Vol.2010-AL-128 No.4
2010/1/26

Rendezvous of Asynchronous Mobile Agents in Trees

Daisuke Baba ,[†1] Tomoko Izumi ,[†2]
Fukuhito Ooshita ,[†1] Hirotsugu Kakugawa [†1]
and Toshimitsu Masuzawa [†1]

This paper reveals the relation between the time complexity and the space complexity for the rendezvous problem with k agents in asynchronous tree networks. The rendezvous problem requires that all the agents in the system have to meet at a single node within a finite time. We first prove that at least $\Omega(n)$ memory size per agent is required to solve the rendezvous problem in $O(n)$ time where n is the number of nodes. Next, we present the rendezvous algorithm which terminates in $O(n)$ time. The space complexity of this algorithm is also $O(n)$ per agent. From this lower/upper bound, $\Theta(n)$ memory size per agent is necessary and sufficient to solve the problem in $O(n)$ time (asymptotically time-optimal). Finally, we present the asymptotically space-optimal rendezvous algorithm. This algorithm has space complexity $O(\log n)$ and time complexity $O(\Delta n^8)$ where Δ is the maximum degree of the tree.

1. Introduction

1.1 Background and Motivation

In this paper, we are interested in the relation between the time and the memory size for each mobile agent to solve the rendezvous problem. In the problem, agents, which are initially distributed in a network, have to meet on a single node. The rendezvous problem is one of the fundamental problems that are required for a lot of agent systems. For example, an application may require rendezvous to share the information of all the agents.

To solve the rendezvous problem is easy if each node in a network has a unique identifier or ID: Each agent explores the network and terminates at the node with the smallest ID. However, such unique ID may not be available for the agents in some cases. It is prohibited to publish the unique ID to agents for security reasons, or the agents cannot store it because of its small memory. Hence, it is important to design algorithms which work in anonymous networks.

In anonymous networks, agents using the same deterministic algorithm cannot meet at a single node if there are cycles in the network. The problem can be feasible with some additional assumptions such that agents can leave marks on nodes or the network topology is restricted. We are interested in the rendezvous problem where agents with the same deterministic algorithm meet without leaving marks. To solve the problem in this model, we consider only tree networks.

In the rendezvous problem, the time complexity and the space complexity are important metrics to show the efficiency of algorithms. Because agents are moving entities in the computer network, the size of each agent, that is the memory space of the agent, is desired to be small. In addition, meeting at a single node is not a goal of the agent system but the means to achieve another task. Therefore, the algorithm which achieves rendezvous in a short time is required.

1.2 Related Work

A number of researchers have studied the rendezvous problem in a variety of models. In the area of anonymous networks and anonymous agents, identical tokens, which are put on nodes, are often used to solve the problem since Kranakis et al. showed that two agents in a ring network can meet by the same deterministic algorithm using a token[11]. If one token is available for each agent, the rendezvous problem in a synchronous ring is solvable for any number of agents[4),6)]. The lower bound on the space complexity in this model is $\Omega(\log k + \log\log n)$ where k is the number of agents and n is the number of nodes, and the asymptotically space-optimal algorithm is proposed for uni-directional ring networks by Gasieniec et al.[6)]. The effect of token failure is also considered in some papers[1)–3)].

The model described above is different from ours in terms of the availability of the memory of nodes. Fraigniaud et al. proved that two anonymous agents can rendezvous in any synchronous tree network without using a token unless the tree is symmetric[5)]. This is the same model as we consider in this paper. The memory size of this algorithm is $O(\log n)$ and this is asymptotically optimal. Although the authors say that the algorithm does not work for more than two

†1 Graduate School of Information Science and Technology, Osaka University, Osaka, Japan.
†2 College of Information Science and Engineering, Ritsumeikan University, Shiga, Japan.

© 2010 Information Processing Society of Japan

agents, we believe it can be easily extended to work for any number of agents.

Lastly, we refer to another model that agents are oblivious[8),9)]. This model may be weaker in that agents cannot remember any information of the past. However, agents in this model can take the snapshot of the system, which is different from ours.

1.3 Our Contributions

In this paper, we first prove that it takes at least $\Omega(n)$ time to solve the rendezvous problem in trees with n nodes. Then, we show that there are trees in which $\Omega(n)$ memory size is necessary to solve the rendezvous problem in $O(n)$ time. Moreover, we present the asymptotically space-optimal algorithm on the condition that the time complexity is also asymptotically optimal, i.e., both the time complexity and the space complexity are $O(n)$. Next, we are interested in whether the rendezvous problem is solvable with lower memory space. In fact, the answer is positive. We present an algorithm such that any number of agents with $O(\log n)$ memory space can rendezvous in any asynchronous tree unless the tree is symmetric. This space complexity is asymptotically optimal. This algorithm attains a significant improvement compared to the previous one[5)] in that this algorithm is applicable for any number of agents and any asynchronous non-symmetric tree.

2. Terminology and Preliminaries

2.1 The Network Model

We consider the tree network $T = (V, E)$ with $n = |V|$ nodes where V is the set of anonymous nodes and E is the set of undirected edges. A tree network is an arbitrary connected network with no cycle. Every node $u \in V$ has some ports, each of which connects to an edge. The number of ports node u has is denoted by $\delta(u)$ or *degree* of u. Every port is assigned a *port number* $\lambda_u(e)$ from the set $\{0, \ldots, \delta(u)-1\}$, which is unique at u, using a *local labeling function* λ_u. This local labeling function is determined at each node and there is no coherence between $\lambda_u(e)$ and $\lambda_v(e)$ for any $e = \{u, v\} \in E$.

There are $k \geq 2$ agents with no identifiers in the tree T. Each agent has a bounded amount of memory. Each node can host at most k agents, but it does not provide agents with any whiteboard, which is the local memory of a node agent can freely read and write. Each agent initially stays at a node called its *home node* and starts the same deterministic algorithm at any time. The agents have no priori knowledge about the network or other agents, that is, they do not know n, k, the shape of the tree, or where other agents are. After the algorithm is started, the agent can move in the network by the following three operations: 1) When the agent walks across an edge e into node v (resp. immediately after the agent initiates the algorithm at node v), it memorizes $\lambda_v(e)$ (resp. 0) and the degree of v. 2) The agent computes internally at v, and determines the port number it leaves next or notices that it should terminate. 3) If the agent decides to move to the neighboring node, it leaves node v through the port which is determined by the previous operation. These actions, such as computing or moving, are progressed asynchronously in the sense that the processing period is finite but there is no assumption of the upper bound on the length of the period. If two or more agents passing through the same edge, they cannot detect each other regardless of the direction of each agent.

2.2 Definition of Terms and Problem

The *path* $P(v_0, v_k) = (v_0, v_1, \ldots, v_k)$ with length k is a sequence of nodes from v_0 to v_k such that $\{v_i, v_{i+1}\} \in E$ $(0 \leq i < k)$ and $v_i \neq v_j$ if $i \neq j$. Note that, for any $u, v \in V$, $P(u, v)$ is unique in a tree. The *distance* from u to v, denoted by $dist(u, v)$, is the length of the path from u to v. The *eccentricity* $r(u)$ of node u is the maximum distance from u to an arbitrary node, i.e., $r(u) = \max_{v \in V} dist(u, v)$. The *diameter* D of the network is the maximum eccentricity in the network. The *radius* R of the network is the minimum eccentricity in the network. A node with eccentricity R is called *center*.

A tree T is *symmetric* iff there exists a function $g : V \to V$ such that all the following conditions hold:

(1) For any $v \in V$, $v \neq g(v)$ holds.
(2) For any $v \in V$, the $\delta(v)$ and $\delta(g(v))$ are the same value.
(3) For any $u, v, g(u), g(v) \in V$, u is adjacent to v iff $g(u)$ is adjacent to $g(v)$.
(4) For any $\{u, v\}, \{g(u), g(v)\} \in E$, $\lambda_u(\{u, v\})$ is equal to $\lambda_{g(u)}(\{g(u), g(v)\})$.

In the *rendezvous problem*, $k \geq 2$ agents have to meet at a single node, which is not predetermined. The node all the agents meet is called *rendezvous point*. However, if the tree T is symmetric, there are some cases that agents with the

© 2010 Information Processing Society of Japan

same deterministic algorithm cannot meet at a single node[5]. For this reason, we modify the requirement of the rendezvous problem: All the agents meet and terminate at a single node if T is not symmetric, otherwise all the agents gather and terminate at two neighboring nodes. We say an algorithm $\mathcal{A}$ *solves* the rendezvous problem if agents executing $\mathcal{A}$ satisfy the above conditions for any tree, any location of home nodes, any starting time of agents, and any execution of agents. The efficiency of an algorithm is measured by the *time complexity* and the *space complexity*. The time complexity is defined as the maximum number of movements for an agent because there is no assumption about the period of each action of agents in asynchronous system. The space complexity is defined as the maximum number of bits an agent requires to store all the local variables.

2.3 Basic Properties

In the followings, we show basic properties of tree networks.

Theorem 1 *There exist one or two center nodes in a tree. If there exist two center nodes, they are neighbors[10].* ■

Theorem 2 *Let v' be the farthest node from node v in a tree (i.e., $dist(v, v') = r(v)$). The eccentricity of node v' is equal to the diameter of the tree, that is, $r(v') = D$.* ■

Theorem 3 *Let the distance from node u to node v be D. The node c is a center if and only if c is included in the path $P(u, v)$ and $r(c) = \lceil \frac{D}{2} \rceil$ holds[10].* ■

3. Asymptotically Time-optimal Rendezvous Algorithm

3.1 Lower Bound on the Memory Space

In this section, we discuss the lower bound on the time complexity and the space complexity of algorithms for the rendezvous problem. As for the time complexity, Theorem 4 clearly holds.

Theorem 4 *To solve the rendezvous problem in a tree network with n nodes, it takes at least $\Omega(n)$ time to terminate.* ■

Next, we consider how much memory space on each agent is required for asymptotically time-optimal rendezvous algorithm.

Theorem 5 *If an algorithm $\mathcal{A}$ is asymptotically time-optimal (i.e. $O(n)$) one for the rendezvous problem in a tree network with n nodes, then it requires $\Omega(n)$ memory space per agent.* ■

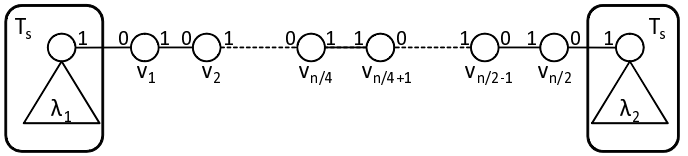

Fig. 1 The labeled tree (T, λ) we consider in Theorem 5

Proof. In our model, every agent has to determine the rendezvous point. The reason of this is that the agents do not have the number k of agents and any communication tools with other agents. In addition, they operate asynchronously, and when moving on a edge, they do not detect other agents passing the same edge. Thus, there is an asynchronous execution that no agent meets with another agent until it terminates.

We define the *labeled tree* (T, λ) as the tuple of the tree T and the *labeling function* $\lambda = \bigcup_{v \in V} \lambda_v$. Consider the labeled tree (T, λ) described in Fig. 1. In (T, λ), each node v_i is connected to only v_{i-1} and v_{i+1} for $1 < i < \frac{n}{2}$, and port numbers are symmetric for these nodes. Moreover, node v_1 is connected to a labeled tree (T_s, λ_1) with $\frac{n}{4}$ nodes and node $v_{\frac{n}{2}}$ is connected to (T_s, λ_2). Note that, given a tree with $O(n)$ nodes, there exists a tree such that at least $2^{\Omega(n)}$ kinds of labeled trees can be defined (for example, line network). If T_s is such a tree, it requires $\Omega(n)$ bits of memory to compare λ_1 with λ_2.

Assume that an algorithm $\mathcal{A}$ solves the rendezvous problem in $O(n)$ time with $o(n)$ space of memory per agent and prove by contradiction. Consider an agent executing $\mathcal{A}$ in (T, λ). If the given labeled tree (T, λ) is symmetric (i.e., $\lambda_1 = \lambda_2$ holds), the agents gather at two neighboring nodes because there is no algorithm to meet at a single node in this case[5]. Otherwise (i.e., $\lambda_1 \neq \lambda_2$ holds), all the agents have to meet at a single node. Thus, the action of the agent when (T, λ) is symmetric must be different from that when (T, λ) is not symmetric. However, the agent executing $\mathcal{A}$ cannot check whether $\lambda_1 = \lambda_2$ holds or not. Thus, the agent cannot change its action in these two cases. This conflicts with the assumption that $\mathcal{A}$ solves the rendezvous problem. □

© 2010 Information Processing Society of Japan

3.2 The Algorithm with $O(n)$ Memory Space

3.2.1 Outline of the Algorithm

In this subsection, we present the asymptotically time-optimal algorithm Rendezvous-T, which requires at most $O(n)$ memory space per agent. In Rendezvous-T, each agent traverses the tree, finds a center, and terminates there (i.e., the rendezvous point is the center). Note that, since each agent does not have any device to influence other agents, each agent does such works independently.

First, we introduce the *basic step*, which is the way of movements for agents. In the basic step, when the agent arrives at node u through the port i, it leaves u through the port $(i+1) \mod \delta(u)$ in the next step. The agent starts its basic steps by leaving the port 0. By continuing the basic steps, the agent realizes DFS-traversal. We also define *reverse step* as the backward step of the basic step. In our algorithms, the agent starts reverse step only after its basic steps. When an agent starts the reverse step at a node, it leaves the node through the port passed in the previous step.

We assume that each agent starts Rendezvous-T at a leaf node of the tree. If the home node of an agent is not a leaf, the agent moves to a leaf without memory by using basic steps. A leaf node where the agent starts the algorithm is called its *start node*. In Rendezvous-T, each agent performs the following six phases to find a rendezvous point. Fig. 2 shows the locations of the agent according to the execution sequence.

Phase 1 The agent computes the eccentricity $r(s)$ of the start node s.

Phase 2 The agent moves to the farthest node v' from s (we call node v' *second node*).

Phase 3 The agent computes $r(v')$ of the second node v' (i.e., $r(v') = D$ from Theorem 2).

Phase 4 The agent moves to the farthest node v'' from v' (we call node v'' *third node*).

Phase 5 The agent moves back to the first node c which satisfies $dist(v', c) = \lceil \frac{D}{2} \rceil$. (The node c is one of the centers from Theorem 3.)

Phase 6 If there exist two centers (i.e., D is odd), the agent chooses the rendezvous point from two centers and terminates there.

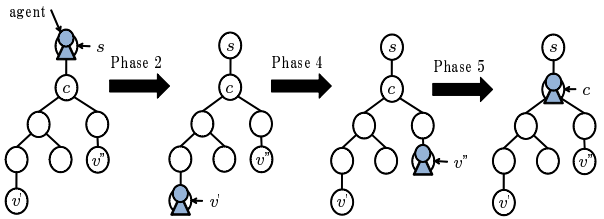

Fig. 2 Location of the agent

In Phase 1 to 3, each agent computes the diameter D. After that, the agent moves to the center. Note that there may be one or more nodes whose distance from the second node v' is $\lceil \frac{D}{2} \rceil$. To detect the center among them correctly, the agent once moves to the third node v'' whose distance from v' is D because the center node is on the path from v' to v'' (refer Theorem 3). That is, the center node is the first node whose distance from v' is $\lceil \frac{D}{2} \rceil$ and that the agent visits after leaving from v''. These works are done in Phase 4 to 5. In Phase 6, the agent terminates at the rendezvous point, which is one of the centers. Note that, the agent can also understand whether the tree is symmetric or not in Phase 6.

To realize Rendezvous-T, we introduce two functions MoveAndCompute and Choose. By calling function MoveAndCompute(h_1, h_2) at node v (called *initial node*), the agent starts DFS-traversal of the tree using basic steps. The agent continues the basic steps until it visits node s_2 satisfying $dist(v, s_2) = h_2$ after it visits node s_1 satisfying $dist(v, s_1) = h_1$, and stops the execution of MoveAndCompute at s_2. The agent keeps the maximum distance from the initial node v to a visited node during the execution of MoveAndCompute, and the maximum distance is returned as the output of MoveAndCompute. Function MoveAndCompute is used in Rendezvous-T as follows: In Phase 1, the agent executes MoveAndCompute$(1, 0)$ at a start node s to compute $r(s)$ by visiting all nodes and to return to s. And in Phase 2, it moves to v' with $dist(s, v') = r(s)$ by calling MoveAndCompute$(0, r(s))$ at s. In Phase 3, MoveAndCompute$(1, 0)$ is executed just like Phase 1. In Phase 4 and 5, by calling MoveAndCompute$(D, \lceil \frac{D}{2} \rceil)$ at the second node v', the agent moves to the third node and moves back to center c. Function Choose chooses a rendezvous point from two centers, and it is used to

© 2010 Information Processing Society of Japan

achieve Phase 6 when there exist two centers.

In the function MoveAndCompute, the agent has to compute the distance from its initial node v to a current node. To compute the distance in the anonymous networks, the agent uses the port numbers for the following strategy: Whenever the agent leaves a node u and moves to the adjacent node, it checks whether the step leads it close to or away from its initial node v. The following lemma implies that the agent can decide it by checking the port number the agent will move to. From the property of trees, the lemma clearly holds.

Lemma 1 *Let v be an initial node of* MoveAndCompute. *Assume that an agent has arrived at u for the first time through the port i. When the agent leaves u through the port i', the agent gets close to v if $i = i'$, and it gets away from v otherwise.* ■

3.2.2 Implementation of MoveAndCompute

In this subsection, we explain how to implement MoveAndCompute in Rendezvous-T. By Lemma 1, the agent can calculate the distance from the initial node v to the current node u by computing whether the following condition is satisfied or not: The port through which the agent will pass to leave u is the same as the one through which it first visited u. To compute it correctly, the agent keeps the sequence $H = h_1 h_2 \dots$ called *history*. The i-th element h_i of the history indicates the i-th movement of the basic steps. Each movement is kept by the fact whether the agent gets close to the initial node v or gets away from v. In more detail, $h_i =$'+' if the agent gets away from v in the i-th movement, and $h_i =$'−' otherwise. Note that, since each movement is kept with one bit and the agent moves at most $2(n-1)$ times in each MoveAndCompute, the agent requires at most $O(n)$ memory space to keep the history. By using the history, when the agent leaves a node, it calculates whether it gets close to the initial node v or away from v from the following lemma.

Lemma 2 *We assume that an agent visits a node u through the port i' after l basic steps in* MoveAndCompute, *and its history is $H_0 = h_1, h_2, \dots, h_l$. Let i be the one through which the agent first visits u in the* MoveAndCompute. *Then, the followings hold.*

Case1: *If $h_l =$'+' holds, $i' = i$ holds.*

Case2: *If $h_l =$'−' holds, we define $H_1, H_2, \dots$ as follows: Let S_0 be the minimum suffix of H_0 in which the number of '+' is equal to the number of '−'. Then, we define H_1 as the prefix of H_0 such that $H_0 = H_1 S_0$. If the last element of H_1 is '−', we can define H_2 in similar way. We continue the definitions, and assume the last element of H_t is '+'. Then, $i = (i' - t) \bmod \delta(u)$ holds.* ■

From Lemma 2, when the agent visits u, it can locally compute the port passed when it first visited u. Thus, the agent can determine whether it gets away from the initial node v or close to v in the next step locally at u.

We explain the implementation of MoveAndCompute(h_1, h_2) as follows. Let d be the distance from an initial node v to a current node and d_{max} be the maximum number of d the agent has computed. The agent prepares an empty history and sets two variables d and d_{max} to be 0 at v. In MoveAndCompute(h_1, h_2), it performs the following three operations when the agent leaves u: 1) By using the history, the agent determines whether it gets close to the initial node v or away from v in the next step. 2) Next, it moves to the neighboring node by the basic step. 3) After that, it updates the history and two values d and d_{max}. The agent can calculate the distance from v to each node it has visited by performing above three operations repeatedly. If d becomes h_2 after d becomes h_1 once, the agent stops the execution of MoveAndCompute.

3.2.3 Implementation of Choose

Here, we explain the implementation of Choose in Rendezvous-T. If the diameter D is even, the algorithm Rendezvous-T is terminated without executing Choose because there is only one center in this case. Thus, we assume that D is odd, i.e., there are two centers in the tree. Let the agent exist at node c, which is one of the centers, after it has completed the execution in Phase 5. Let c' be another center and e be the edge that connects two centers c and c'. When the agent starts the execution of Choose, the agent knows the value of n and D, and it recognizes e in the execution of MoveAndCompute. These values are computed in Phase 1, Phase 3, and Phase 5, respectively. We consider the two connected components T_c and $T_{c'}$ by removing edge e from T which include c and c' respectively.

To choose one of the centers as the rendezvous point, each agent compares T_c with $T_{c'}$. However, if the agent keeps the complete map of T_c and $T_{c'}$, it requires $\Omega(n \log n)$ bits. Fortunately, we have a strategy to identify a tree T_c with only

© 2010 Information Processing Society of Japan

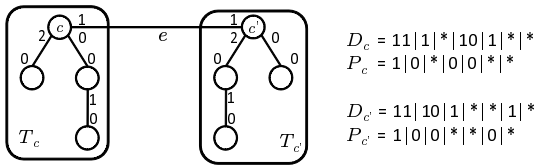

Fig. 3 An example of execution Choose

$O(n)$ bits. In this strategy, the agent keeps only the degree and the port number of one edge for every node in T_c. The port number $\lambda_u(e)$ kept by the agent on node u is the one connecting the first edge on path $P(u, c)$.

To realize above strategy, the agent prepares two sequences D_c and P_c. These sequences are used to keep the value $\delta(u)$ and $\lambda_u(e')$ for every node in T_c (See Fig. 3). Initially, both D_c and P_c are empty words. These sequences are updated whenever the agent makes a movement: When the agent arrives at a node u through an edge e', it adds the value $\delta(u)$ (resp. $\lambda_u(e')$) if it visits u for the first time, and otherwise it adds a symbol $*$. In our algorithm, the sequences D_c and P_c are kept by the agent as a string over alphabet $\Sigma = \{0, 1, |, *\}$. Symbols 0 and 1 are used to represent a degree or a port number as a binary value, and symbol | is inserted between numbers as a separator. For example, sequence $31 * 2$ is represented by $11|1| * |10$. When the agent explores the whole of tree T_c from c using basic steps, it can identify the tree T_c from the tuple (D_c, P_c).

The algorithm of Choose consists of three operations: 1) The agent explores the whole of tree T_c using basic steps, and gets the tuple (D_c, P_c). 2) It moves another center c', explores the whole of tree $T_{c'}$, and gets the tuple $(D_{c'}, P_{c'})$. 3) It compares the tuple (D_c, P_c) with $(D_{c'}, P_{c'})$ lexicographically. If these tuples are different, it terminates at the smaller node (c or c'), otherwise it terminates at c' because the tree is symmetric.

3.2.4 Efficiency of the algorithm

From the proposed algorithm Rendezvous-T, we can state the following theorem.

Theorem 6 *The rendezvous problem in a tree network with n nodes is solved in $O(n)$ time with $O(n)$ memory space on each agent.* ■

Proof. The time complexity of Rendezvous-T is clearly $O(n)$ because the agent makes at most $2(n-1)$ basic steps in each execution of MoveAndCompute or Choose. Thus, we focus on the space complexity in the followings.

In MoveAndCompute, the agent keeps its history whose length is at most $2(n-1)$ because the number of steps the agent makes is at most $2(n-1)$. Other variables the agent keeps are clearly at most $O(\log n)$ bits each. Thus, it suffices $O(n)$ memory space to execute MoveAndCompute.

Next, we show the space complexity of Choose. To keep all the variables except for the sequences D_c, P_c, $D_{c'}$ and $P_{c'}$, $O(n)$ memory space is sufficient. Thus, we show the number of bits to keep these sequences in the followings.

First we consider the number of bits to keep the sequence D_c. Since the number of nodes included in T_c is lower than n, both of the number of symbol | and that of symbol $*$ included in D_c are at most $n-1$. Let d_c be the sum of symbol 0 and symbol 1 included in D_c. Since symbols 0 and 1 are used in D_c to represent a degree as a binary value for every node u in T_c, the inequality $d_c = \sum_{v \in T_c} (\lfloor \log \delta(v) \rfloor + 1) \leq \sum_{v \in T_c} \delta(v) \leq 2(n-1)$ holds. Thus, the length of D_c is at most $4(n-1)$. Note that since Σ has four symbols, the sequence D_c is stored using at most $8(n-1)$ bits. Next, we consider the number of bits to keep the sequence P_c. By the definition of labeling function λ_v, each port number on node u is lower than $\delta(u)$ and the number of bits to keep the sequence P_c is not larger than that of D_c. The numbers of bits required for other sequences are shown in the same way and the space complexity of Choose is also $O(n)$. □

4. Asymptotically Space-optimal Rendezvous Algorithm

4.1 The Algorithm with $O(\log n)$ Memory Space

In this subsection, we present an asymptotically space-optimal rendezvous algorithm Rendezvous-S which uses $O(\log n)$ memory space per agent. The idea of the algorithm is the same as that of previous algorithm Rendezvous-T: The agent moves to a center by executing MoveAndCompute four times, and then chooses one of the centers as the rendezvous point by executing Choose. Moreover, the idea of MoveAndCompute is almost same. Whenever the agent moves to the adjacent node, it computes the distance from the initial node v to the current node u. However, two points are significantly different. One is how the agent determines whether the next step makes it close to or away from v in MoveAndCompute. The other is how the agent chooses one of the centers as the rendezvous point.

© 2010 Information Processing Society of Japan

Because the agent can use only $O(\log n)$ memory space, it can keep none of the history, the sequence D_c, nor the sequence P_c. Thus, we focus on only these two points in the followings.

Before we provide solutions to them, we need to explain two existing functions LogExploration and MatchingEdge, which are proposed to solve the exploration problem for trees[7]. The space complexity of LogExploration is proved to be $O(\log n)$, and the time complexity to be $O(\Delta n^7)$[7], where Δ is the maximum degree of nodes in the network. We use these functions as subroutines in MoveAndCompute. These functions are proposed in the *symmetric-label* tree, where each edge has the same label in both sides. Thus, in the followings, we consider only symmetric-label trees. Note that, however, this restriction is removed in Section 4.2.

Function LogExploration is the one to realize the exploration in arbitrary trees: The agent traverses all the edges and all the nodes in a tree started at its home node v. After the execution of LogExploration, the agent returns back to v. Moreover, the agent can recognize whether the tree is symmetric or not by executing LogExploration. If and only if the tree is symmetric, there exists exactly one edge called *orphan edge* defined by the topology of the tree. In fact, the orphan edge corresponds to an edge connecting two centers in a symmetric tree. Therefore, the agent can check whether the tree is symmetric or not by checking whether there is an orphan edge. As a subroutine in LogExploration, function MatchingEdge is presented. Let $S = e_1 e_2 \dots e_{2(n-1)}$ be a sequence of edges that an agent passes through when the agent explores whole of the tree from node v using basic steps. Since basic steps realize DFS-traversal, each edge is included in S exactly twice. By calling MatchingEdge(i) at v, the agent can compute $j(\neq i)$ satisfying $e_i = e_j$ in S if the tree is not symmetric.

Now, we are ready to settle the first problem: How the agent determines whether the next step makes it close to or away from the initial node v in MoveAndCompute? The approach explained from now is available only if the tree is not symmetric. However, this never causes a problem: Before the agent starts MoveAndCompute (i.e., before Phase 1 in Section 3.2.1), it executes LogExploration to check whether the tree is symmetric or not. If the tree is symmetric, the agent moves to a node adjacent to the orphan edge and terminates there. Thus, the agent executes MoveAndComute only if the tree is not symmetric. Assume the agent visits node u on its $(l-1)$-th basic step. Let $S = e_1 e_2 \dots e_{2(n-1)}$ be the sequence of edges that the agent will pass through in $2(n-1)$ basic steps from its initial node v. The behavior of the agent consists of the following three operations: 1) The agent returns to v by using $(l-1)$ reverse steps. 2) By calling MatchingEdge(l) at v, it computes j satisfying $e_j = e_l$ in S. If $l < j$ holds, the agent passes e for the first time at the l-th step and thus the l-th step makes the agent away from v. Otherwise, since the agent has passed e before the l-th step, the l-th step makes the agent close to v. Note that, only when the agent executes MatchingEdge(l) at v, it can compute whether it passes through e for the first time or not by this way. 3) After the computation, the agent gets back to u by $(l-1)$ basic steps and makes l-th step.

Next, we explain the implementation of Choose to settle the second problem: How the agent chooses one of the centers as the rendezvous point? When the agent starts Choose, it stays at a center c with recognizing another center c', the edge e connecting c and c', and the value of $f = 2(n-1)$. Besides, the tree T is not symmetric (Otherwise, the agent terminates before it starts MoveAndCompute). Let $t_c[1..j] = p_1, \dots, p_j$ be the sequence of port numbers the agent has left during j basic steps from c. In Choose, the agent compares $t_c[1..f]$ with $t_{c'}[1..f]$ lexicographically and terminates at a center with the smaller one.

The detail of Choose in Rendezvous-S is described as follows. Initially, the agent sets the variable i to be 1. The agent makes i basic steps from c and gets the value $t_c[i]$. Next, it returns to c by i reverse steps and moves to c'. Similarly, it gets the value of $t_{c'}[i]$. If $t_c[i]$ is different from $t_{c'}[i]$, then it terminates at the node with the smaller one. Otherwise, the variable i is incremented by one and compares $t_c[i]$ with $t_{c'}[i]$ repeatedly. Since the tree is not symmetric, there is an integer i such that $t_c[i] \neq t_{c'}[i]$.

Since the space complexity of LogExploration is $O(\log n)$ and the time complexity of it is $O(\Delta n^7)$, we can state the following theorem.

Theorem 7 *The rendezvous problem in a tree network with n nodes is solved in $O(\Delta n^8)$ time with $O(\log n)$ memory space on each agent.* ■

4.2 Extension from symmetric-label trees to general trees

In the previous subsection, we consider only symmetric-label trees. How-

© 2010 Information Processing Society of Japan

ever, our algorithm works for general trees. The method to obtain a virtual (symmetric-label) tree T' from an original (general) tree T has been introduced[7]. In this method, a virtual node x is put on every edge $e = \{u, w\}$ such that $\lambda_u(e) \neq \lambda_w(e)$, and port numbers $\lambda_u(e)$ and $\lambda_w(e)$ are assigned to $\lambda_x(\{x, u\})$ and $\lambda_x(\{x, w\})$. Since virtual tree T' is a symmetric-label tree, the agent can solve the rendezvous problem by executing Rendezvous-S in T'.

We should handle two troubles caused by using the method. First, when the rendezvous point is a virtual node, the agent moves the neighboring real node. That is, if center c is a virtual and the only center, it leaves c through the port 1 and terminates at the arrival node. If there are two centers and one of them is a virtual node, the agent terminates at the real one. Second, when the agent recognizes that virtual tree T' is symmetric and it moves to one of nodes w and w' adjacent to the orphan edge on T', the agent should check whether the real tree T is symmetric or not: Let $v_w(i)$ be *true* (resp. *false*) if the agent visits a real (resp. virtual) node after i basic steps from w. If $v_w(j) = v_{w'}(j)$ holds for all $j < i$ and $v_w(i) \neq v_{w'}(i)$ for some i, the agent can terminate at w (resp. w') if $v_w(i) =$*true* (resp. $v_{w'} =$*true*). If $v_w(i) = v_{w'}(i)$ holds for all the integer i $(1 \leq i \leq 2(n' - 1)$, where n' is the number of nodes in T' computed in LogExploration), T is also symmetric and terminate at w without meeting there.

5. Conclusion

In this paper, we have presented two rendezvous algorithms which work with any number of agents in any asynchronous trees. One is the asymptotically time-optimal algorithm and the other is the space-optimal one. The space complexity of first algorithm is also asymptotically optimal on the condition that the time complexity is asymptotically optimal.

However, our current study does not deal with the solvability in the case that the tree is symmetric and the initial locations of agents is asymmetric. It would be an interesting problem whether all the agents can meet at a single node or not in that case. If it is possible, how much is its time and memory cost?

Acknowledgment

This work is supported in part by Global COE Program of MEXT, Grant-in-Aid for Scientific Research ((B)19300017, (B)20300012) of JSPS, and the Kayamori Foundation of Informational Science Advancement.

References

1) Das, S.: Mobile Agent Rendezvous in a Ring Using Faulty Tokens, *Proc. 9th International Conference in Distributed Computing and Networking(ICDCN 2008)*, pp. 292–297 (2008).
2) Das, S., Mihalak, M., Sramek, R., Vicari, E. and Widmayer, P.: Rendezvous of Mobile Agents When Tokens Fail Anytime, *Proc. 12th International Conference on Principles of Distributed Systems*, pp.463–480 (2008).
3) Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N. and Sawchuk, C.: Mobile Agents Rendezvous When Tokens Fail, *Proc. 11th Colloquium on Structural Information and Communication Complexity(SIROCCO 2004)*, pp.599–608 (2004).
4) Flocchini, P., Kranakis, E., Krizanc, D., Sawchuk, C. and Santoro, N.: Multiple Mobile Agents Rendezvous in a Ring, *Proc. 6th Latin American Theoretical Informatics(LATIN 2004)*, pp.599–608 (2004).
5) Fraigniaud, P. and Pelc, A.: Deterministic rendezvous in trees with little memory, *Proc. 22nd International Symposium on Distributed Computing(DISC 2008)*, pp. 242–256 (2008).
6) Gasieniec, L., Kranakis, E., Krizanc, D. and Zhang, X.: Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring, *Proc. 32nd International Conference on Current Trends in Theory and Practice of Computer Science(SOFSEM 2006)*, pp.282–292 (2006).
7) Gasieniec, L., Pelc, A., Radzik, T. and Zhang, X.: Tree Exploration with Logarithmic Memory, *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms(SODA 2007)*, pp.585–594 (2007).
8) Klasing, R., Kosowski, A. and A.Navarra: Taking advantage of symmetries: gathering of asynchronous oblivious robots on a ring, *Proc. 12th International Conference on Principles of Distributed Systems*, pp.446–462 (2006).
9) Klasing, R., Markou, E. and Pelc, A.: Gathering asynchrounous oblivious mobile robots in a ring, *Theoretical Computer Science*, Vol.390, No.1, pp.27–39 (2008).
10) Korach, E., Rotem, D. and Santoro, N.: Distributed Algorithms for Finding Centers and Medians in Networks, *ACM Transactions on Programming Langages and Systems*, Vol.6, No.3, pp.380–401 (1984).
11) Kranakis, E., Krizanc, D., Santoro, N. and Sawchuk, C.: Mobile Agent Rendezvous in a Ring, *Proc. 23rd International Conference on Distributed Computing Systems(ICDCS 2003)*, pp.592–599 (2003).

© 2010 Information Processing Society of Japan