying $k \leq pn$. □

Next, we define the *time complexity* as the time required to achieve uniform deployment. Since there is no assumption on time in asynchronous systems, it is impossible to measure the exact time. Instead, we consider the *ideal time complexity*, which is defined as the execution time under the following assumptions: 1) The time required for an agent to move from a node to its neighboring node or to wait until execution of the next action is at most one, and 2) the time required for local computation is ignored (i.e., zero)[1]. For example, if some agent continues to move in the ring from the beginning to the end of execution of the algorithm, the ideal time complexity is equivalent to the number of moves for the agent. These assumptions are introduced only to evaluate the time complexity, that is, algorithms are required to work correctly without such assumptions. In the following, we simply use terms "time complexity" and "time" instead of "ideal time complexity". Then, we can show the following theorem similarly to Theorem 1.

Theorem 2. *A lower bound of the time complexity to solve the uniform deployment problem (with or without termination detection) is* $\Omega(n)$. □

3. Agents with knowledge of k

In this section, we consider the uniform deployment problem for agents with knowledge of k.[2] We propose two algorithms to solve the uniform deployment problem with termination detection. The first algorithm is native one and requires $O(k \log n)$ memory space per agent, $O(n)$ time, and $O(kn)$ total moves. The second algorithm reduces the memory space per agent to $O(\log n)$, but uses $O(n \log k)$ time, and requires $O(kn)$ total moves.

3.1. A native algorithm with $O(k \log n)$ agent memory

In this section, we propose an algorithm to solve the uniform deployment problem with termination detection which requires $O(k \log n)$ memory space per agent, $O(n)$ time, $O(kn)$ total moves. For simplicity, we assume $n = ck$ for some positive integer c, and we can remove this assumption in Section

[1] This definition is based on the ideal time complexity for asynchronous message-passing systems [20].

[2] We assume agents with knowledge of k, but agents with knowledge of n can similarly solve the problem.

3.1.1. The algorithm consists of the following two phases: the selection phase and the deployment phase. In the selection phase, each agent travels once around the ring and selects a *base node* as a reference node of uniform deployment. In the deployment phase, based on the base node, each agent determines a *target node* where it should stay and moves to the node.

In the selection phase, each agent a_i firstly releases its token at its home node $v_{HOME}(a_i)$, and travels once around the ring. Note that since each agent has knowledge of k, it can detect when it completes one circuit of the ring (or when it returns to its home node). During the traversal, a_i measures the distance *dis* between every pair of adjacent token nodes, and stores *dis* to an array D for memorizing the distance sequence. When completing one circuit of the ring, a_i gets the value of n and the distance sequence $D = (d_0, d_1, \ldots, d_{k-1})$, where d_j is the distance from the j-th token node it found to the $(j+1)$-th token node. Note that a_i's home node $v_{HOME}(a_i)$ is considered as the 0-th token node. Let x be the minimum number such that $\mathit{shift}(D, x) = D_{min}$ holds, where D_{min} is the lexicographically minimum distance sequence among $\{\mathit{shift}(D, x) | 0 \leq x \leq k-1\}$. Then, a_i selects its base node $v_{base}(a_i)$ as the home node of the x-th agent of a_i. Note that the x-th agent has the minimum distance sequence D_{min}. If D is aperiodic, all the agents select the same node as a base node. If D is periodic, multiple nodes are selected as base nodes (Fig. 4 (a)).

In the deployment phase, each agent a_i determines its target node and moves to the node. First, a_i considers that it is the *rank*-th agent ($0 \leq \mathit{rank} \leq k' - 1$) to $v_{base}(a_i)$. Here, if a_i is the *rank*-th agent, it means that there exist $\mathit{rank}-1$ agents (or tokens) between $v_{HOME}(a_i)$ and $v_{base}(a_i)$. Thus, *rank* is equal to x in the previous paragraph. Note that agent a_i staying at $v_{base}(a_i)$ is considered as the 0-th agent. In Fig. 4 (a), agent a_2 (resp., a_5) is the 1-st agent since there exists no agent between a_2 (resp., a_5) and $v_{base}(a_2)$ (resp., $v_{base}(a_5)$). Similarly, a_1 and a_4 are the 2-nd agents. Let *disBase* be the distance from its home node $v_{HOME}(a_i)$ to $v_{base}(a_i)$. Note that the value of *disBase* for the 0-th agent is 0. First, a_i moves *disBase* times and reaches $v_{base}(a_i)$. After this, a_i moves to its target node by moving $\mathit{rank} \times n/k$ times and terminates the algorithm. In Fig. 4 (b), the target nodes of a_0, a_1, a_2, a_3, a_4, and a_5 are $v_t^0, v_t^1, v_t^2, v_t^3, v_t^4$, and v_t^5, respectively.

Note that if multiple base nodes are selected like Fig. 4, the following properties are satisfied: 1) The distance between every pair of two adjacent base nodes is identical, and 2) the number of agents and their locations between every pair of adjacent base nodes are also identical. Thus, the base

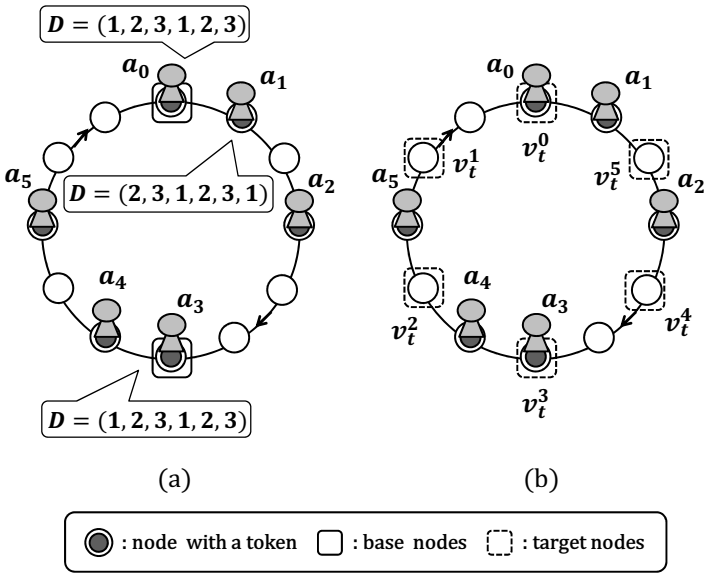

Figure 4: The base nodes and the target nodes

nodes can be reference nodes of uniform deployment, and each agent can determine its base node and target node uniquely.

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

Theorem 3. *For agents with knowledge of k, Algorithm 1 solves the uniform deployment problem with termination detection. This algorithm requires $O(k \log n)$ memory space per agent, $O(n)$ time, and $O(kn)$ total moves.*

Proof. It is obvious that Algorithm 1 solves the uniform deployment problem with termination detection, and in the following we analyze the complexity measures.

At first, we evaluate the memory requirement per agent. Each agent eventually gets the distance sequence $D = (d_0, d_1, \ldots, d_{k-1})$. Since each d_i is at most n, this sequence requires $O(k \log n)$ memory. Moreover, the other variables require $O(\log n)$ bit memory. Therefore, the memory requirement per agent is $O(k \log n)$.

Next, we analyze the time complexity and the total moves. In the selection phase, each agent travels once around the ring to get D, which takes n

Algorithm 1 A time optimal algorithm for agents with knowledge of k

Main behavior of Agent a_i

1: /* selection phase */
2: $j = 0$
3: release a token at its home node $v_{HOME}(a_i)$
4: **while** $j \neq k$ **do**
5: move to the nearest token node and get the distance *dis* between two token nodes
6: $D[j] = dis$
7: $j = j + 1$
8: **end while**
9: // a_i completes travelling once around the ring and gets the number of nodes
10: $n = D[0] + D[1] + \cdots + D[k-1]$
11:
12: /* deployment phase */
13: let D_{min} be the lexicographically minimum sequence among $\{shift(D, x) | 0 \leq x \leq k-1\}$
14: $rank = \min\{x \geq 0 | shift(D, x) = D_{min}\}$
15: **if** $rank = 0$ **then** $disBase = 0$
16: **if** $rank \neq 0$ **then** $disBase = D[0] + D[1] + \cdots + D[rank-1]$
17: move $disBase + rank \times n/k$ times
18: terminate the algorithm

time units and n moves. In the deployment phase, each agent moves to its own target node, which takes at most $2n$ time units and $2n$ moves. Thus, the time complexity is $O(n)$ and the total number of moves is $O(kn)$. □

3.1.1. The uniform deployment for the case of $n \neq ck$

To remove the restriction of $n = ck$ imposed in Section 3.1, only the parts for determining the target node and for moving to the target node should be modified. In the case that n is not a multiple of k, the distance between adjacent target nodes should be $\lceil n/k \rceil$ or $\lfloor n/k \rfloor$.

Since all the agents recognize a single base node or uniformly distributed base nodes, they can determine the uniformly distributed target nodes using the base nodes as reference nodes: Let b be the number of the base nodes, and $r = n \bmod k$. The distance of every pair of adjacent base nodes is identical

even in the case of $n \neq ck$, and is $n/b = (\lfloor n/k \rfloor \times k + r)/b = \lfloor n/k \rfloor \times k/b + r/b$ (notice that k/b and r/b are integers). This implies that we should select $k/b - 1$ target nodes between two adjacent base nodes so that the first r/b intervals between adjacent target nodes should be $\lceil n/k \rceil$ and others should be $\lfloor n/k \rfloor$. With considering the above, each agent can determine its own target node by local computation so that all the agents can spread over the ring to achieve uniform deployment.

3.2. An algorithm with $O(\log n)$ agent memory

In this section, we propose an algorithm to solve the uniform deployment problem with termination detection which reduces the memory space per agent to $O(\log n)$, but uses $O(n \log k)$ time, and requires $O(kn)$ total moves. The algorithm consists of two phases similarly to Algorithm 1 in Section 3.1: the selection phase and the deployment phase. The selection phase of Algorithm 1 requires $O(n \log k)$ space per agent, so we modify the selection phase to reduce the space. The deployment phase is also slightly modified so that it can adapt to the modified selection phase. For simplicity we assume $n = ck$ for some positive integer c in the following description, and this assumption is removed similarly to Section 3.1.1.

3.2.1. Selection phase

As in the selection phase of Algorithm 1, some of home nodes are selected as the base nodes, and they are used as reference nodes for uniform deployment. Algorithm 1 achieves this by making each agent get the whole distance sequence, which requires $O(k \log n)$ memory space, and determine the base node independently. To reduce the memory space per agent, the criterion of the base nodes is relaxed and agents cooperatively select the base nodes.

The base nodes are selected to satisfy the following three conditions called the **base node conditions**: 1) There exists at least one base node, 2) the distance between every pair of adjacent base nodes is the same, and 3) the number of home nodes between every pair of adjacent base nodes is the same. Compared to Algorithm 1, the last condition is relaxed in the sense that locations of the home nodes are not considered. However, this condition still guarantees that the number of the selected base nodes is a divisor of k. For example, let us consider the initial locations of agents of Fig. 5. Then, distances from $v_{HOME}(a_1)$ to $v_{HOME}(a_2)$, from $v_{HOME}(a_2)$ to $v_{HOME}(a_3)$, and from $v_{HOME}(a_3)$ to $v_{HOME}(a_1)$ are all 6, and the number of home nodes between $v_{HOME}(a_1)$ and $v_{HOME}(a_2)$, between $v_{HOME}(a_2)$ and $v_{HOME}(a_3)$, and

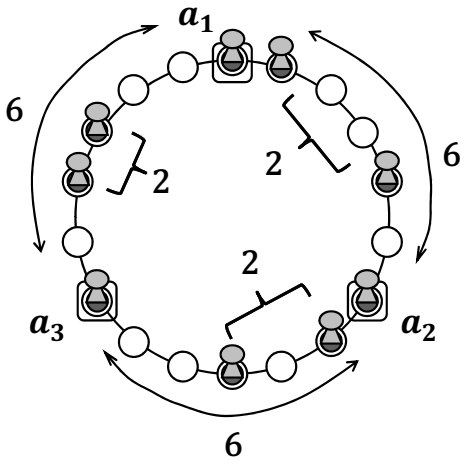

Figure 5: An example of the base node conditions ($n = 18, k = 9, d = 2$)

between $v_{HOME}(a_3)$ and $v_{HOME}(a_1)$ are all 2. Thus, $v_{HOME}(a_1)$, $v_{HOME}(a_2)$, and $v_{HOME}(a_3)$ satisfy the base node conditions. Agents select such base nodes with $O(\log n)$ memory. When the selection phase is completed, each agent stays at its home node and knows whether its home node is selected as a base node or not. We call an agent a *leader* (but probably not unique) when its home node is selected as a base node, and call it a *follower* otherwise. The state of an agent is *active*, *leader* or *follower*. Active agents are candidates for leaders, and initially all agents are active. Once an agent becomes a follower or a leader, it never changes its state. In the following, we say that a node v is active (resp., a follower) when v is the home node of an active (resp., a follower) agent.

Now, we explain the outline of the selection phase. In this phase, agents get *IDs* from the distance between token nodes and decrease the number of active agents using the IDs. We explain the detail of the IDs later. At the beginning of the algorithm, each agent a_i releases its token at its home node $v_{HOME}(a_i)$. The selection phase consists of at most $\lceil \log k \rceil$ sub-phases. At the beginning of each sub-phase, each agent stays at its own home node. During the sub-phase, if agent a_i is a follower, it keeps staying at its home node. On the other hand, each active agent a_i travels once around the ring and gets its

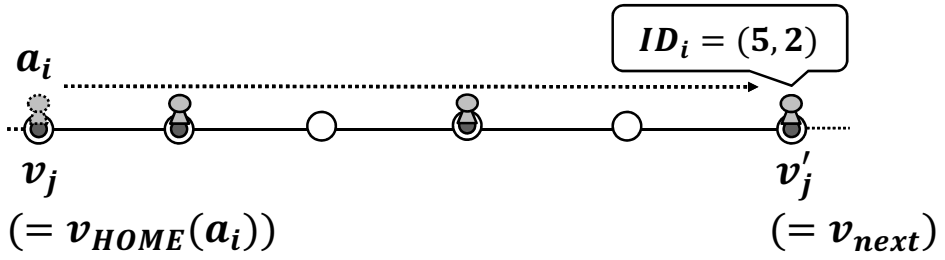

Figure 6: An ID of an active agent a_i

own ID by the method described later.[3] Then, a_i compares its own ID with IDs of other agents and determines the next behavior. Briefly speaking, if all active agents have the same IDs, it means that all the distances between neighboring active nodes are the same, and home nodes of the active agents satisfy the base node conditions. Hence, the active agents become leaders and enter to the deployment phase. If all agents do not have the same IDs but a_i's ID is the minimum, it remains active and executes the next sub-phase. If a_i does not satisfy any of the above conditions, it becomes a follower. Each active agent executes such sub-phases at most $\lceil \log k \rceil$ times.

Now, we explain the detail of the ID. The ID (not necessarily unique) of an active agent a_i is given in the form of $(d_i, fNum_i)$, where d_i is the distance from its home node $v_{HOME}(a_i)$ to the next active node, say v_{next}, in the sub-phase and $fNum_i$ is the number of follower nodes between $v_{HOME}(a_i)$ and v_{next}. For example, in Fig. 6 when agent a_i moves from its home node v_j to the next active node v'_j, it visits five nodes and observes two follower nodes. Hence, a_i gets its own ID $ID_i = (5, 2)$. We compare two IDs by the lexicographical order: for $ID_1 = (d_1, fNum_1)$ and $ID_2 = (d_2, fNum_2)$, $ID_1 < ID_2$ if $(d_1 < d_2) \vee ((d_1 = d_2) \wedge (fNum_1 < fNum_2))$ holds. Each active agent decides whether it remains active or not using such IDs. Notice that an agent gets different IDs in different sub-phases since at least one of the neighboring active agents becomes a follower in every sub-phase.

In the following, we explain the implementation of the sub-phase. In the sub-phase, each active agent a_i travels once around the ring. While traveling,

[3]Each agent can detect when it completes one circuit of the ring since it has knowledge of k.

a_i executes the following actions:

1. Get its own ID $ID_i = (d_i, fNum_i)$:
   Agent a_i gets its own ID ID_i by moving from its home node $v_{HOME}(a_i)$ to the next active node v_{next} with counting the numbers of nodes and follower nodes (Fig. 6). Since all active agents are traversing the ring and all follower agents are staying at their home nodes, a_i can detect its arrival at the next active node when it visits a node with a token but with no agent. Note that this statement holds even in asynchronous systems because active agents do not pass other active agents from the FIFO property of links and the atomicity of execution.

2. Get the ID $ID_{next} = (d_{next}, fNum_{next})$ of its next active agent:
   Similarly, with counting the numbers of nodes and follower nodes, a_i moves from v_{next} to the next active node (i.e., the node with a token but with no agent). Then, a_i gets the ID of a_i's next active agent and stores it to ID_{next}.

3. Compare ID_i with those of all active agents:
   During the traversal of the ring, a_i compares ID_i with IDs of all active agents one by one, and checks 1) whether ID_i is the minimum and 2) whether the IDs of all active agents are the same. To check these, agent a_i keeps boolean variables *min* ($min = true$ means ID_i is the minimum among ever-found IDs) and *identical* ($identical = true$ means that ever-found IDs are the same), and it updates the variables (if necessary) every time it finds an ID of another active agent.

When a_i completes one circuit of the ring, it determines its state for the next sub-phase. If $identical = true$ holds, this means that all active agents have the same IDs. In this case, a_i (and all the other active agents) becomes a leader and completes the selection phase. If $identical = false$, $min = true$ and $ID_i < ID_{next}$ hold, a_i remains active and executes the next sub-phase. The last condition means that, when active agents with the minimum ID appear consecutively, only one of them (or the last agent in the consecutive agents) remains active. This guarantees that the number of active agents is at least halved in each sub-phase. If a_i does not satisfy any of the above conditions, it becomes a follower. By repeating such sub-phase at most $\lceil \log k \rceil$ times, all the remaining active agents have the same IDs in some sub-phase and they

Algorithm 2 The behavior of active agent a_i (selection phase)

Behavior of Agent a_i

1: /*selection phase*/
2: $phase = 1$, $identical = true$, $min = true$
3: release a token at its home node $v_{HOME}(a_i)$
4: **while** $phase \neq \lceil \log k \rceil$ **do**
5: move to the next active node and get its own ID $ID_i = (d_i, fNum_i)$
6: **if** a_i is at $v_{HOME}(a_i)$ **then** terminate the selection phase and start the deployment phase with a leader state // only a_i is active
7: move to the next active node and get ID $ID_{next} = (d_{next}, fNum_{next})$ of the next active agent
8: **if** $ID_i \neq ID_{next}$ **then** $identical = false$
9: **if** $ID_i > ID_{next}$ **then** $min = false$ // there exists an agent having a smaller ID
10: **while** a_i is not at $v_{HOME}(a_i)$ **do**
11: move to the next active node and get ID $ID_{other} = (d_{other}, fNum_{other})$ of the next active agent
12: **if** $ID_i \neq ID_{other}$ **then** $identical = false$
13: **if** $ID_i > ID_{other}$ **then** $min = false$ // there exists an agent having a smaller ID
14: **end while**
15: **if** $identical = true$ **then** terminate the selection phase and start the deployment phase with a leader state // all active agents have the same IDs
16: **if** $(min = false) \vee (ID_i = ID_{next})$ **then** terminate the selection phase and start the deployment phase with a follower state
17: $phase = phase + 1$, $identical = true$, $min = true$
18: **end while**

are selected as leaders so that their home nodes (or the base nodes) should satisfy the base node conditions.

The pseudocode is described in Algorithm 2. Note that in the first subphase of Algorithm 2, each agent can get the number n of nodes when it finishes traveling once around the ring, but we omit the description.

3.2.2. Deployment phase

As in Algorithm 1, each agent determines its target node and moves to the node. From the base node conditions, the base nodes are first selected as the target nodes. Hence, letting b be the number of the base nodes, other $k - b$ target nodes are selected so that the distance between two adjacent target nodes should be n/k.

While the leaders know the completion of the selection phase, followers do not know the fact. Hence, at the beginning of the deployment phase, each leader notifies followers that the selection phase is completed. To do this, each leader moves to the next base node. During the movement, if there exists an agent, the leader informs the agent of the number of tokens (or home nodes) *tBase* to the next base node. If the leader arrives at the next base node, it terminates the algorithm there since the current base node is its target node.

When each follower receives the value of *tBase*, it knows the completion of the selection phase. Then, it starts the deployment phase. Each follower moves in the ring until it observes *tBase* tokens, and then it reaches the nearest base node. After this, the agent traverses the ring until it finds a vacant target node: every time the agent moves n/k times, it reaches a target node and stays there if the node is vacant (i.e., no agent is staying), otherwise (i.e., when the target node is already occupied by another agent) it keeps moving to the next target node by making another n/k moves. Note that from the atomicity of execution, it does not happen that two follower agents arrive at the same target node at the same time, that is, exactly one follower stays at each target node. The pseudocode is described in Algorithm 3. We have the following theorem about the presented algorithm.

Theorem 4. *For agents with knowledge of k, Algorithms 2 and 3 solve the uniform deployment problem with termination detection. This algorithm requires $O(\log n)$ memory space per agent, $O(n \log k)$ time, and $O(kn)$ total moves.*

Proof. It is obvious that Algorithms 2 and 3 solve the uniform deployment problem with termination detection even in periodic rings, and in the following we analyze the complexity measures.

At first, we evaluate the memory requirement per agent. Each agent a_i has three variables ID_i, ID_{next}, and ID_{other} to store IDs, each of which requires $O(\log n)$ memory. Since other variables require $O(\log n)$ or less memory, each agent requires $O(\log n)$ memory.

Algorithm 3 The behavior of leader or follower agent a_i (deployment phase)

Behavior of Agent a_i

```
1: /*deployment phase*/
2: // the behavior of leader agents
3: if a_i is in the leader state then
4:    t = 0
5:    while t ≠ fNum_i do
6:       move to the next node where a token exists  // look for a follower agent
7:       send tBase (= fNum_i − t) to the agent staying at the current node
8:       t = t + 1
9:    end while
10:   move to the next node where a token exists  // move to the next base node
11:   terminate the algorithm
12: end if
13:
14: // the behavior of follower agents
15: if a_i is in the follower state then
16:   wait at the current node until a_i receives the value of tBase
17:   move until it observes tBase tokens // a_i reaches the nearest base node
18:   while true do
19:      move n/k times // move to the next target node
20:      if there exists no agent staying at the current node then terminate the algorithm
21:   end while
22: end if
```

Next, we consider the time complexity. The selection phase requires at most $n\lceil \log k \rceil$ time units because each sub-phase requires n time units and agents execute at most $\lceil \log k \rceil$ sub-phases. In addition, the deployment phase requires at most $2n$ time units. Hence, the time complexity is $O(n \log k)$.

Lastly, we consider the total moves. First, we consider the selection phase. In each sub-phase, each active agent travels once around the ring, and then at least half active agents become followers or all active agents become leaders. Hence, in the beginning of the x-th sub-phase, the number of active agents is at most $k/2^{x-1}$. Since follower agents and leader agents never move in the

selection phase, the total number of moves in the selection phase is at most $\sum_{1 \leq x \leq \log k}(k/2^{x-1})n \leq 2kn$. In the deployment phase, each leader moves to the next base node and each follower moves to a target node to achieve uniform deployment. Each leader obviously moves at most n times, and each follower moves at most $2n$ times since it first moves to the nearest base node, which requires at most n moves, and then moves to a vacant target node, which requires at most n moves. Thus, the total moves in the deployment phase is $O(kn)$. Therefore, the total moves is $O(kn)$. □

4. Agents with no knowledge of k or n

In this section, we consider the uniform deployment problem for agents with no knowledge of k or n. We consider cases with termination detection and without termination detection in this order.

4.1. Uniform deployment problem with termination detection

When termination detection is required, we show that there exists no algorithm to solve the problem. Intuitively, it is due to impossibility of finding correct k or n when some part of the initial configuration has a repeated distance sequence: due to this some agent in this part misestimates k and n at smaller numbers than actual ones, it prematurely terminates and uniform deployment cannot be achieved.

Theorem 5. *There exists no algorithm to solve the uniform deployment problem with termination detection if agents have no knowledge of k or n.*

Proof. We use the similar idea to that in [16], which shows that for agents without any knowledge there exists no algorithm to solve the rendezvous problem with termination detection. For simplicity, we assume that agents move in a synchrnous manner, that is, in each round agents move instantaneously and there is no agent in transit from some node v_i to v_{i+1}. We prove the theorem by contradiction, that is, we assume that there exists algorithm $\mathcal{A}$ to solve the uniform deployment problem with termination detection.

At first, let us consider n-node ring R and the initial configuration C_0 such that k agents $a_0, a_1, \ldots, a_{k-1}$ exist in this order. Let $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and assume that $d = n/k$ is a positive integer. From the hypothesis, there is an execution E_R of $\mathcal{A}$ to solve the uniform deployment problem in R. We define $T(E_R)$ as the length (or the number of rounds) of E_R and denote

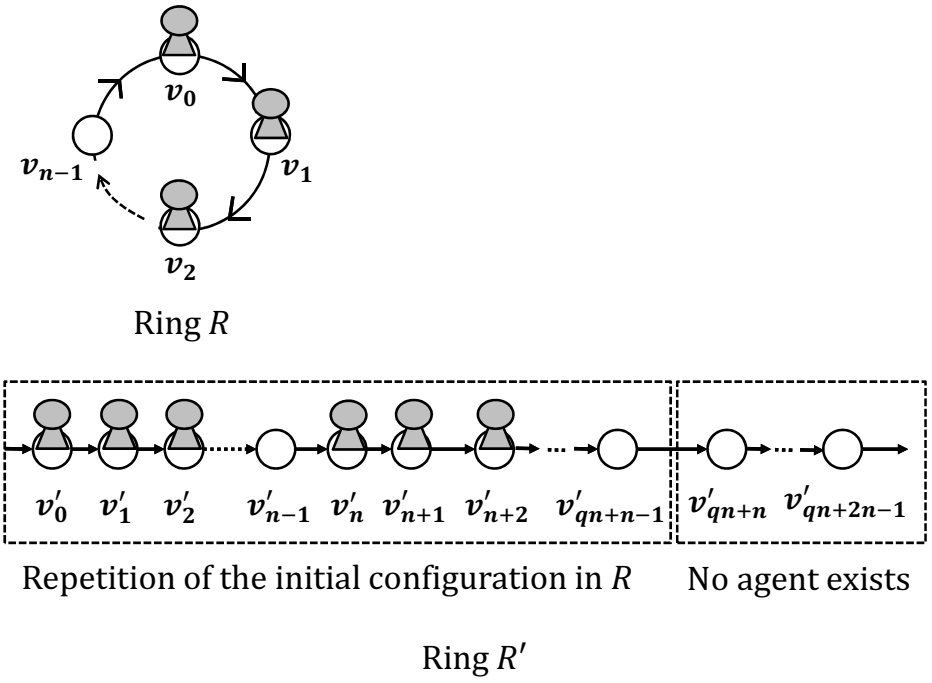

Figure 7: Comparison between R and R'

$E_R = C_0, C_1, \ldots, C_{T(E_R)}$. Note that, in $C_{T(E_R)}$ all agents are in the halt state and every distance between two adjacent agents is d.

Next, let us consider a larger ring R' consisting of $2qn + 2n$ nodes, where q is the minimum integer satisfying $qn \geq T(E_R)$. The comparison between R and R' is shown in Fig. 7. Let $V' = \{v'_0, v'_1, \ldots, v'_{2qn+2n-1}\}$. We consider the initial configuration C'_0 such that $kq + k$ agents $a'_0, a'_1, \ldots a'_{kq+k-1}$ exist in this order in R'. Then, in R' the interval of uniform deployment is $2d$. 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 $v'_{f(i \mod k)+n\cdot\lfloor i/k \rfloor}$. That is, the initial positions for R are repeated from v'_0 to v'_{qn+n-1}, and there is no agent from v'_{qn+n} to $v'_{2qn+2n-1}$. For each node v'_j in R', we define $C_v(v'_j) = v_{j \mod n}$ as the corresponding node of v'_j in R. In the following, we show that each agent $a'_i\,(0 \leq i \leq k-1)$ behaves in the exactly same way as agent a_i in R and a'_i enters the halt state at the same time as a_i. Then, the distance between the two adjacent agents of them is d, which contradicts that the interval of

uniform deployment in R' is $2d$.

At first, we have the following lemma. We 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.

Lemma 1. *Let us consider execution $E_{R'} = C'_0, C'_1, \ldots, C'_{T(E_R)}, \ldots$ for ring R'. We define $V'_t = \{v'_t, v'_{t+1}, \ldots, v'_{qn+n-1}\}$. 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 Lemma 1 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. Hence, agents staying at v'_{j-1} and v'_j in C'_t behave in the exactly same way as those at $C_v(v'_{j-1})$ and $C_v(v'_j)$ in C_t. Since only agents staying at nodes v'_{j-1} and v'_j can change the local configuration of v'_j in unidirectional rings[4], 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, we have the lemma. □

From Lemma 1, in $C'_{T(E_R)}$ local configuration of each node in $V^* = \{v'_{qn}, v'_{qn+1}, \ldots, v'_{qn+n-1}\} \subseteq V'_{T(E_R)}$ is the same as that of the corresponding node in $C_{T(E_R)}$. Note that the set of nodes corresponding to nodes in V^* is equal to V, and every agent in V^* also stops in the halt state in configuration $C'_{T(E_R)}$. Hence, in $C'_{T(E_R)}$ there exist k agents in the halt state in V^*. Then, the distance between the adjacent agents in V^* is d, which is a contradiction.

Therefore, we have the theorem. □

4.2. Uniform deployment problem without termination detection

In this section, we propose an algorithm for agents without knowledge of k or n to solve the uniform deployment problem without termination detection which requires $O((k/l)\log(n/l))$ memory space per agent, $O(n/l)$ time, and $O(kn/l)$ total moves, where l is the symmetry degree of the initial configuration. This result means that when the initial configuration has a

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

higher symmetry degree, agents can solve the problem more efficiently. At first, we consider the case for aperiodic rings. After this, we show that our proposed algorithm achieves uniform deployment also in periodic rings.

4.2.1. Case for aperiodic rings

In Section 3, an agent can detect using knowledge of k when it completes one circuit of the ring. However, in this section agents cannot do this since they have no knowledge of k or n. Hence, at first agents estimate the number of nodes in the ring, and after this they move to their target nodes based on the estimations. Concretely, the algorithm consists of three phases: the estimating phase, the patrolling phase, and the deployment phase. In the estimating phase, each agent a_i moves in the ring and estimates the number of nodes. At the end of this phase, at least one agent estimates the correct number n of nodes as we show later. In the patrolling phase, a_i moves in the ring several times depending on its estimated number of nodes. During the movement, if a_i visits the node where another agent exists, this agent may misestimate the number of nodes and prematurely stop at an incorrect target node. Hence, a_i sends its estimated number of nodes (with some information) to the agent. By this behavior, every agent eventually gets the correct number n of nodes and the location of its correct target node. In the deployment phase, a_i moves to its target node and enters a suspended state. After this, if a_i receives a message from another agent and recognizes that it misestimates the number of nodes, a_i determines its new target node from the message and moves to the node. For simplicity we assume $n = ck$ for some positive integer c in the following description, and this assumption can be removed similarly as in Section 3. In addition, for sequence Y we define $Y^1 = Y$ and $Y^{l+1} = Y^l \cdot Y$ (or concatenation of $(l+1)$ Ys).

Estimating phase. In this phase, each agent a_i firstly releases its token at its home node $v_{HOME}(a_i)$. After this, a_i moves in the ring, measures the distance *dis* between every pair of adjacent token nodes, and stores *dis* to an array D for memorizing the distance sequence. Agent a_i continues such a behavior until it completes estimating the number of nodes. Concretely, a_i continues to move until it observes the same distance sequence repeateadly four times. Let $4n'$ be the number of nodes that a_i ever visited by the time. Then, a_i considers it traveled four times around the ring and estimates the number of nodes in the ring at n'. For example, let us consider Fig. 8. Each number in the figure represents the distance between two adjacent token nodes. Agent a_i moves from node v_j to v'_j and gets the distance sequence

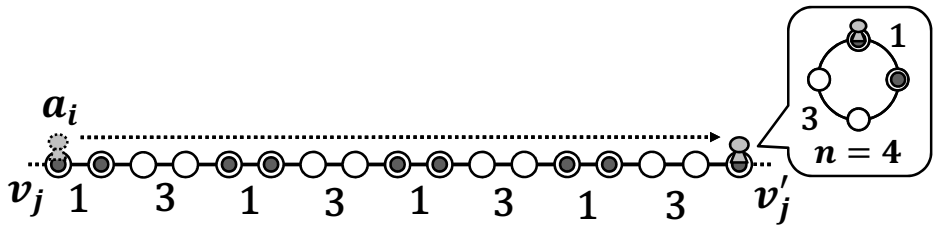

Figure 8: An example that an agent estimates the number of nodes

Algorithm 4 The behavior of agent a_i in the estimating phase

Behavior of Agent a_i

```
1: /* estimating phase */
2: n' = 0, k' = 0, nodes = 0, j = 0
3: release a token at its home node v_HOME(a_i)
4: while n' = 0 do
5:    move to the next token node and get the distance dis between the two
      token nodes
6:    D[j] = dis, j = j + 1
7:    if (j mod 4 = 0) ∧ (∀x (0 ≤ x ≤ j/4 − 1)
      D[x] = D[x + j/4] = D[x + 2 × j/4] = D[x + 3 × j/4]) then
8:       // completing the estimation of the numbers of nodes and tokens
9:       k' = j/4
10:      n' = D[0] + D[1] + ··· + D[k' − 1]
11:      nodes = 4n'
12:   end if
13: end while
14: change to the patrolling phase
```

$D = (1, 3, 1, 3, 1, 3, 1, 3) = (1, 3)^4$. Then, a_i estimates the number of nodes at 4. By this behavior, we can show that 1) at least one agent estimates the correct number n of nodes (in the aperiodic ring), and 2) if the estimated number n' is not correct, $n' \leq n/2$ holds. The pseudocode is described in Algorithm 4. During the estimating phase, a_i uses a variable k' for storing the estimated number of agents (tokens) and a variable *nodes* for storing the number of nodes that a_i has ever visited. These variables (including n' and D) are also used in the patrolling phase and the deployment phase.

Algorithm 5 The behavior of agent a_i in the patrolling phase

Behavior of Agent a_i

```
1: /* patrolling phase */
2: while nodes ≠ 12n' do
3:    move to the forward node
4:    nodes = nodes + 1
5:    if there exists another agent a_h then send (n', k', nodes, D) to a_h
6: end while
7: change to the deployment phase
```

Patrolling phase. In this phase, a_i moves $8n'$ times. Then, a_i considers it traveled twelve times around the ring from the beginning with respect to its estimated number of nodes n'. During the movement, a_i may observe some agent a_h staying at some node. In this case, a_h may misestimate the number of nodes and prematurely stop at an incorrect target node. Hence if a_i observes such an agent, a_i sends $n', k', nodes$, and D to a_h. By this behavior, every agent eventually gets the correct number n of nodes and the location of its correct target node. The pseudocode is described in Algorithm 5.

Deployment phase. In this phase, a_i selects its target node and moves to the node as follows. Let $D = (d_0, d_1, \ldots, d_{k'-1})^4$ be the distance sequence that a_i obtained in the estimating phase. Then, a_i selects its base node similarly to Section 3.1, that is, letting D_{min} be the lexicographically minimum distance sequence among $\{shift(D, x) | 0 \leq x \leq k' - 1\}$, a_i selects its base node $v_{base}(a_i)$ as the home node of the x-th agent of a_i. Note that the x-th agent has the minimum distance sequence D_{min}. In addition, a_i determines its target node and moves to the node similarly to Section 3.1. First, a_i considers that it is the *rank*-th agent ($0 \leq rank \leq k' - 1$) to $v_{base}(a_i)$, where the *rank*-th agent a_i means that there exist $rank - 1$ agents (or tokens) between $v_{HOME}(a_i)$ and $v_{base}(a_i)$. Note that agent a_i staying at $v_{base}(a_i)$ is considered as the 0-th agent. Let *disBase* be the distance from its home node $v_{HOME}(a_i)$ to $v_{base}(a_i)$. Then, a_i firstly moves *disBase* times and reaches $v_{base}(a_i)$. After this, a_i moves to its target node by moving $rank \times n'/k'$ times and enters a suspended state. When all agents enter suspended states, agents solve the uniform deployment problem.

However, a_i may stay at an incorrect target node when it misestimates the number of nodes. In this case, a_i eventually receives a message from

Algorithm 6 The behavior of agent a_i in the deployment phase

Behavior of Agent a_i

1: /* deployment phase */
2: let D_{min} be the lexicographically minimum sequence among $\{shift(D, x) | 0 \leq x \leq k' - 1\}$
3: $rank = \min\{x \geq 0 | shift(D, x) = D_{min}\}$
4: **if** $rank = 0$ **then** $disBase = 0$
5: **if** $rank \neq 0$ **then** $disBase = D[0] + D[1] + \cdots + D[rank - 1]$
6: move $disBase$ times
7: $nodes = nodes + disBase$
8: move $rank \times n'/k'$ times
9: $nodes = nodes + rank \times n'/k'$
10: change its state to a suspended state
11:
12: /* behavior in the suspended state */
13: wait at the current node until a_i receives $(n'_\ell, k'_\ell, nodes_\ell, D)$ from some agent a_ℓ
14: **if** $(n' \leq n'_\ell/2) \wedge$ (there exists t such that $(\forall j\, (0 \leq j \leq 4k' - 1)$ $D[j] = D_\ell[j + t]) \wedge (D_\ell[0] + \cdots D_\ell[t - 1] = nodes_\ell - nodes)$ holds) **then**
15: // a_i recognizes that it misestimates the number of nodes
16: $n' = n'_\ell$, $k' = k'_\ell$, $D = shift(D_\ell, t)$
17: move $12n' - nodes$ times
18: $nodes = 12n'$
19: go to line 2
20: **end if**

another agent a_ℓ. Let n'_ℓ, k'_ℓ, $nodes_\ell$, and D_ℓ be the estimated number of nodes, the estimated number of agents, the number of nodes ever visited, and the distance sequence included in the message from a_ℓ, respectively. If $n' \leq n'_\ell/2$ holds and there exists t such that $(\forall i\, (0 \leq i \leq 4k' - 1)\ D[i] = D_\ell[i + t]) \wedge (D_\ell[0] + \cdots D_\ell[t - 1] = nodes_\ell - nodes)$ holds, it means that a_ℓ estimates at least twice number of nodes than a_i and memorizes a_i's whole distance sequence D as a part of D_ℓ. Then, a_i recognizes that it misestimates the number of nodes and resumes its behavior. Concretely, a_i firstly moves $12n'_\ell - nodes$ times. We can show that $12n'_\ell - nodes$ is always positive, as proved in the proof of Lemma 5. Then, a_i considers it traveled twelve times around the ring from the beginning with respect to the new estimated number

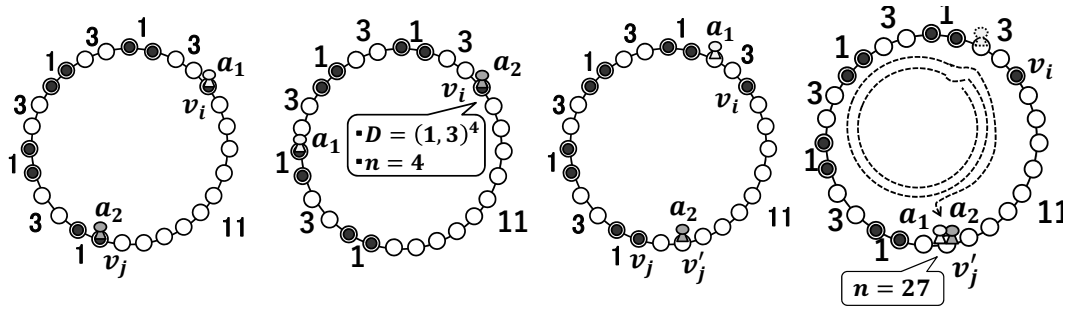

Figure 9: An example in the ring having some periodic subsequence ($n = 27, k = 9, d = 3$)

of nodes n'_ℓ. After this, it determines the new base node and its new target node from n'_ℓ, k'_ℓ, $nodes_\ell$ and D_ℓ, moves to its new target node as mentioned before, and enters a suspended state again. The pseudocode is described in Algorithm 6. When all agents enter suspended states, agents solve the uniform deployment problem.

An example As an example, let us consider the ring in Fig. 9. This ring is aperiodic but has some *periodic subsequence*, that is, some agent observes a 4-times repeated subsequence before it travels once around the ring. In such a ring, some agent misestimates the number of nodes and enters a suspended state at an incorrect target node. However, in this case we can show that at least one agent a_i estimates the correct number n of nodes and informs prematurely suspending agents of n during the patrolling phase. Let us consider the behavior of agents a_1 and a_2. For simplicity, we assume that they behave in a synchronous manner. In the estimating phase, agent a_2 gets the distance sequence $D = (1, 3, 1, 3, 1, 3, 1, 3) = (1, 3)^4$ and estimates the number of nodes at 4, which is incorrect (Fig. 9 (a) to Fig. 9 (b)). After this, a_2 executes the patrolling and the deployment phases, and enters a suspended state at incorrect target node v'_j (Fig. 9 (b) to Fig. 9 (c)). On the other hand, agent a_1 is still in the estimating phase. When a_1 observes $D = (11, 1, 3, 1, 3, 1, 3, 1, 3)^4$, it completes the estimating phase and estimates the correct number of nodes 27. After this, in the patrolling phase a_1 observes a_2 at v'_j, sends its estimated number of nodes with other information to a_2 (Fig. 9 (c) to Fig. 9 (d)), and moves to its target node. When a_2 receives the message from a_1, it recognizes that it misestimates the number of nodes and

resumes its behavior.

In the following, we show that every agent eventually gets the correct number n of nodes and its correct target node. To show this, we use the following lemma.

Lemma 2. [16] *Consider an p-length sequence $A = a_0, \ldots, a_{p-1}$ and an p'-length sequence $B = b_0, \ldots b_{p'-1}$ such that $p' < p$ holds. If B^3 is the prefix of A^3, either $p' \leq p/2$ holds or B is periodic.* □

Then, we have the following lemmas.

Lemma 3. *If agent a_ℓ estimates the incorrect number of nodes n_ℓ (i.e., $n_\ell \neq n$ holds), $n_\ell \leq n/2$ holds.*

Proof. Let $k_\ell\,(< k)$ be the number of agents (tokens) estimated by a_ℓ. Since a_ℓ observes $4k_\ell$ tokens in the estimating phase, it stores the same distance sequence $(D[0], \ldots, D[k_\ell - 1])$ four times, that is, $(D[0], \ldots, D[4k_\ell - 1]) = (D[0], \ldots, D[k_\ell - 1])^4$ holds. Then, $n_\ell = D[0] + \cdots + D[k_\ell - 1]$ holds. On the other hand since the number of tokens in the ring is $k > k_\ell$, sequence $(D[0], \ldots, D[k_\ell - 1])^4$ is the prefix of $(D[0], \ldots, D[k-1])^4$. Note that, $n = D[0] + \cdots + D[k-1]$ holds. Then from Lemma 2, $(D[0], \ldots, D[k_\ell - 1])$ is periodic or $k_\ell \leq k/2$ holds. If $D([0], \ldots, D[k_\ell - 1])$ is periodic, there exists $k'_\ell < k_\ell$ such that $(D[0], \ldots, D[4k'_\ell - 1]) = (D[0], \ldots, D[k'_\ell - 1])^4$ holds. This is a contradiction because a_ℓ should estimate the number of nodes at n_ℓ. Hence, $k_\ell \leq k/2$ holds. Then since $(D[0], \ldots, D[k_\ell - 1])$ is the prefix of $(D[0], \ldots, D[k-1])$, $(D[0], \ldots, D[k-1]) = (D[0], \ldots, D[k_\ell - 1], D[0], \ldots, D[k_\ell - 1], D[2k_\ell], D[2k_\ell + 1], \ldots)$ holds. Thus, $(D[0] + \cdots + D[k_\ell - 1]) \leq (D[0] + \cdots + D[k-1])/2$ holds, that is, $n_\ell \leq n/2$ holds. Therefore, we have the lemma. □

Lemma 4. *If ring R is aperiodic, at least one agent estimates the correct number n of nodes and gets distance sequence D of the initial configuration in R.*

Proof. We show that at least one agent estimates the correct number n of nodes. Then, from Algorithms 4 to 6 the agent clearly gets the distance sequence D for the initial configuration in R. We prove the lemma by contradiction, that is, we assume that the number of nodes estimated by each agent is less than n. We assume that in the initial configuration agents $a_0, a_1, \ldots, a_{k-1}$ exist in this order. We define n_i as the number of nodes estimated by a_i and D_i as the distance sequence observed by a_i. In addition, let S_i be the distance sequence such that $D_i = S_i^4$ holds.

Let a_m be the agent that estimates the maximum number of nodes $n_m\,(< n)$ among all agents, and let $\ell = |S_m|\,(< k)$. We assume that the distance sequence a_m observes in Algorithm 4 is $D_m = (d^m_0, \ldots, d^m_{\ell-1}, d^m_\ell, \ldots, d^m_{2\ell-1}, d^m_{2\ell}, \ldots, d^m_{3\ell-1}, d^m_{3\ell}, \ldots d^m_{4\ell-1}) = (d^m_0, \ldots, d^m_{\ell-1})^4 = S^4_m$. Note that, $S_m = (d^m_0, \ldots, d^m_{\ell-1})$ is aperiodic and $\forall j\ (0 \leq j \leq \ell - 1)\ d^m_j = d^m_{j+\ell} = d^m_{j+2\ell} = d^m_{j+3\ell}$ holds.

Next, let us consider the agent $a_{m+\ell}$. Then, either $n_{m+\ell} < n_m$ or $n_{m+\ell} = n_m$ holds because n_m is the maximum. We show that $n_{m+\ell} = n_m$ always holds by contradiction, that is, we assume that $n_{m+\ell} < n_m$ holds. Then, $|S_{m+\ell}| < |S_m|$ clearly holds. Consequently, $S^3_{m+\ell}$ is the prefix of S^3_m because $a_{m+\ell}$ gets the distance sequence $(d^m_\ell, \ldots, d^m_{2\ell-1}) = S_m$ when it observes ℓ tokens. Then, from Lemma 2 either $|S_{m+\ell}| \leq |S_m|/2$ holds or $S_{m+\ell}$ is periodic. If $|S_{m+\ell}| \leq |S_m|/2$ holds, agent a_m observes $S^4_{m+\ell}$ before observing S^4_m because $(d^m_0, \ldots, d^m_{2\ell-1}) = (d^m_\ell, \ldots, d^m_{3\ell-1})$ contains $S^4_{m+\ell}$ as its prefix. Consequently, a_m estimates the number of nodes at $n_{m+\ell} < n_m$, which is a contradiction. If $S_{m+\ell}$ is periodic, $S_{m+\ell} = (S'_{m+\ell})^t$ holds for some distance sequence $S'_{m+\ell}$ and some positive integer t ($S'_{m+\ell}$ is aperiodic and $|S'_{m+\ell}| \leq |S_{m+\ell}|/2$ holds). Hence, a_m observes $(S'_{m+\ell})^4$ before observing S^4_m and the number of nodes a_m estimates is less than n_m, which is also a contradiction. Therefore, $n_{m+\ell} = n_m$ holds.

Let $m(i) = m + i\ell$ and $A_m = \{a_{m(i)} | \, i \geq 0\}$. As mentioned above, $n_m = n_{m+\ell}$ and $S_{m(0)} = S_{m(1)} = S_m$ hold. In addition, $a_{m(1)}$ observes the same distance sequence of length $4|S_m|$ as $a_{m(0)}$. Hence, recursively $a_{m(i+1)}$ observes the same distance sequence of length $4|S_m|$ as $a_{m(i)}$ and consequently each agent in A_m observes S_m as the first ℓ consecutive distances. When k is divided by ℓ, since every agent $a_{m(i)}$ observes S_m as the first ℓ consecutive distances and $\ell < k$ holds, the ring is periodic, which is a contradiction. In the following, we consider the case that k is not divided by ℓ and show that $S_{m(0)} (= S_m)$ is periodic in this case. When k is not divided by ℓ, $k = \alpha\ell + \beta\,(0 < \beta < \ell)$ holds for some positive integers α and β. Then, the prefix of $S_{m(0)}$ is identical to the suffix of $S_{m(\alpha)}$ because the trajectories of $a_{m(0)}$ and $a_{m(\alpha)}$ include the same part of the ring. We assume that t elements are overlapped, that is, $(d^{m(0)}_0, \ldots, d^{m(0)}_{t-1}) = (d^{m(\alpha)}_{\ell-t}, \ldots, d^{m(\alpha)}_{\ell-1})$ holds. Let T be the sequence consisting of the t overlapped elements and T'_0 (resp., T'_α) be the sequence consisting of the other $(\ell - t)$ elements in $S_{m(0)}$ (resp., $S_{m(\alpha)}$). Then, $S_{m(0)} = TT'_0$ (resp., $S_{m(\alpha)} = T'_\alpha T$) holds (Fig. 10). In addition, $T'_0 = T'_\alpha$ holds because agent $a_{m(\alpha)}$ observes $S^4_{m(\alpha)} = (T'_\alpha T)^4$ and T'_α that $a_{m(\alpha)}$ observes for

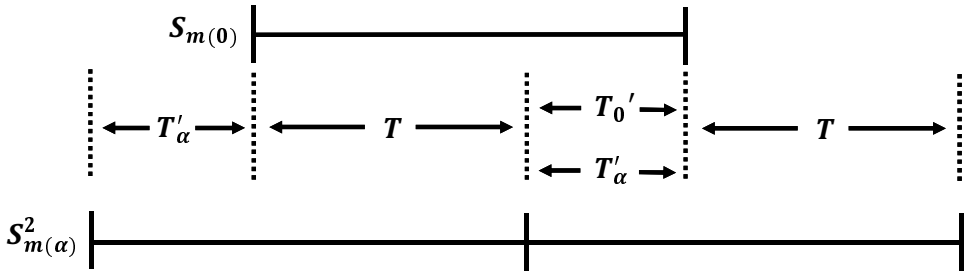

Figure 10: An examples of $S_{m(0)}$ and $S_{m(\alpha)}$

the second time is equivalent to T'_0 that agent $a_{m(0)}$ observes for the first time. Then, since $S_{m(0)} = S_{m(\alpha)}$ holds, $\mathit{shift}(S_{(m(0))}, t) = T'_0T = T'_\alpha T = S_{m(\alpha)} = S_{m(0)}$ holds. Therefore, $S_{m(0)}$ is periodic since $0 < t < \ell$ holds. However, this contradicts the assumption that $S_{m(0)} (= S_m)$ is aperiodic.

Therefore, we have the lemma. □

Lemma 5. *If ring R is aperiodic, every agent eventually gets the correct number n of nodes and distance sequence D of the initial configuration in R.*

Proof. We show that all agents eventually get the correct number n of nodes. Then, from Algorithms 4 to 6 all agents can clearly get distance sequence D of the initial configuration in R. We prove the lemma by contradiction, that is, we assume that when all agents are in the suspended states, there exists at least one agent a_h whose estimated number of nodes n' is less than n. Then from Lemma 3, $n' \leq n/2$ holds. On the other hand, from Lemma 4 at least one agent a_c estimates the correct number n of nodes. In the following we show that a_c observes a_h during the patrolling phase and sends its estimated number of nodes n to a_h, which contradicts the assumption of $n' < n$.

At first, let us consider the number of nodes a_h visits. Let n_1 be the number of nodes a_h estimates in the estimating phase. From Algorithms 4 to 6, a_h moves at most $14n_1$ times by the time a_h enters a suspended state for the first time. After this, we assume that a_h receives messages and updates its estimated number of nodes to $n_2, n_3, \ldots, n_l = n'$ in this order. When a_h updates its estimated number of nodes to n_2, a_h's total moves at that point (i.e, *nodes*) is at most $7n_2$ since $n_1 \leq n_2/2$ holds. Hence, $12n_2 - \mathit{nodes}$ is clearly positive. Then, a_h firstly moves in the ring until its total moves becomes $12n_2$ by moving $12n_2 - \mathit{nodes}$ times. After this, a_h moves to a new

target node and enters a suspended state again. This requires at most $14n_2$ total moves. Then since $n_2 \leq n_3/2$ holds from Algorithm 6, *nodes* is at most $7n_3$ and $12n_3 - nodes$ is clearly positive. Thus recursively, we can show that $12n_i - nodes$ is always positive ($2 \leq i \leq l$) and a_h's total moves is at most $14n' \leq 7n$ unless it does not get the correct number n of nodes. On the other hand, agent a_c moves $8n$ times in the patrolling phase. Thus, a_c clearly observes a_h during the patrolling phase and sends its estimated number n of nodes to a_h suspended at some node, which is a contradiction.

Therefore, we have the lemma. □

Then, we have the following lemma for aperiodic rings.

Lemma 6. *When ring R is aperiodic, agents solve the uniform deployment problem without termination detection.*

Proof. From Lemma 5, all agents eventually get the correct number n of nodes and distance sequence D for the initial configuration in R. Then, each agent can find its correct target node from D and move to the node. Thus, we have the lemma. □

4.2.2. Case for periodic rings

Next, we consider the case for periodic rings. Let R' be a periodic ring and D' be the distance sequence of the initial configuration in R'. We say R' is a (N, l)-node ring if there exists an aperiodic distance sequence D such that $D' = D^l$ holds and the total sum of elements of D is N. Then, $n = Nl$ holds and l is equivalent to the symmetry degree of the initial configuration in R'. We call the ring R with the distance sequence D the *fundamental ring of* R' (e.g., Fig. 11 (a), (b)). Note that an aperiodic ring can be denoted by a $(n, 1)$-node ring. In addition, for each agent a_i in R there exist l agents in R' such that the distance sequence of each agent is l-times repetition of the distance sequence of a_i. We say such agents in R' are *corresponding agents* of agent a_i in R and denote by a_i^j ($0 \leq j \leq l-1$). We assume that agents $a_i^0, a_i^1, \ldots, a_i^{l-1}$ exist in this order and operations to the above index of a_i^j assume calculation under modulo l. Then, the distance from a_i^j to a_i^{j+1} is N. In this case, all agents eventually estimate the incorrect number $N = n/l$ of nodes, but we can show that agents can achieve uniform deployment similarly to in R. Concretely, from algorithms in Section 4.2.1 each agent moves to its target node after considering, based on the estimated number N of nodes, it traveled twelve times around the ring. This means that each agent becomes

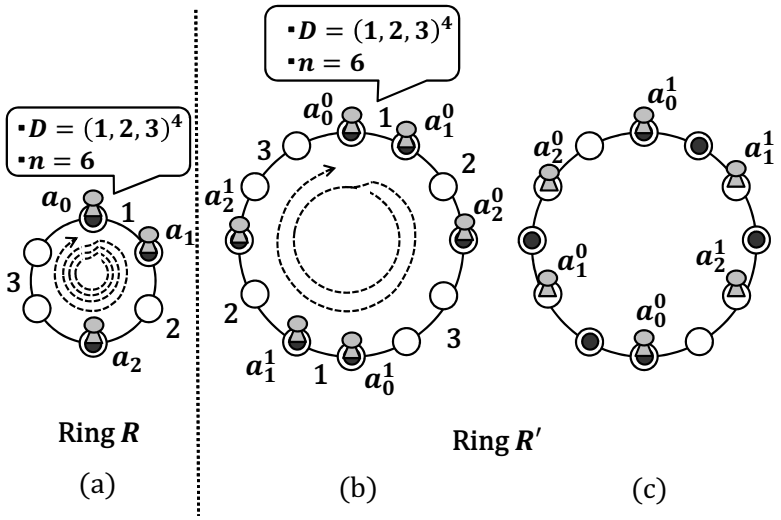

Figure 11: An example for the periodic ring

suspended at its target node during its twelfth or thirteenth circulations in the ring with respect to the estimated size N, which guarantees that when all agents are in the suspended states, no agents stay at the same node and they can achieve uniform deployment. For example, let us consider rings in Fig. 11. Ring R' is a (6,2)-node periodic ring and R is the fundamental ring of R'. In R, each agent estimates the correct number 6 of nodes in the estimating phase and moves to its correct target node (Fig. 11 (a)). On the other hand, in R' each agent also estimates the number 6 of nodes, which is incorrect (Fig. 11 (b)). By algorithms in Section 4.2.1, each agent moves to its target node after considering, based on the estimated size 6, it traveled twelve times around the ring, that is, after each agent moves 72 times (actually, each agent traveled six times around ring R'). This guarantees that when all agents are in the suspended states, no agents stay at the same node and they can achieve uniform deployment (Fig. 11 (c)).

Now, we have the following lemmas, which can be proved similarly to the case of aperiodic rings.

Lemma 7. *Let R' be a (N, l)-node periodic ring and R be the fundamental ring of R'. Let a_i in R be the agent estimating the number N of nodes in the estimating phase. Then, in R' agent a_i^j $(0 \leq j \leq l-1)$ corresponding to a_i also estimates the number N of nodes.*

Proof. From the definition of R', a_i^j observes the same distance sequence as that of a_i. In addition, since agents have no knowledge of k or n, agents determine their estimated number of nodes depending only on the distance sequence they observe. Thus, a_i^j estimates the same number of nodes as that of a_i. □

Lemma 8. *Let R' be a (N, l)-node periodic ring and R be the fundamental ring of R'. Then, in R' every agent eventually gets the number N of nodes and distance sequence D of the initial configuration in R.*

Proof. We show that all agents eventually get the number N of nodes. Then from Algorithms 4 to 6, all agents can clearly get distance sequence D of the initial configuration in R. We prove the lemma by contradiction, that is, we assume that when all agents are in the suspended states, there exists at least one agent a_h whose estimated number of nodes n' is less than N. On the other hand, from Lemma 7 there exists agent $a_c^j\,(0 \leq j \leq l-1)$ estimating the number N of nodes in the estimating phase. Let $A_c = \{a_c^0, a_c^1 \ldots, a_c^{l-1}\}$. In the following, we show that some agent in A_c observes a_h suspended at some node during the patrolling phase and sends its estimated number N of nodes to a_h, which contradicts the assumption of $n' < N$.

At first, let us consider the number of nodes a_h visits. Similarly to the case for aperiodic rings, when a_h updates its estimated number of nodes from n'' to n', it firstly moves in the ring until its total moves becomes $12n'$ by moving $12n' - \mathit{nodes}$ times. After this, a_h moves to a new target node and enters a suspended state again. This requires at most $14n'$ total moves. Hence unless a_h gets the number N of nodes, its total moves is at most $14n' \leq 7N$.

On the other hand, from Lemma 7 there exists agent a_c^j in A_c such that it estimates the number of nodes at N and the distance from $v_{HOME}(a_c^j)$ to $v_{HOME}(a_h)$ is less than N. Recall that, $v_{HOME}(a)$ is the home node of agent a. Then, let us consider the behavior of agent a_c^{j-4}. Agent a_c^{j-4} firstly moves $4N$ times and finishes the estimating phase at node $v_{HOME}(a_c^j)$. After this, a_c^{j-4} moves $8N$ times from $v_{HOME}(a_c^j)$ in the patrolling phase. On the other hand, a_h moves at most $7N$ times from $v_{HOME}(a_h)$. Since the distance from $v_{HOME}(a_c^j)$ to $v_{HOME}(a_h)$ is less than N, a_c^{j-4} observes a_h suspended at some node during the patrolling phase and sends the number N of nodes to a_h, which is a contradiction.

Therefore, we have the lemma. □

Lemma 9. *Even when ring R' is periodic, agents solve the uniform deployment problem without termination detection.*

Proof. From Lemma 8, all agents eventually get the number N of nodes and distance sequence D of the initial configuration in R, where R is the fundamental ring of R'. From Algorithm 6, when agent a_i^j gets the number N of nodes it firstly moves in the ring until its total moves becomes $12N$. Then, a_i^j is at $v_{HOME}(a_i^{j+12})$. After this, a_i^j finds its target node from D and moves to the node, which requires at most $2N$ moves. Hence, a_i^j eventually stays between $v_{HOME}(a_i^{j+12})$ and $v_{HOME}(a_i^{j+14})$. This means that letting $v_{base}(a_i^j)$ (resp., $v'_{base}(a_i^j)$) be the base node existing between $v_{HOME}(a_i^{j+12})$ and $v_{HOME}(a_i^{j+13})$ (resp., $v_{HOME}(a_i^{j+13})$ and $v_{HOME}(a_i^{j+14})$) a_i^j eventually stays between $v_{base}(a_i^j)$ and $v'_{base}(a_i^j)$. Moreover, it clearly holds the total moves of each of a_i^j $(0 \leq j \leq l-1)$ are the same. Thus when all agents are in the suspended states, no agents stay at the same node and agents can achieve uniform deployment.

Therefore, we have the lemma. □

Finally, we have the following theorem for (N, l)-node rings.

Theorem 6. *For agents with no knowledge of k or n, the proposed algorithm solves the uniform deployment problem without termination detection. This algorithm requires $O((k/l)\log(n/l))$ memory space per agent, $O(n/l)$ time, and $O(kn/l)$ total moves.*

Proof. From Lemmas 6 and 9, agents solve the uniform deployment problem without termination detection. In the following, we analyze complexity measures.

At first, we evaluate the memory requirement per agent. Each agent eventually gets the distance sequence $D = (d_0, d_1, \ldots, d_{(4(k/l))-1})$. Since each d_i is at most n/l, this sequence requires $O((k/l)\log(n/l))$ memory. Moreover, the other variables require $O(\log(n/l))$ bit memory. Therefore, the memory requirement per agent is $O((k/l)\log(n/l))$.

Next, we analyze the time complexity. Let $A_{correct}$ be the set of agents that estimate the number $n/l\,(= N)$ of nodes in the estimating phase. Each agent $a_c \in A_{correct}$ finishes its patrolling phase in $12n/l$ time units, and moves to its correct target node, which requires at most $14n/l$ time units from the beginning of the algorithm. In addition, from the proof of Lemmas 5 and 8 each agent $a_h \notin A_{correct}$ gets the number n/l of nodes within $12n/l$ time units since each $a_c \in A_{correct}$ finishes its patrolling phase in $12n/l$ time units. After this, a_h requires at most $14n/l$ time units to moves to its correct target node from the beginning of the algorithm. Thus, the time complexity is $O(n/l)$.

At last, we analyze the total number of agent moves. Each agent requires at most $14n/l$ moves to move to its target node. Thus, the total number of agent moves is $O(kn/l)$. □

5. Conclusion

In this paper, we considered the uniform deployment problem of mobile agents in asynchronous unidirectional ring networks. The uniform deployment problem, which is a striking contrast to the rendezvous problem, is interesting to investigate. We proposed three algorithms to solve the uniform deployment problem from any initial configuration such that all agents are in the initial state and placed at the distinct nodes. These algorithms utilize the essential characteristic of the uniform deployment problem: the problem aims to attain the symmetry, and these algorithms solve the problem without breaking symmetry that the initial agent locations have. Such an approach in designing mobile agent algorithms seems to be applicable to other problems that aim to attain the symmetry.

As a future work, we will consider the uniform deployment problem in networks other than rings, such as tree networks and general networks. This problem may be solved by embedding a ring in the networks [12, 13] and applying the idea in this paper. Concretely, for tree networks agents embed the ring by the Euler tour technique, that is, if an agent moves in the tree network by the depth-first manner and visits $2(n-1)$ nodes, the agent can see the nodes as a virtual ring of with $2(n-1)$ nodes. For general network, agents can embed a ring by constructing a spanning tree and embedding a ring in the spanning tree. Since an embedded ring consists of $2(n-1)$ nodes for an original network with n nodes, we can show that the total moves between the embedded ring and the original network is asymptotically equivalent.

References

[1] D. Kotz S. R. Gray, G. Cybenko, A.R. Peterson, and D. Rus. D'agents: Applications and performance of a mobile-agent system, Software: Practice and Experience. 32(6):543–573, 2002.

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

[4] 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.

[5] B. Pagurek A. Bieszczad and T. White. Mobile agents for network management. IEEE Communications Surveys, 1(1):2–9, 1998.

[6] E. Kranakis and D. Krizanc. An algorithmic theory of mobile agents. International Symposium on Trustworthy Global Computing, Vol. 4661. pages 86-97, 2006.

[7] S. Lipperts and B. Kreller. Mobile agents in telecommunications networks - a simulative approach to load balancing. Proc. of 5th Information systems analysis and synthesis, pages 231–238, 1999.

[8] X Wang SK Das J Cao, Y Sun. Scalable load balancing on distributed web servers using mobile agents. Journal of Parallel and Distributed Computing, 63(10):996–1005, 2003.

[9] P. Flocchini, G. Prencipe, and N. Santoro. Self-deployment of mobile sensors on a ring, Theoretical Computer Science. 402(1):67–80, 2008.

[10] Y. Elor and A.M. Bruckstein. Uniform multi-agent deployment on a ring, Theoretical Computer Science. 412(8):783–795, 2011.

[11] L. Barriere, P. Flocchini, E. Mesa-Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid, International Journal of Foundations of Computer Science. 22(03):679–697, 2011.

[12] S. Dolev. *Self-stabilization*. MIT press, 2000.

[13] Y. Yamauchi, T. Masuzawa, and D. Bein. Preserving the fault-containment of ring protocols executed on trees. The Computer Journal, 52(4):483–498, 2008.

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

[15] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in a ring, Proc. of the 6th Latin American Theoretical Informatics, LNCS, Vol. 2976. pages 599–608, 2004.

[16] S. Kawai, F. Ooshita, H. Kakugawa, and T. Masuzawa. Randomized rendezvous of mobile agents in anonymous unidirectional ring networks, Proc. of the 19th International Colloquium on Structural Information and Communication Complexity, LNCS, Vol. 7355. pages 303–314, 2012.

[17] E. Kranakis, D. Krozanc, and E. Markou. The mobile agent rendezvous problem in the ring, Synthesis Lectures on Distributed Computing Theory, Vol. 1. pages 1–122, 2010.

[18] 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.

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

[20] G. Tel. Introduction to distributed algorithms. Cambridge university press, 2000.