A Study on Sub-linear Time Algorithms in Mobile Agent Systems

モバイルエージェントシステムにおける劣線形時間アルゴリズムについての研究

2022

Ryota Eguchi

Abstract

Mobile agent is an abstract notion that is widely used to model several distributed computing systems consisting of mobile computing entities, such as robots, web crawlers, and a collection of sensor devices. Broadly speaking, those systems are divided into the two categories: autonomous mobile system and passively mobile system. The main difference of these categories lies at the type of mobility. In the autonomous mobile systems, agents can autonomously move in some specified locations, according to the program (i.e., algorithm) that is installed to agents in advance. On the other hand, in passively mobile systems, the program of each agent cannot control its physical behavior (i.e., how they move) because it is a computing device attached to other moving entities such as cars, animals or humans, i.e., examples of such passively mobile system are ranging from flock of birds for nature observation to humans with smartphone or cars with sensor devices for computing some attributes of the groups. Various computing models are proposed to represent autonomous/passively mobile systems, and much research has been devoted to reveal the computational power of these models, as well as complexity analyses for various problems. In this dissertation, we explore the capabilities of autonomous/passively mobile systems by focusing on their representative models, namely the population protocol model and mobile agents on graphs. For the passively mobile system, one of the promising models is called population protocol model, introduced by Angluin et al. [12]. A Population protocol consists of n anonymous and identical agents, each of which is defined as a deterministic state machine. The communication among agents is performed by pairwise interactions, where two interacting agents change their states following the transition function (algorithm) deployed to all agents. An interaction corresponds to the physical behavior that two agents get close in the region and exchange information through their short-range (wireless) communication channels. An execution of a population protocol is a sequence of pairwise interactions. Since an interaction is a consequence of the physical movement by agents, the system cannot control which pair of agents interact. As an autonomous mobile agent system, this dissertation deals with one of the representative models, so-called the graphical mobile agent system. In the model, mobile agents are located at any vertices in a graph $G = (V, E)$, and they repeatedly move to an adjacent node to solve a given task, such as graph traversal and rendezvous.

In this dissertation, we consider the sub-linear time solvability of several problems in passively/autonomous mobile agent systems. The sub-linear time is namely $o(n)$ or $o(m)$ time, where n is the number of vertices of input graph and m is the number of edges. The notion is one of the standard criteria of the efficiency for the parallelism in several distributed computing systems. Conceptually, the sub-linear time algorithms are important for the following reasons: The first reason is that the linear time algorithm does not achieve essential optimality in the situation of distributed computing systems because of their parallelism, while it does in the sequential computing. In fact, it is not realistic to apply linear time algorithms for a large scale network. Second, in principle some specific problems can be solved without scanning the whole network topology. For those reasons, it is a natural and important question if the sub-linear time solvability is achievable for a given task. However, it is highly challenging to reveal such a solvability since in standard computing models the capability of computing entities is often limited, and/or for a certain kind of the problems their problem specifications inherently requires the scan of the whole input. Hence to the goal of sub-linear time algorithms, it is often indispensable to introduce some additional assumptions and/or the relaxation of problems. The central research question of this dissertation is what is the sufficient conditions or minimal assumptions for achieving sub-linear time solvability. We explore the answer of this question for the two fundamental problems of distributed computing theory. Specifically, in the population protocol model, we consider a general form of the aggregation problem with a base station. The base station is a special agent having the computational power more powerful than others. In the aggregation problem, the base station has to sum up for inputs distributed to other agents. We propose an algorithm that solves the aggregation problem in sub-linear parallel time using a relatively small number of states per agent. More precisely, our algorithm solves the aggregation problem with input domain X in $O(\sqrt{n}\log^2 n)$ parallel time and $O(|X|^2)$ states per agent (except for the base station) with high probability.

In the graphical mobile agent system we consider the rendezvous problem. In the rendezvous problem, two mobile agents located at different vertices in a graph have to meet at the same vertex. In this dissertation, we consider the synchronous neighborhood rendezvous problem, where the agents are initially located at two adjacent vertices. While this problem can be trivially solved in $O(\Delta)$ rounds (Δ is the maximum degree of the graph), it is highly challenging to reveal whether that problem can be solved in $o(\Delta)$ rounds, even assuming the rich computational capability of agents. The only known result is that the time complexity of $O(\sqrt{n})$ rounds is achievable if the graph is complete and agents are probabilistic, asymmetric, and can use whiteboards placed at vertices. Our main contribution is to clarify the situation (with respect to computational models and graph classes) admitting such a sublinear-time rendezvous algorithm. More precisely, we present two algorithms achieving fast rendezvous additionally assuming bounded minimum degree, unique vertex identifier, accessibility to neighborhood IDs, and randomization. The first algorithm runs within $\tilde{O}(\sqrt{n\Delta/\delta} + n/\delta)$ rounds for graphs of the minimum degree larger than $\sqrt{n}$, where n is the number of vertices in the graph, and δ is the minimum degree of the graph. The second algorithm assumes that the largest vertex ID is $O(n)$, and achieves $\tilde{O}\left(\frac{n}{\sqrt{\delta}}\right)$-round time complexity without using whiteboards. These algorithms attain $o(\Delta)$-round complexity in the case of $\delta = \omega(\sqrt{n}\log n)$ and $\delta = \omega(n^{2/3}\log^{4/3} n)$ respectively.

On the negative side, we also present the impossibility of sublinear-time rendezvous when we relax the assumptions. There lie four unconventional assumptions for our algorithm, which are bounded minimum degrees, the accessibility to neighborhood IDs, initial distance one, and randomization. Interestingly, the time lower bound of $\Omega(n)$ rounds for graphs of $\Delta = \Theta(n)$ is deduced even if we remove only one of them; this implies that our algorithm runs under a minimal assumption.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background and Motivation

The distributed computing systems are computing systems in which a collection of computing entities perform computation through interactions among them and achieves a certain type of goals. In the systems, one of the problems of the computation is often lack of the information about other computing entities. Thus it is considered that the procedures for aggregating information among the entities or gathering a certain position for moving entities. In principle, such problems are often solvable if reasonable amount of time is consumed, so technical challenge lies at the time complexity of the problem (that is, time to solve the problem). Recently, sub-linear time solvability plays a central role in the problem of time complexity. The sub-linear time solvability is often to achieve a certain goal without using trivial procedures, though it differs in the situation that what trivial means. For example, if we want to aggregate whole information of all processors, one solution for the problem is to elect a leader among the processors and the leader interact (or communicate) with all of the other processors. For the sub-linear time solvability, one should design an algorithm without using such (trivial) procedures.

Mobile agent is an abstract notion that is widely used to model several distributed computing systems consisting of mobile computing entities, such as robots, web crawlers, and a collection of sensor devices. Broadly speaking, those systems are divided into the two categories: **autonomous** mobile system and **passively** mobile system. The main difference of these categories lies at the type of mobility. In the autonomous mobile systems, agents can autonomously move in some specified locations, according to the program (i.e., algorithm) that is installed to agents in advance. On the other hand, in passively mobile systems, the program of each agent cannot control its physical behavior (i.e., how they move) because it is a computing device attached to other moving entities such as cars, animals or humans, i.e., examples of such passively mobile system are ranging from flock of birds for nature observation to humans with smartphone or cars with sensor devices for computing some attributes of the groups. Various computing models are proposed to represent autonomous/passively mobile systems, and much research has been devoted to clarify the computational power of these models and exploring complexities of various problems. In this dissertation, we explore the capabilities of the autonomous/passively mobile systems by studying representative models of each systems, namely mobile agents on a graph and population protocol model.

For the passively mobile system, one of the promising models is called population protocol model, introduced by Angluin et al. [12]. A Population protocol consists of n anonymous and identical agents, each of which is defined as a deterministic state machine. The communication among agents is performed by pairwise interactions, where two interacting agents change their states following the transition function (algorithm) deployed to all agents. An interaction corresponds to the physical behavior that two agents get close in the region and exchange information through their short-range (wireless) communication channels. An execution of a population protocol is a sequence of pairwise interactions. Since an interaction is a consequence of the physical movement by agents, the system cannot control which pair of agents interact. Formally, the sequence of interactions is determined by an abstract daemon called *scheduler*, which is out of the control by algorithm designers. Throughout this dissertation, we assume the probabilistic scheduler, which chooses next interacting pairs uniformly and independently at random from all pairs.

As autonomous mobile agent system, this dissertation deals with one of the representative models that we call in this dissertation graphical mobile agent system. In the model, mobile agents are located at any vertices in a graph $G = (V, E)$, and they repeatedly move to an adjacent node to solve a given task, such as graph traversal and rendezvous.

1.2 Overview of The Dissertation

In this dissertation, we consider the sub-linear time solvability of several problems in passively/autonomous mobile agent systems. The sub-linear time is namely $o(n)$ or $o(m)$ time, where n is the number of vertices of input graph and m is the number of edges. The notion is one of the standard criteria of the efficiency for the parallelism in several distributed computing systems. Conceptually, the sub-linear time algorithms are important for the following reasons: The first reason is that the linear time algorithm does not achieve essential optimality in the situation of distributed computing systems because of their parallelism, while it does in the sequential computing. In fact, it is not realistic to apply linear time algorithms for a large scale network. Second, in principle some specific problems can be solved without scanning the whole network topology. For those reasons, it is a natural and important question if the sub-linear time solvability is achievable for a given task. However, it is highly challenging to reveal such a solvability since in standard computing models the capability of computing entities is often limited, and/or for a certain kind of the problems their problem specifications inherently requires the scan of the whole input. Hence to the goal of sub-linear time algorithms, it is often indispensable to introduce some additional assumptions and/or the relaxation of problems. The central research question of this dissertation is what is the sufficient conditions or minimal assumptions for achieving sub-linear time solvability. We explore the answer of this question for the two fundamental problems of distributed computing theory. Specifically, our main results are summarized as follows.

1.2.1 Sub-linear Time Aggregation in Probabilistic Population Protocol Model

The dissertation considers a general form of *aggregation problems* in the population protocol model with a base station. The base station is a special agent having more powerful computational power. Precisely, an aggregation problem is specified by any commutative monoid $(X, +)$. Each agent i (except for the base station) initially has a value $x_i \in X$, and eventually the base station must output the value of $\hat{x} = \sum_i x_i$. It should be noted that the binary operator + is not necessarily the arithmetic addition over integers. This problem covers many popular tasks in the population protocol models, such as total sum, counting, and majority: The total sum is obvious, and counting and majority are both special cases of the total sum. The counting problem is the total sum with input $x_i = 1$ for any ordinary agent i, and the majority problem is the one with input $x_i \in \{+1, -1\}$. In addition, the aggregation problem is also used in some of known algorithms explicitly or implicitly as a building block.

Any aggregation problem can be solved by the *coalescence* algorithm [12], where the interaction between two agents with values a and b respectively results in the values $a + b$ and 0. One can see that the coalescence algorithm trivially finishes the aggregation when only one agent i has a non-zero value in x_i. Unfortunately, it needs $\Theta(n^2)$ steps of interactions (i.e., $\Theta(n)$ parallel time*) for convergence because the

*Parallel time is a measurement often adopted in the literature of population protocols with the probabilistic scheduler, where n consecutive steps of interactions are regarded as a unit time.

probability that the last two agents having non-zero values interact is $O(1/n^2)$.

In this dissertation, we present a new algorithm which solves the aggregation problem with sub-linear parallel time in the probabilistic population protocol model with the base station. More specifically, for any monoid $(X, +)$, our algorithm achieves the aggregation in $O(\sqrt{n} \log^2 n)$ time with high probability. The number of states used by each agent is $O(|X|^2)$ (equivalently, $O(\log |X|)$ bits). The comparison with prior work is shown in Table 1.1. The first result by Angluin et al. [12] is the standard coalescence algorithm we mentioned above. The second result [13] is the algorithm for simulating register machines in the population protocol model with the initial leader. Since the register machine model can count the number of agents with a specific state, it is possible to solve the aggregation problem for any monoid $(X, +)$ by counting the number of agents with input x sequentially for all $x \in X$. However, this approach takes the running time depending on the alphabet size $|X|$. The algorithm in [13] takes $O(\text{polylog}(n))$ time for counting the number of a given state (or input), solving the aggregation problem needs $O(|X|\text{polylog}(n))$ time in total. To best of our knowledge, our algorithm is the first result achieving the sub-linear running time independent of $|X|$.

Table 1.1: Comparison with previous work

	Angluin at el. [12]	Angluin at el. [13]	This dissertation
Time	$O(n)$	$O(\|X\| \cdot \log^5 n)$	$O(\sqrt{n} \log^2 n)$
#States	$O(\|X\|)$	$O(\|X\|)$	$O(\|X\|^2)$
Initial leader	no	yes	
Success probability	1	$1 - 1/n^{O(1)}$	

1.2.2 Fast Neighborhood Rendezvous

The *rendezvous problem* is well-studied in distributed computing theory. A typical setting of the problem requires two agents located at any vertices in a graph $G = (V, E)$ to meet and halt. This is recognized as a fundamental problem for designing distributed algorithms of mobile agents. The hardness of symmetry breaking is often seen as an essential difficulty of the rendezvous problem. For example, we consider a ring network of four vertices, and the situation that the two agents located at two vertices that are not adjacent to each other. Then, the agents running the same algorithm symmetrically move and thus their relative distance two is kept forever.

That is, any deterministic algorithm does not achieve rendezvous in this situation. To make the rendezvous problem solvable, the system model must be equipped with some mechanism enabling two agents to move asymmetrically. Much of the previous work focuses on what models or assumptions provide such a capability [37,42,49].

Unlike the viewpoint mentioned above, we assume a model that easily breaks symmetry, i.e., allowing randomized and/or asymmetric algorithms, and focuses on the time complexity of the rendezvous problem. When we allow two agents to run different algorithms, the rendezvous problem can be solved using graph exploration. Specifically, one of the agents halts at the initial location and the other one traverses all the vertices. Hence the time complexity of graph exploration is a trivial upper bound for the rendezvous problem. In contrast, the half of the initial distance between two agents is a trivial lower bound for the problem. Since both of the bounds can be $\Theta(n)$ in a specific class of n-vertex instances (e.g., a ring network of n vertices) the exploration-based approach is existentially optimal, but not universally optimal. When the initial distance is small in terms of n, the approach based on graph exploration does not necessarily exhibit optimal algorithms. However, due to the unavailability of the location information of other agents, achieving rendezvous without exploring all vertices is a highly non-trivial challenge, even if we assume stronger capability of agents such as randomization, asymmetry, and non-obliviousness.

Contribution

In this dissertation, we consider what instances and what computational power of models (oracles) admit efficient algorithms that do not use exhaustive search strategy, such as graph exploration. As we stated, the key characterization of the instances is distance of initial location of both agents. We consider the initial distance is small in terms of n, to avoid $\Omega(n)$ lower bound. In this setting, the meaning of "without exhaustive search" will be clear, namely presenting algorithms that achieve rendezvous in $o(n)$ rounds.

In this dissertation, we consider an extreme variant of the rendezvous problem, called the *neighborhood rendezvous problem*, where two agents are initially located at *two adjacent vertices* (i.e., initial distance one). This problem can be also seen as a generalized version of the rendezvous problem in complete graphs [11] because in that case any two agents always have distance one. Since the neighborhood rendezvous problem can be trivially solved in $O(\Delta)$ rounds (Δ is the maximum degree of the graph), the technical challenge lies in the design of algorithms achieving rendezvous within $o(\Delta)$ rounds. As well as the algorithm shown in [11], we assume the rich capability of agents (i.e., randomized, asymmetric, and non-oblivious), unique vertex identifiers, and the availability of whiteboards placed at each vertex. In addition, we assume that agents at a vertex v can know the IDs of all v's neighbors (which is analogous to the *KT1 model* in message passing systems [55]). Specifically, we present two randomized algorithms. The first algorithm achieves rendezvous within $O\left(\frac{n}{\delta}\log^2 n + \sqrt{\frac{n\Delta}{\delta}}\log n\right)$ rounds with high probability for graphs whose minimum degree is larger than $\sqrt{n}$. Thus, this algorithm achieves fast rendezvous (i.e., sublinear of Δ) in graphs with minimum degree $\delta = \omega(\sqrt{n}\log n)$. The second algorithm trades the use of whiteboards into the assumption of *tight* naming of vertices, that is, the assumption that the largest vertex ID is $O(n)$. It achieves rendezvous within $O\left(\frac{n}{\sqrt{\delta}}\log^2 n\right)$ rounds with high probability[†], and thus fast rendezvous is attained in the case of $\delta = \omega(n^{2/3}\log^{4/3} n)$. While these algorithms are designed for the specific case of initial distance one, it is easy to extend them to address general initial distance. That is, we can obtain the rendezvous algorithms with *adaptive running time*, which attains a sublinear time for some nice setting (i.e., $d = 1$ and an appropriate lower bounds for δ), and even guarantees the rendezvous for any setting.

On the negative side, we also present the impossibility of sublinear-time rendezvous when we relax the assumptions. There lie four unconventional assumptions for our algorithm, which are bounded minimum degrees, the accessibility to neighborhood IDs, initial distance one, and randomization. Interestingly, the time lower bound of $\Omega(n)$ rounds for graphs of $\Delta = \Theta(n)$ is deduced even if we remove only one of them; this implies that our algorithm runs under a minimal assumption.

1.3 Related Work

1.3.1 Sub-linear Time Aggregation in Probabilistic Population Protocol Model

The population protocol model first appeared in the seminal papers by Angluin et al. [12, 14], where the main interest is to clarify the class of computable predicates with inputs distributed over all agents. The authors prove that any predicate stably computable in the fundamental model of population protocols is semi-linear (or equivalently, definable by Presburger arithmetics). Then several fundamental problems in the context of distributed computing, such as leader election [4, 18, 27, 58], majority, plurality consensus [5, 50], self-stabilizing algorithm [15, 19, 20, 45] are explored in the population protocol model.

[†]Throughout this dissertation, we say that an event $\mathcal{E}$ holds *with high probability* if $\Pr[\mathcal{E}] \geq 1 - 1/n^{O(1)}$ holds

The paper by Angluin et al. [13] studies the predicate computation in the system with one unique leader under the probabilistic scheduler. They present a fast (i.e., $O(\text{polylog}(n))$ parallel time) algorithm to simulate an abstract register machine. The authors also present a technical tool called *phase clock*, which is also a key tool for our algorithm. Since this result yields the fast predicate computation by installing a fast (i.e., polylogarithmic parallel time) leader election algorithm, a number of time-complexity analyses for the leader election problem are considered. The primary negative result in this context is by Doty and Soloveichik [40]. It proved that any stable leader election algorithm utilizing $O(1)$ states per agent requires $\Omega(n)$ parallel time to convergence, where the stability means that the system must elect a leader with probability one. On the positive side, Alistarh and Gelashvili [4] proposed a polylogarithmic-time leader election algorithm using $O(\log n)$ states per agent. Recently, Alistarh et al. explored the trade-offs between the time to convergence and the space of states per agent [2] . They show that any protocol which solves leader election (or majority) using $O(\log\log n)$ states per agent must take $\Omega(n/\text{polylog}(n))$ parallel time. Alistarh et al. [3] shows that any protocol solving stabilizing majority problem in expected time $o(n^{1-c})$ requires $\Omega(\log n)$ states for any constant $c \quad (0 < c < 1)$, and propose space-optimal majority protocol. In [44], a space optimal leader election protocol which uses $O(\log\log n)$ states per agent and converges parallel time $O(\log^2 n)$ is proposed. The model with a base station is first considered in the context of the counting problem [20]. A few follow-up papers are also published, where the space-efficient (self-stabilizing) solutions with respect to agent memory are presented [16, 45].

Recent progress of population protocol model is as follows. Recent results about considering time/space complexity of majority and leader election problems are summarized in the survey paper by Elsasser and Radzik [41]. As a new protocol for majority problem, Berenbrink et al. [21] present a protocol which has $O(\log n)$ states and $O(\log^{5/3} n)$ stabilization time with high probability. The state complexity of population protocol model is initiated in the paper [23], and papers [22, 30] also consider the problem. Roughly speaking, the problem is the minimal number of states needed to evaluate systems of linear constraints, which is a large subclass of the predicates computed by population protocols. In particular, the first paper presents a protocol for so-called the flock-of-birds problem, namely the family of predicates $\{x \geq \eta : \eta \in \mathbb{N}\}$, where η is a fixed constant. This problem is interpreted as evaluating if the number of agents is larger or equal than the constant η. The first paper shows that the state complexity of the protocol is $O(\log \eta)$ for leaderless situation (without assumption of initial existence of leaders). In the paper [22], Blondin et al. present a protocol having $O(\log\log \eta)$ states with initial existence of leaders. As a lower bound, Czerner and Esparza [30] show that the state complexity of $x \geq \eta$ is $\Omega(\log\log\log\eta)$ for leaderless protocols. On the self-stabilizing leader election problem, Burman et al. [26] present a time-optimal protocol, which has optimal $O(n)$ parallel time and $O(n)$ states. A previously known result is presented by Cai, Izumi, and Wada [28], where the paper presents a protocol having n states and $O(n^2)$ expected time, and shows that any protocol solving self-stabilizing leader election with silent property must have at least n states. The silent property ensures that after the system stabilized, all agents do not change their states according to the transition function. The time/space tradeoffs of loosely stabilizing leader election problem are also considered. The term "loosely" roughly means that a protocol must stabilize to the desired configurations (ones with a leader), but allowing a tiny probability for abandoning the desired configurations. This definition leads an interpretation that the protocol stabilize to configurations with a leader, and the configurations last sufficiently large long time (called holding time). For this problem, many papers improve the stabilizing time and the states needed, or explore the lower bound for the problem [46, 56, 58, 59]. In paper [56], sudo et al. present a loosely stabilizing leader election protocol with $O(\text{polylog}(n))$ states, $O(\tau\log n)$ convergence time, and $O(n^\tau)$ holding time for a design parameter $\tau \geq 1$. This protocol is time-optimal since leader election requires $O(\log n)$ parallel time to stabilize [57]. A general form of majority problem is also considered [47]. In this paper, Alistarh et al. consider the comparison problem, where initially fixed but possibly small counts of agents have opinions X_0, Y_0 (with a constraint of $|X_0| \geq C|Y_0|$ for some constant $C > 1$), and the system must stabilize a consensus about which opinion has higher initial count. For the problem, the authors present a protocol that stabilizes $O(\log n)$ parallel time and uses $O(\log n)$ states, with properties of self-stabilization and leak-robustness, in the sense that interactions tend to faulty for the latter property. A problem with opposite purpose for the majority problem, called partition problem or diversity problem also considered in the series of the papers [62–64] and [6].

1.3.2 Fast Neighborhood Rendezvous

The solvability and complexity of the rendezvous problem is affected by many factors, such as synchrony, randomness of algorithms, graph classes, symmetry of agents, and so on. For that reason it is difficult to compare our results with past literature directly. Nevertheless, several results aim to achieve sublinear-time rendezvous explicitly or implicitly. Collins et al. [29] demonstrate that two agents with a common map (i.e., whole information of G), which are initially placed with distance d, can achieve rendezvous deterministically within $O(d\log^2 n)$ rounds, they also show a nearly tight

$\Omega(d \log n / \log \log n)$-round lower bound. Das et al. [36] assume that two agents can detect their distance, and present a deterministic rendezvous algorithm within $O(\Delta(d + \log l))$ rounds, where l is the minimum value of the IDs of agents. It is also proven that any algorithm requires $\Omega(\Delta(d + \log l / \log \Delta))$ rounds in this model. The result by Anderson et al. [11] is the closest to our result in the sense that it assumes no oracle such as maps and distance detection stated above. It considers the model of anonymous vertices with whiteboards, and presents a randomized algorithm that achieves rendezvous for complete graphs in $O(\sqrt{n})$ expected rounds. As we mentioned, the neighborhood rendezvous problem can be seen as a relaxation of rendezvous in complete graphs, and thus we can regard our result as the one extending the graph classes allowing fast rendezvous (using a stronger assumption of vertex identifiers). There are also several studies [39, 51, 52] for achieving fast rendezvous using side information coming from oracles (so-called *advice*). In this model, agents cannot see the whole map of G, but instead can know the (partial) information on their initial locations.

Due to the interest on hardness of symmetry breaking, the solvability of the rendezvous problem for ring networks has received much consideration in several different models [37, 42, 49]. In this context, the analysis of complexity has not received much attention. The study of rendezvous in trees has focused on time and space complexities. The paper by Baba et al. [17] presents a linear-time (equivalently, $O(n)$ time) algorithm under the assumption that agents have $O(n)$ bits of memory, and the authors also show its optimality with respect to space in the class of linear-time algorithms. Czyzowicz et al. [32] generalized this result, and presented an algorithm achieving rendezvous in $\Theta(n + n^2/k)$ rounds for agents having k bits of memory. Fraigniaud et al. [43] presents the rendezvous algorithm in trees with the optimal memory complexity ($\Theta(\log n)$ bits). The feasibility of rendezvous in general graphs are also considered in several papers [24, 31, 33, 38]. In paper [31], the memory requirement for the rendezvous of uniform agents is considered, which presents that $\Theta(\log n)$ bits are necessary and sufficient for two agents in any anonymous graph. Recently, Miller et al. [53] consider the trade-offs between time and cost (the number of edges traversed by agents).

The rendezvous problem allowing randomization is often considered as a part of the theory of random walks. The time taken for two tokens to meet at a common vertex is called the *meeting time* [25,60]. The rendezvous problem in the analyses of Markov chain theory is also considered in the context of operations research [1,8,11,34,61,65].

A comprehensive overview of the rendezvous problem can be found in the books by Alpern and Gal [10] and Alpern et al. [9], and several surveys [7,48,54].

1.4 Organization

This dissertation consists of four chapters. In Chapter 2, we deal with sublinear-time aggregation in probabilistic population protocol model, which is one of the promising model for the passively mobile agent system. In chapter 3, we consider the rendezvous problem in graphical autonomous mobile agent system, and show the minimal assumption for the sublinear solvability of the problem. We conclude this dissertation in Chapter 4.

Chapter 2

Sub-linear time Aggregation in Probabilistic Population Protocol Model

A passively mobile system is an abstract notion of mobile ad-hoc networks. It is a collection of agents with computing devices. Agents move in a region, but the algorithm cannot control their physical behavior (i.e., how they move). The population protocol model is one of the promising models in which the computation proceeds by the pairwise communication between two agents. The communicating agents update their states by a specified transition function (algorithm).

In this chapter, we consider a general form of the *aggregation* problem with a base station. The base station is a special agent having the computational power more powerful than others. In the aggregation problem, the base station has to sum up for inputs distributed to other agents. We propose an algorithm that solves the aggregation problem in sub-linear parallel time using a relatively small number of states per agent. More precisely, our algorithm solves the aggregation problem with input domain X in $O(\sqrt{n}\log^2 n)$ parallel time and $O(|X|^2)$ states per agent (except for the base station) with high probability.

This chapter organized as follows. In section 2.1 we give the definition of the model, notations, and several useful tools presented by other papers. In section 2.2 we present our protocol for aggregation.

2.1 Preliminaries

2.1.1 Population Protocol Model with Base Station

A *population protocol with a base station* consists of n anonymous *agents*, which contains one special agent called the *base station*. In what follows, the agents other than the base station are called *ordinary agents*. Each agent is defined as a bounded-space random access machine and updates its state by *interactions*. An interaction is a pairwise communication with other agents in the system. Note that an interacting pair of two agents is represented as an ordered pair. That is, they have the mechanism of symmetry breaking and thus even if two agents with the same states interact their resultant states can be different. For any interacting pair (r_1, r_2), we call the first agent r_1 the *sender* of that interaction, and call r_2 the *receiver*. A directed *interaction graph* $G = (V, E)$ of n nodes defines a capability of interactions. Each node in G corresponds to an agent, and each edge in G indicates the possibility of interactions between two endpoints. Throughout this chapter we assume that G is complete (i.e., any pair of agents can interact). For reference, we give each node an integer identifier $\{1, 2, \ldots, n\}$, but identifiers are not accessible to algorithms.

In this chapter, we assume that the agents know a common upper bound N for the number of agents n. More precisely, an algorithm is defined as an infinite sequence of concrete algorithms $P_1, P_2, \ldots, P_N, \ldots$. A concrete algorithm P_N with parameter N is a 6-tuple $(Q_N, \delta_N, I_N, O_N, \iota_N, \gamma_N)$, where Q_N is a finite set of the states, $\delta_N : Q_N \times Q_N \to Q_N \times Q_N$ is a transition function, I_N is a finite set of input symbols, O_N is a finite set of output symbols, $\iota_N : I_N \to Q_N$ is an input function, and $\gamma_N : Q_N \to O_N$ is an output function. The number of states in I_N, Q_N, and O_N can depend on N. Initially, each agent i has an input $x_i \in I_N$, which is converted to the initial state $\iota_N(x_i)$. When two agents interact with each other, they update their states according to the function δ_N. The output function γ_N decodes an output value from the current state of an agent. Since the parameter N implies the upper bound for the number of agents n. the algorithm P_N must work correctly when it is deployed to the system with at most N agents.

A *configuration* $C : \mathbb{Z}^+ \to Q$ of P_N with n agents $(N \geq n)$ is defined as a n-dimensional vector where each element $C[i]$ corresponds to the state of agent i. An execution is an infinite sequence of configurations $C_1, C_2, \ldots$ such that C_i is obtained from C_{i-1} by making some two agents interact with each other. The next interacting pair is determined by the probabilistic scheduler. Formally, for any algorithm P with transition function δ, let $\delta(C, j, k)$ be the configuration obtained from C by the interaction of pair (j, k). Letting $C_0, C_1, \ldots, C_i, \ldots$ be the execution of P, for any $i \geq 1$, C_{i+1} is chosen uniformly at random from the set $\{\delta(C_i, j, k) \mid (j, k) \in E\}$.

The i-th interaction in an execution is called the i-th *step* of the execution. In the literature of population protocols, the measurement of *parallel time* is often adopted. One parallel time is a consecutive n steps of the execution. That is, the length of a finite (sub)execution with m steps with respect to parallel time is m/n. For short, we use terminology "time" as the meaning of parallel time in the following argument.

2.1.2 Aggregation Problem

Let $(X, +)$ be an arbitrary commutative monoid whose identity element is zero (where + is not necessarily the standard arithmetic sum), and $\hat{X} = X \setminus \{0\}$. In the aggregation problem for $(X, +)$, each ordinary agent i initially has a value x_i, and the goal of the task is that the base station computes the value $\hat{x} = \sum_i x_i$. We assume that the base station is equipped with an output register storing a value in X. The value of the output register must be converged into $\hat{x}$ with high probability (the probability at least $1 - 1/n$) with respect to the distribution of possible executions. More precisely, an aggregation algorithm is correct if the execution of the algorithm has an infinite-length suffix where the output register of the base station always stores $\hat{x}$ with high probability. The computation time of an aggregation algorithm is defined as the parallel time taken until the convergence of the output register. Note that the requirement of commutativity naturally arises in the population protocol model because the anonymity of agents implies that any permutation of input values cannot affect the output.

2.1.3 Epidemics

The algorithm called *epidemics* (or propagation) is a simple subroutine used in many algorithms. The abstract structure of the epidemics is as follows: At first there is at least one agent with value v, which wishes to propagate v to all other agents, and other agents initially have value $\perp$. The transition rule is to change $(v, \perp)$ or $(\perp, v)$ into (v, v). The analysis by Angluin et al. [13] shows that under the random scheduler the epidemics algorithm finishes within $O(\log n)$ parallel time with high probability:

Theorem 1 ([13]) *At any step t_0, suppose there is at least one agent which wishes to start the epidemics algorithm. Then there exists a constant c such that the epidemics finishes at step $t_0 + cn \log n$ or earlier with probability at least* $1 - 1/n^3$.

2.1.4 Phase Clock

Specification The phase clock, introduced by Angluin et al. [13], allows the base station to count approximately $O(\log n)$ time. The phase clock tells the base station that $O(\log n)$ time has passed from the last activation, which is informed through a special one-bit variable named Clock-Pulse. The Clock-Pulse basically keeps value zero, and periodically activated (i.e., changes to value one). We call the activation of the bit a *signal* of the phase clock. The (value-one) signal is immediately deactivated to 0 at the interaction next to the activation. A configuration C_i in an execution of the phase clock is said to be in the k-th round when the base station already outputted k signals but does not activate the $(k+1)$-th. The algorithm proposed in [13] implements the phase clock mechanism only using $O(1)$ states per agent. We use it as a black box. Note that the extra cost (with respect to the number of states) is (asymptotically) negligible because it consumes a constant number of states.

Theorem 2 ([13]) *Let $t(i)$ be the step when i-th signal is activated, and c be the fixed constant from Theorem 1. Then there exists a a constant $d(c)(> 2c)$ such that with probability at least $1 - 1/n^3$ for all $i \in [1, n]$ $2cn \log n \leq |t(i+1) - t(i)| \leq d(c)n \log n$ holds.*

Simulation of Round-Based Synchrony Intuitively, in each interval of two consecutive signals (i.e., in one round), with high probability the system can complete two times of epidemics. It yields the simulation of *round-based* synchrony to the population protocol. In each round, the base station propagates the clock-pulse message as the first epidemics. When an agent receives the clock-pulse message, it can perform a message propagation by the second epidemics (specified by the algorithm). We can probabilistically guarantee that the message propagation always finishes within the round. In the following argument, the notification to each (ordinary) agent is also modeled by the variable named Clock-Pulse. We assume that our algorithm runs the round-based synchrony mechanism as an underlying process of our algorithm. That is, in our algorithm, each agent performs message exchange following the signal of the Clock-Pulse. For simplicity of the arguments, we also assume that the round-based synchrony always succeeds (at least until round n). The high success probability of the round-based synchrony is achieved by taking the union bound over the low failure probabilities of the phase clock and epidemics.

2.2 Fast Aggregation Algorithm

2.2.1 Outline

Consider the aggregation problem $(X, +)$ where each ordinary agent i initially has an input value x_i. A trivial algorithm for the aggregation problem $(X, +)$ is the *coalescence* algorithm defined by the transition rules $(a, b) \longrightarrow (a + b, 0)$ for any pair $(a, b) \in \hat{X}^2$. Running the coalescence algorithm over all agents, an agent obtains the value $\hat{x} = \sum_i x_i$ in $O(n)$ expected parallel time. When the base station interacts with the ordinary agent with a non-zero value k, it copies the value k to its output register. This algorithm guarantees that the output value of the base station eventually converges into $\hat{x}$. It is not difficult to see that $O(n)$-round upper bound for the running time of the coalescence algorithm is tight. The advantage of our algorithm achieves much faster convergence. More precisely, it finishes in $O(\sqrt{n}\log^2 n)$ parallel time with high probability.

The idea of our algorithm is very simple: The bottleneck of the coalescence algorithm lies in the situation where the number of ordinary agents with non-zero values becomes small. If only m ($m \ll n$) agents have non-zero values, an interaction selected by the scheduler makes no progress of the algorithm with probability $1 - \Theta((m/n)^2)$. That is, if $m = O(1)$, the system wastes $\Omega(n^2)$ interactions in expectation. Circumventing this situation, we utilize another mechanism called *sequential absorption* when only $O(\sqrt{n})$ agents have non-zero values. The sequential absorption first elects a unique ordinary agent having a non-zero value by spending $O(\log N)$ rounds. We call the elected agent an *absorption agent*. The absorption agent runs the epidemics for propagating its own value, which reaches the base station within one round. Repeating this procedure $\Theta(\sqrt{n})$ times, we can complete the aggregation. Since the election and epidemics take $O(\log^2 n)$ parallel time, the total running time of the sequential absorption algorithm is $O(\sqrt{n}\log^2 n)$. The remaining issue is to combine the sequential absorption mechanism with the coalescence algorithm. While the sequential composition is obviously correct, it requires the timer for (exactly or approximately) counting $\Theta(\sqrt{n})$ parallel time. To avoid consuming extra memory space, we choose fair composition, that is, simply running them concurrently. We refer ordinary agents with value zero as *zero agents*, and others as *non-zero agents*.

2.2.2 Election of an Absorption Agent

Before specifying the whole of the algorithm, we present an election algorithm used in the sequential absorption algorithm as a subroutine. While we can utilize previous fast leader election algorithms for this part, we newly introduce a space-efficient (($O(1)$

states per agent) algorithm with the help of the base station. The proposed algorithm elects an absorption agent in $O(\log n)$ rounds using random coin flipping. While the coin flip is not available in our model, it does not matter because we can create local random bits from the randomness of the scheduler. The simplest way of creating one random bit is that the sender and receiver of each interaction get values one and zero respectively as the result of the coin flip. Since the choice by the scheduler is uniformly at random, this mechanism obviously implements a perfect (i.e., unbiased) coin flipping. It should also be mentioned that another technique for generating longer random bits is also introduced in [2]. However, only one random bit is enough to implement our election algorithm. Since the space required for the random-bit generation is one, this mechanism does not affect the asymptotic complexity for the number of states used by each agent.

The algorithmic idea of our election algorithm is to eliminate a constant fraction of the candidates by coin flipping. The run of our algorithm consists of $O(\log n)$ rounds managed by the round-based synchrony. The algorithm starts with the epidemic of a signal from the base station. When a non-zero agent receives the signal, it becomes a candidate of a leader. At the beginning of each round, the surviving candidates flip a coin and get a random value. The agent getting value one is still alive and propagates the value-one information using epidemics. On the other hand, the agents getting value zero do nothing. They are killed when the propagated value-one information is received. Note that if there is no agent with value one in some round, the number of candidates does not decrease and thus that round is wasted. This mechanism is necessary for guaranteeing that at least one candidate survives. In expectation, it suffices to iterate the elimination process $O(\log n)$ times for electing a unique absorption agent. The number of iterations is counted by the base station.

In order to prevent candidate agents from receiving expired value-one information propagated in past rounds, the base station detects the termination of the propagation in each round. The termination detection simply follows the timeout mechanism. When the synchrony notifies the end of the round, the base station starts the propagation of the clean-up message in the next round. Any agent receiving the clean-up message stops the propagation of the value-one information. The termination of the clean-up process also follows the notification by the synchronization mechanism. Consequently, one elimination process is implemented using two rounds.

2.2.3 Sequential Absorption Algorithm

Utilizing the election algorithm shown in the previous section, we can construct the sequential-absorption side of our algorithm. As we explained its outline, one iteration of the sequential absorption consists of the following tasks:

1. Select an absorption agent by the election algorithm,

2. The absorption agent propagates its value using the epidemics algorithm.

3. At the end of the round when the absorption agent propagates the value (notified by the clock pulse of the round-based synchrony), all agents delete the message in propagation.

Note that the third task is to guarantee that no propagated value is left in the next iteration as an expired one. It should also be noted how the sequential absorption and the coalescence are composed. As we stated, it is simply achieved by running them in parallel. A point to be careful is that we have to avoid the value kept by an absorption agent is doubly accumulated in both mechanisms. This issue is easily resolved by excluding the absorption agent from the coalescence process. That is, when an agent i is elected as an absorption agent, the value stored in the coalescence side of i is reset to zero.

2.2.4 Algorithm Details

The transition rules of ordinary agents and the base station are presented in Algorithm 1. We explain its details below.

Variables

The state of each agent (including the base station) is a 5-tuple (v, buf, exec, rand, BST). We denote the variables of agent i by v_i, buf_i, exec_i, rand_i, and BST_i) respectively. The role of the variables is summarized as follows.

- The variable v stores a partial sum of input values in the coalescence algorithm. The initial value of the variable v_i is x_i.

- The variable buf is a buffer to store the value propagated by an absorption agent or the value of a random bit.

- The variable exec represents the current mode in the run of the sequential absorption algorithm, which stores one of five values E (electing), $\mathrm{C_E}$ (cleanup of electing) $\mathrm{S_1}, \mathrm{S_2}$ (spreading), and $\mathrm{C_S}$ (cleanup of spreading).

- The variable rand stores the random bit used in the election algorithm. For simplicity, we also use this variable for indicating whether agent i is currently joining the election process or not. If i does not join the process (i.e., it is a zero agent or a non-zero but already killed), rand_i stores $\perp$.

- The variable BST is the fixed one-bit flag which stores one only at the base station. The ordinary agents have zero.

These variables are initially set to $\mathrm{v}_i = x_i$, $\mathrm{buf}_i = 0$, exec = S, rand = 0 for each ordinary agent i. The base station has initial values as v = 0, buf = 0, exec = S, rand = 0. These initial values enable the base station to be regarded as a zero agent, thus the variables of the base station do not affect the computation. The space requirement for each agent is summarized as follows: Variables exec_i and rand_i use $O(1)$ states, and variables v_i and buf_i use $O(|X|)$ states respectively. The total space complexity for ordinary agents is $O(|X|^2)$ states.

The base station also has additional variables m, round and coalesced for the special operations of the base station.

- The variable $m = \log N$ is the upper bound for the value of $\log n$. Since m is only used for detecting the termination of the election algorithm, the gap between N and n does not affect the correctness of the algorithm.

- The variable round counts the number of rounds passed in the E (electing) mode.

- The variable coalesced is a flag indicating whether the base station has already accumulated the propagated value of an absorption agent in the mode of S_2.

Transition Rules of Coalescence Algorithm

In this algorithm, all the agents naively aggregate non-zero values of v by pairwise interactions between non-zero agents. The algorithm consists of two rules: If two non-zero agents interact then one of the agents gets the coalesced value and the other agent becomes a zero agent (lines 3-4). Another rule specifies operations between the base station and one of the non-zero agents. If the base station and a non-zero agent interact, then the base station aggregates the values, and the ordinary agent becomes zero agent (lines 5-6).

Transition rules of Sequential Absorption Algorithm

As mentioned in subsection 2.2.3, this algorithm iteratively accumulates the value held by a non-zero agent to the base station. The algorithm runs with alternately switching two subalgorithms called *election* and *spreading* respectively. In the election, agents elect an absorption agent from non-zero agents using the election algorithm stated in subsection 2.2.2. In the spreading, the elected absorption agent propagates its value of v, and the base station aggregates it. The mode of each agent is determined by the value of exec. More precisely, an agent with exec $\in \{\mathrm{E}, \mathrm{C_E}\}$ runs the election, and one with exec $\in \{\mathrm{S_1}, \mathrm{S_2}, \mathrm{C_S}\}$ runs the spreading. The run of each mode is globally synchronized by the round-based synchrony. The election subalgorithm consists of $O(\log n)$ rounds, and that of the spreading subalgorithm consists of two rounds. The details of the two modes are explained below.

Election Algorithm In the election algorithm, we divide ordinary agents into four types: zero agents satisfying $\mathrm{v} = 0$, *killed agents* satisfying $\mathrm{v} \neq 0$ and rand $=\perp$, *weak* candidates satisfying $\mathrm{v} \neq 0$ and rand $= 0$, and *strong* candidates satisfying $\mathrm{v} \neq 0$ and rand $= 1$. The run of this subalgorithm starts when agents with mode C_S (i.e., in the previous spreading) receives the signal of Clock-Pulse. By that trigger, non-zero agents changes the mode to E, and flip random coins. If the value of rand is one, it also writes the value one to buf (lines 22-25) for informing the existence of strong candidates (i.e., the value-one information). The propagation of the value-one information is equivalant to the epidemics for variable buf (line 8,9). Also, if an agent i is a weak candidate and gets the value-one information, it becomes a killed agent (line 11). Note that even killed agents support the propagation. Agents proceed to the next round when the activations of the Clock-Pulse occurs again. Then agents change its mode to C_E (line 26). Agents utilize this mode for cleaning variables buf. Each agent i resets buf_i to zero (line 27), and waits until the next trigger of the phase clock is activated. At the activation, the agent returns to the mode E. In addition to the operations above, the base station executes the special operation of incrementing the variable round and checking if round exceeds the bound $96m = O(\log n)$ (line 36-37, 44-45). If it exceeds the bound, the base station initiates the subalgorithm spreading. Otherwise, it continues to the election with changing the state to E (lines 38,39, 46, and 47).

Spreading Algorithm The Spreading subalgorithm starts when the base station enters this mode (i.e., exec $= \mathrm{S_1}$). The base station propagates the termination signal (of the election subalgorithm) in the first round. In this propagation, an ordinary agent receiving that signal changes its mode to S_1 (line 16). At the interaction of the mode change, the agent sets up variables buf and rand according to its type in the election algorithm. Any zero agent or killed agent resets buf to zero (line 16). If the agent is still a candidate (regardless of weak or strong), it becomes an absorption agent. That is, it stores the value of v to buf, and becomes a zero agent by setting v = 0 and rand $=\perp$ (line 18). Then, all agents wait for the next triggers of the phase clock. In the next round, they proceed to the mode S_2 and conduct the propagation of the value of the absorption agent (line 20). Agents except for the absorption agent overwrite buf to the value of the absorption agent (lines 13-14). When the value reaches the base station, the aggregation finishes (lines 33,34, 41, and 42). In order to avoid multiple accumulation, the base station set a flag coalesced to one and ignores values of buf afterward (lines 34 and 42). This flag is reset when the base station enters the next iteration (lines 39 and 47). After the propagation, according to the trigger of the phase clock, they go back to the mode C_S and reset buf to zero (line 21).

2.2.5 Correctness Proof

Lemma 3 *Let r be the round where* round $= 0$ *holds at the base station. Then at round $r + 96 \log N$, with probability at least $1 - 1/n^3$, the system has exactly one agent i such that* $\mathrm{rand}_i \neq \perp$.

Proof 1 *By the structure of the algorithm, at least one agent survives the election process because if all surviving agents take random value zero, no agent is further killed in that round. Hence it suffices to show that the number of candidates decreases quickly. More precisely, we show that a constant fraction of surviving agents are killed with a constant probability at each iteration of the elimination process.*

Assume that at least $l \geq 2$ agents still survive as candidates at the beginning of a round. We first handles the case of $2 \leq l \leq 4$ exceptionally. In this case, with protability $1/4$ or more, at least $\lceil l/2 \rceil$ candidates get random value 0 and at least one candidate gets value one. Consider the case of $l > 4$. Let X_b be the number of candidates getting the value $b \in \{0, 1\}$. It is obvious that $\mathbb{E}[X_1] = l/2$ holds. By Markov's inequality, we have

$$\Pr[X_1 \geq (3l)/4] \leq 2/3$$
$$\Rightarrow \Pr[X_0 \geq l/4] \geq 1/3.$$

Hence with probability at least $1/4$, a quarter of all surviving agents are killed in each elimination process. We call an elimination process good *if it successfully kills a $1/4$ fraction of candidates. Let Y be the number of good elimination processes* *during* $96 \log N$ *times of iterations. The expectation* $\mathbb{E}[Y]$ *is* $24 \log N$. *By Chernoff bound, the probability that* $Y \leq 12 \log n$ *holds is bounded by*

$$\Pr[Y \leq 12 \log n] \leq \mathrm{e}^{3 \log n} \leq 1/n^3.$$

It follows that the algorithm elects a unique agent after $r + 96 \log N$ *rounds with probability at least* $1 - 1/n^3$ *because the number of candidates at the first round is at most* N. *and each iteration consists of exactly two rounds.* □

Lemma 4 *The output register of the base station eventually stores* $\hat{x} = \sum_i^n x_i$ *with probability at least* $1 - O(1/n^2)$.

Proof 2 *By Theorem 3, the election of an absorption agent succeeds with probability at least* $1 - O(1/n^3)$. *Since one election increases the number of zero agents by one, the election process is activated at most n times. By taking the union bound, we can show that all the election processes succeed with probability at least* $1 - O(1/n^2)$. *Then, we can see that the value of the output register converges to* $\hat{x}$ *by the following observation: At any configuration* C, *the value of* buf_i *for each agent* i *is either* 0 *or the value in propagation (which is common among all agents). Let* $b(C)$ *be the non-zero value stored in* buf *in* C. *If all buffers store zero, we define* $b(C) = 0$. *Then we obtain the following invariant:* $\sum_j^{n-1} v_j + b(C) + out = \hat{x}$. *It is easy to check that this equation is actually an invariant (that is, no interaction breaks it). It obviously follows that the output of the base station is equals to* $\hat{x}$ *when any ordinary agent becomes a zero agent.* □

Lemma 5 *After* $O(\sqrt{n} \log^2 n)$ *parallel time, each ordinary agent* i *satisfies* $v_i = 0$ *with high probability.*

Proof 3 *We first show that the coalescence algorithm decreases the number of non-zero agents to* $\sqrt{n}$ *after* $O(\sqrt{n} \log n)$ *parallel time with high probability. Suppose that there exists k non-zero agents in the system. Then, the probability that any two non-zero agents interact with each other is* $k(k-1)/n(n-1)$, *and the expected time taken to decrease the number of non-zero agents by one is its inverse* $n(n-1)/k(k-1)$. *Thus the expected number of steps taken until the number of non-zero agents becomes less than* $\sqrt{n}$ *is*

$$\begin{aligned} \sum_{k=\lfloor \sqrt{n} \rfloor}^{n} \frac{n(n-1)}{k(k-1)} &= n(n-1) \sum_{k=\lceil \sqrt{n} \rceil}^{n} \left(\frac{1}{k-1} - \frac{1}{k} \right) \\ &\leq 2n^{1.5}. \end{aligned}$$

Furthermore, by Markov's inequality, after $4n^{1.5}$ *interactions, the number of non-zero agents becomes less than* $\sqrt{n}$ *with probability at least* $1/2$*. Consequently, the probability that the number of non-zero agents is still more than* $\sqrt{n}$ *after* $8n^{1.5}\log n$ *interactions is at most* $(1-1/2)^{2\log n} \leq 1/n^2$.

Next, we bound the time taken to eliminate $\sqrt{n}$ *non-zero agents. Since each iteration of the sequential absorption takes* $\Theta(\log^2 n)$ *parallel time and decreases the number of non-zero agents by one with high probability,* $O(\sqrt{n}\log^2 n)$ *parallel time suffices to eliminate all non-zero agents. Putting the two case analyses together, we have the lemma.* □

Theorem 6 *The proposed algorithm solves the aggregation problem with high probability. Its running time is* $O(\sqrt{n}\log^2 n)$ *and the space complexity of ordinary agents is* $O(|X|^2)$ *states.*

Proof 4 *By Lemma 4, the algorithm aggregates all the inputs with high probability and by Lemma 5 the algorithm converges in* $O(\sqrt{n}\log^2 n)$ *time with high probability. We focus on the space complexity of the algorithm. Ordinary agents have variables* $(\mathrm{v}, \mathrm{buf}, \mathrm{exec}, \mathrm{rand})$. *The variables* exec *and* rand *use only* $O(1)$ *states respectively. Since the variables* v *and* buf *are used for storing the input and coalesced inputs, each of the variables uses* $O(|X|)$ *states. Hence the space complexity of the algorithm is* $O(|X|^2)$ *states.* □

Algorithm 1 Fast Aggregation Algorithm(1/2)

Variables

5-tuples $(\mathrm{v}_i, \mathrm{buf}_i, \mathrm{exec}_i, \mathrm{rand}_i, \mathrm{BST}_i)$

Variables of the base station

m: the upper bound of $\log n$

round: to count the round in electing mode

coalesced: a flag that determines whether a value of an absorption agent is coalesced in spreading mode

out: the output register

Auxiliary procedures and signals

Clock-Pulse: of phase clock

coin-flip(): a function that output $b \in \{0, 1\}$ with probability $1/2$ for each value

Algorithm

```
1: The transitions when an ordinary agent $i$ interacts with an agent $j$
2: // rules for the coalescence algorithm
3:   if $\mathrm{BST}_i = 0 \wedge \mathrm{BST}_j = 0 \wedge \mathrm{v}_i > 0 \wedge \mathrm{v}_j > 0$ then
4:     if agent $i$ is sender then $\mathrm{v}_i \leftarrow 0$; else $\mathrm{v}_i \leftarrow \mathrm{v}_i + \mathrm{v}_j$;
5:   if $\mathrm{BST}_i = 1 \wedge \mathrm{BST}_j = 0 \wedge \mathrm{v}_j > 0$ then
6:     $out \leftarrow out + \mathrm{v}_j$;  $\mathrm{v}_j \leftarrow 0$;
7: // rules for the sequential absorption algorithm
8:   if $\mathrm{exec}_i = E \wedge \mathrm{exec}_j = E \wedge \mathrm{buf}_i = 0$ then
9:     $\mathrm{buf}_i \leftarrow \mathrm{buf}_j$;   // propagation of random bit
10:    // changing zero-agent
11:    if $\mathrm{buf}_i = 1 \wedge \mathrm{rand}_i = 0$ then $\mathrm{rand}_i \leftarrow \perp$;
12:  // propagation of a value of an absorption agent
13:  if $\mathrm{exec}_i = \mathrm{S}_2 \wedge \mathrm{exec}_j = \mathrm{S}_2 \wedge \mathrm{buf}_i = 0 \wedge \mathrm{buf}_j \neq 0$ then
14:    $\mathrm{buf}_i \leftarrow \mathrm{buf}_j$;
15:  // shifts to the S mode
16:  if $\mathrm{exec}_i = \mathrm{E} \wedge \mathrm{exec}_j = \mathrm{S}_1$ then $\mathrm{exec}_i = \mathrm{S}_1$; $\mathrm{buf}_i \leftarrow 0$;
17:    // changing to the absorption agent
18:    if $\mathrm{v}_i > 0 \wedge \mathrm{rand}_i \neq \perp$ then $\mathrm{buf}_i \leftarrow \mathrm{v}_i$; $\mathrm{v}_i \leftarrow 0$; $\mathrm{rand}_i \leftarrow \perp$;
19:  if Clock-Pulse is triggered then
20:    if $\mathrm{exec}_i = \mathrm{S}_1$ then $\mathrm{exec}_i \leftarrow \mathrm{S}_2$;
21:    if $\mathrm{exec}_i = \mathrm{S}_2$ then $\mathrm{buf}_i \leftarrow 0$; $\mathrm{exec}_i \leftarrow \mathrm{C_S}$;
22:    if $\mathrm{exec}_i = \mathrm{C_S}$ then $\mathrm{exec}_i \leftarrow \mathrm{E}$;
23:      // changing to a candidate
24:      if $\mathrm{v}_i > 0$ then $\mathrm{rand}_i \leftarrow$ coin-flip();
25:        if $\mathrm{rand}_i = 1$ then $\mathrm{buf}_i \leftarrow 1$;
26:    if $\mathrm{exec}_i = \mathrm{E}$ then
27:      $\mathrm{buf}_i \leftarrow 0$; $\mathrm{exec}_i \leftarrow \mathrm{C_E}$;
28:    if $\mathrm{exec}_i = \mathrm{C_E}$ then
29:      if $\mathrm{v}_i > 0 \wedge \mathrm{rand}_i \neq \perp$ then $\mathrm{rand}_i \leftarrow$ coin-flip();
30:        if $\mathrm{rand}_i = 1$ then $\mathrm{buf}_i \leftarrow 1$;
```

Algorithm 2 Fast Aggregation Algorithm(2/2)

```
// rules for the base station
  if $\text{BST}_i = 1$ then
    if $\text{exec}_i = \text{S}_2 \wedge \text{exec}_j = \text{S}_2 \wedge \text{buf}_j \neq 0 \wedge \text{coalesced} = 0$ then
      $out \leftarrow out + \text{buf}_j$; coalesced $\leftarrow 1$;
    if Clock-Pulse is triggered then
      if $\text{exec}_i = E$ then round$++$; $\text{exec}_i \leftarrow \text{C}_\text{E}$;
      if $\text{exec}_i = \text{C}_\text{E}$ then $\text{exec}_i \leftarrow \text{E}$;
        if round $\geq 96m$ then
          $\text{exec}_i \leftarrow \text{S}_1$; round $\leftarrow 0$; coalesced $\leftarrow 0$;
  if $\text{BST}_j = 1$ then
    if $\text{exec}_j = \text{S}_2 \wedge \text{exec}_i = \text{S}_2 \wedge \text{buf}_i \neq 0 \wedge \text{coalesced} = 0$ then
      $out \leftarrow out + \text{buf}_i$; coalesced $\leftarrow 1$;
    if Clock-Pulse is triggered then
      if $\text{exec}_j = E$ then round$++$; $\text{exec}_j \leftarrow \text{C}_\text{E}$;
      if $\text{exec}_j = \text{C}_\text{E}$ then $\text{exec}_j \leftarrow \text{E}$;
        if round $\geq 96m$ then
          $\text{exec}_j \leftarrow \text{S}_1$; round $\leftarrow 0$; coalesced $\leftarrow 0$;
```

Chapter 3

Fast Neighborhood Rendezvous

In the rendezvous problem, two computing entities (called *agents*) located at different vertices in a graph have to meet at the same vertex. In this chapter, we consider the synchronous *neighborhood rendezvous problem*, where the agents are initially located at two adjacent vertices. While this problem can be trivially solved in $O(\Delta)$ rounds (Δ is the maximum degree of the graph), it is highly challenging to reveal whether that problem can be solved in $o(\Delta)$ rounds, even assuming the rich computational capability of agents. The only known result is that the time complexity of $O(\sqrt{n})$ rounds is achievable if the graph is complete and agents are probabilistic, asymmetric, and can use whiteboards placed at vertices. Our main contribution is to clarify the situation (with respect to computational models and graph classes) admitting such a sublinear-time rendezvous algorithm. More precisely, we present two algorithms achieving fast rendezvous additionally assuming bounded minimum degree, unique vertex identifier, accessibility to neighborhood IDs, and randomization. The first algorithm runs within $\tilde{O}(\sqrt{n\Delta/\delta} + n/\delta)$ rounds for graphs of the minimum degree larger than $\sqrt{n}$, where n is the number of vertices in the graph, and δ is the minimum degree of the graph. The second algorithm assumes that the largest vertex ID is $O(n)$, and achieves $\tilde{O}\left(\frac{n}{\sqrt{\delta}}\right)$-round time complexity without using whiteboards. These algorithms attain $o(\Delta)$-round complexity in the case of $\delta = \omega(\sqrt{n}\log n)$ and $\delta = \omega(n^{2/3}\log^{4/3} n)$ respectively. We also prove that four unconventional assumptions of our algorithm, bounded minimum degree, accessibility to neighborhood IDs, initial distance one, and randomization are all inherently necessary for attaining fast rendezvous. That is, one can obtain the $\Omega(n)$-round lower bound if either one of them is removed.

3.1 Preliminaries

3.1.1 Model and Notations

In this chapter, we consider the rendezvous problem of two agents in any undirected graph $G = (V, E)$ of n vertices. Each vertex in G has a distinct integer identifier in $[0, n' - 1]$, where n' satisfies $n' \geq n$ and $n' = n^{O(1)}$. The value of n' is available to each agent. We denote the identifiers of n vertices by $v_0, v_1, \dots, v_{n-1}$. The minimum and maximum degrees of G are respectively denoted by δ_G and Δ_G. For any vertex v, $N_G(v)$ represents the set of vertices adjacent to v, i.e., $N_G(v) = \{v' \mid (v, v') \in E\}$. We define $N^+_G(v) = N_G(v) \cup \{v\}$, and also define $N_G(X) = \bigcup_{v \in X} N_G(v)$ and $N^+_G(X) = N_G(X) \cup X$ for any vertex set $X \subseteq V$. We often omit subscript G if it is clear from the context.

In the system, two computing entities, called *agents*, are placed at two vertices in G, which are modeled as probabilistic random access machines. The two agents have distinct names denoted by a and b respectively, and can exhibit asymmetric behavior in executions, that is, they can run two different algorithms. Agents are equipped with memory space as their internal states. While we do not assess any assumption on time/space complexity for internal computation of agents, our proposed algorithms terminate within polynomial time, and use $O(n \log n)$-bit memory. We denote by $M \subseteq \{0, 1\}^*$ the set of possible internal states of two agents. When two agents visit the same vertex, they are aware of the presence of each other. On neighborhood knowledge, we define the *local port numbering* of each vertex v_i, which is a bijective function $\hat{P}_{v_i} : [0, |N(v_i)| - 1] \to N(v_i)$. We also define the *accessible local port number* $P_{v_i} : [0, |N(v_i)| - 1] \to \mathbb{N}$. Agents can see only P_{v_i} and have no access to $\hat{P}_{v_i}$. The model supporting the access to neighborhood IDs is defined as the assumption that $\hat{P}_{v_i}$ and P_{v_i} are the same function for any $v_i \in V$. On the lower-bound side, we also consider the case where each agent has no access to its neighborhood IDs. It is defined as the model such that P_{v_i} for any v_i is the identity mapping from $[0, |N(v_i)| - 1]$ to $[0, |N(v_i)| - 1]$ (i.e., it does not provide any information of $\hat{P}_{v_i}$).

Each vertex is equipped with a memory space called *whiteboards*, and an agent at vertex v can access/write to the whiteboard of v in its internal computation. Formally, we define $W \subseteq \{0, 1\}^*$ to be the set of possible contents written in each whiteboard. A state of all the whiteboards in G is represented by an n-dimensional vector W^n indexed by elements in V. While we have no assumption on the size of each whiteboard, $O(\log n)$ bits per vertex suffice for our algorithms.

Executions of two agents follow synchronous and discrete time steps $t = 0, 1, 2, \dots$ called *rounds*. In every round, an agent at vertex v either stays at the present location or moves to one of its neighbors. An algorithm $\mathcal{A}$ determines which action to take based on the information stored in its internal memory, IDs in $N^+(v)$ through the access to P_v, and the contents of the whiteboard at v. We assume that a movement to a neighbor necessarily completes within the current round. In other words, we do not consider the situation where agents are located on edges at the beginning of each round. At each round, agents can modify the whiteboards of their current vertices[*]. Formally, an algorithm is a function $\mathcal{A} : \{a, b\} \times M \times V \times 2^{\mathbb{N}} \times W \times \{0,1\}^* \to M \times \mathbb{N} \times W$. The inputs respectively correspond to the ID of the agent, its internal memory, the IDs of its current location and neighbors (with respect to accessible port numbering functions), the content of the whiteboard at the current location, and random bits. The outputs correspond to the internal state of the agent after the computation, the destination in the following movement (with respect to accessible local port numbers), and the content of the whiteboard left at the current vertex. Note that deterministic algorithms (only used in Section 3.4.4) are defined as the ones such that its behavior is independent of random bits. A *configuration* C at round t is a tuple in $C \in (V \times M)^2 \times W^n$. An *execution* is an infinite sequence of configurations $C_0, C_1, C_2, \ldots$. Precisely, letting v_i^z be the location of agent $z \in \{a, b\}$ at round i, m_i^z be the internal memory of agent z at round i, and w_i^j be the content of the whiteboard of vertex v_j at round i, a configuration C_i is described as $C_i = (v_i^a, m_i^a, v_i^b, m_i^b, w_i^0, \ldots, w_i^{n-1})$. For any $i \in \mathbb{N}$, every execution must satisfy the following conditions: For any $j \in V \setminus \{v_i^a, v_i^b\}$ $w_i^j = w_{i+1}^j$ holds. For each i, there exists $B_i^a, B_i^b \in \{0,1\}^*$ such that $\mathcal{A}(a, m_i^a, v_i^a, P_{v_i^a}, w_{v_i^a}, B_i^a) = (m_{i+1}^a, \hat{P}_{v_i^a}^{-1}(v_{i+1}^a), w_{v_i^a}^i)$ and $\mathcal{A}(b, m_i^b, v_i^b, P_{v_i^a}, w_i^{v_i^b}, B_i^b) = (m_{i+1}^b, \hat{P}_{v_i^b}^{-1}(v_{i+1}^b), w_i^{v_i^b})$ hold, where $P_{v_i^a}^{-1}$ and $P_{v_i^b}^{-1}$ are the inverse mappings of $P_{v_i^a}$ and $P_{v_i^b}$ respectively.

3.1.2 Rendezvous Problem

In the rendezvous problem, two agents initially located at two different vertices are required to visit the same vertex simultaneously and halt. Formally, the rendezvous problem is as follows.

Definition 1 *We say that an algorithm completes rendezvous at round t if the two agents are located at the same vertex at the beginning of that round.*[†]

[*]Strictly, we need to define formally the behavior of agents when they are located at the same vertex and attempt to modify the (common) whiteboard. In the rendezvous problem of two agents, however, such a case can be seen as the completion of the algorithm without loss of generality. Thus, we do not care about simultaneous and parallel write operation for the same whiteboard.

[†]In the synchronous system, we can assume that once two agents meet at a vertex then they halt without loss of generality. That is, agents that complete rendezvous at round t also complete

This chapter considers the rendezvous problem with the constraint on initial locations of agents and graph parameters.

Definition 2 (Specific Rendezvous) *For graph $G = (V, E)$, let $I \subseteq V \times V$ be a possible set of initial locations (v_0^a, v_0^b) of two agents. We say that an algorithm $\mathcal{A}$ solves the rendezvous problem for an instance (G, I) with probability p within t rounds, if for any $(v_0^a, v_0^b) \in I$, the execution of $\mathcal{A}$ in G completes rendezvous at round t with probability p. Moreover, letting $\mathcal{I} = \{(G_0, I_0), (G_1, I_1), \dots\}$ be a (possibly infinite) class of instances, we say that an algorithm $\mathcal{A}$ solves the rendezvous problem for class $\mathcal{I}$ with probability p within $f(n)$ rounds for some non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ if for every instance $((V, E), I) \in \mathcal{I}$, algorithm $\mathcal{A}$ solves the rendezvous problem with probability p within $f(|V|)$ rounds.*

In this chapter we are interested in the case where the distance between two initial locations of agents is upper bounded by d. For any graph G we define $I_d^G = \{(v, v') \mid dist_G(v, v') \leq d\}$, where $dist_G(v, v')$ represents the (hop-)distance of vertices v and v' in G. In addition, we also define the class $\mathcal{G}(\hat{\Delta}(n), \hat{\delta}(n))$ for functions $\hat{\delta} : \mathbb{N} \to \mathbb{N}$, $\hat{\Delta} : \mathbb{N} \to \mathbb{N}$ as the set of graphs $G = (V, E)$ such that $\delta_G \geq \hat{\delta}(|V|)$ and $\Delta_G \leq \hat{\Delta}(|V|)$ hold. The $(\hat{\Delta}, \hat{\delta}, d)$*-rendezvous problem* is defined as that for the instance class $\mathcal{I}_d = \{(G, I_d^G) \mid G \in \mathcal{G}(\hat{\Delta}(n), \hat{\delta}(n))\}$. In particular, we focus on the instance class $\mathcal{I}_1$ in Sections 3.2 and 3.3. In Section 3.4 we show the lower bounds on the problem for $\mathcal{I}_2$.

3.2 Rendezvous Algorithm

3.2.1 Algorithm Overview

In this section, we present an overview of our rendezvous algorithm. For ease of presentation, we assume that each agent has the precise values of δ and $\log n$, but it is not essential. Those values can be replaced with their constant-factor approximate values without increasing the asymptotic running time. A constant factor approximation of $\log n$ can be estimated from the upper bound n' of vertex IDs. The approximation of δ can be obtained by standard doubling estimation, explained in Section 3.3.

First, we introduce several definitions and terminologies used in the following argument.

Definition 3 (α-heaviness, α-lightness) *For any $T \subseteq V$, $v \in V$, and $\alpha \in \mathbb{R}^+$, v is*

rendezvous at any round $t' > t$.

*called α-*heavy *for T if $|T \cap N^+(v)| \geq \alpha$ holds* [‡]*. Similarly we say that v is α-*light *for T if $|T \cap N^+(v)| < \alpha$ holds.*

The following proposition is a trivial fact deduced from the definition above.

Proposition 1 *Let $v \in V$ be an α-heavy vertex for $T \subseteq V$. For any T' such that $T' \supseteq T$ holds, v is also α-heavy for T'.*

Given a vertex set $T \subseteq V$ and $\alpha \in \mathbb{R}^+$, we define $H_\alpha(T), L_\alpha(T) \subseteq V$ as the sets of vertices that are respectively α-heavy and α-light for T.

Definition 4 ((z, α, β)-dense condition) *Given $z \in \{a, b\}$, $T \subseteq V$, and $\alpha, \beta \in \mathbb{R}^+$, T is called (z, α, β)-*dense *if the following three conditions hold:*

- *$v_0^z \in T$,*
- *for any $w \in T$, $dist_G(v_0^z, w) \leq \beta$, and*
- *$N^+(v_0^z) \subseteq H_\alpha(T)$.*

The main idea of our rendezvous algorithm is that agent a constructs an $(a, \delta/8, 2)$-dense vertex set T^a. Since $v_0^b \in N^+(v_0^a) \subseteq H_{\delta/8}(T^a)$, v_0^b is an $(\delta/8)$-heavy vertex for T^a. Then a sublinear number of random vertex samplings from T^a by agent a and those from $N(v_0^b)$ by b ensure that a vertex is commonly sampled with high probability. In this sampling process, agent b leaves the ID of v_0^b at the whiteboards of all the sampled vertices. When agent a visits the common sample, it knows the initial location of v_0^b. Then agent a moves to v_0^b and meets b.

In the following argument, we divide our algorithm into two sub-algorithms. The first one, called Main-Rendezvous, achieves rendezvous provided that agent a knows an $(a, \delta/8, 2)$-dense set $T^a \subseteq N^+(N^+(v_0^a))$. The second sub-algorithm is for agent a to construct such an $(a, \delta/8, 2)$-dense set T^a, which is called Construct. The combination of these two sub-algorithms yields the algorithm we claim.

3.2.2 Rendezvous with T^a

We present the algorithm Main-Rendezvous, which solves the rendezvous problem using the initial knowledge of an $(a, \delta/8, 2)$-dense set $T^a \subseteq N^+(N^+(v_0^a))$ by agent a. Here the "knowledge" implies that (1) a has the list of all vertices in T^a in its memory,

[‡] $\mathbb{R}^+$ is the set of all positive real values.

Algorithm 3 Main-Rendezvous : Rendezvous with T^a

```
$w(v)$ : whiteboard at vertex $v$. Initially $w(v) = \perp$ for all $v \in V$
$q_a, q_b$ : local variables of agents $a$ and $b$
Operations of Agent $a$
1: construct $T^a$ satisfying $(a, \delta/8, 2)$-dense condition
2: repeat
3:     choose $v$ in $T^a$ uniformly at random, and move to $v$
4:     $q_a \leftarrow w(v)$
5:     return to $v_0^a$
6: until $q_a \neq \perp$
7: visit $q_a$ and halt
Operations of agent $b$
1: repeat
2:     move to $v \in N^+(v_0^b)$ chosen uniformly at random
3:     $w(v) \leftarrow v_0^b$
4:     return to $v_0^b$
5: until achieve rendezvous
```

and (2) also has the shortest paths to all vertices in T^a from a's initial location[§]. The pseudocode of Main-Rendezvous is presented in Algorithm 3. First, agent a samples a vertex v in T^a uniformly at random, and visits there. At vertex v, a checks if b has written the ID v_0^b in the whiteboard of v. If so, then a moves to v_0^b and halts. The agent b iteratively visits a vertex u in $N^+(v_0^b)$ chosen uniformly at random, and writes down the ID of v_0^b into the whiteboard of u. If it meets a at vertex v_0^b, then the algorithm terminates. We present the following lemma for the correctness of Main-Rendezvous.

Lemma 7 *Let $G = (V, E)$ be any graph such that $\delta_G \geq \sqrt{n}$ holds. Suppose that agent a constructs an $(a, \delta/8, 2)$-dense set T^a in t_a rounds. Then, Algorithm Main-Rendezvous completes rendezvous within $t_a + O\left(\sqrt{\frac{n\Delta}{\delta}} \log n\right)$ rounds with high probability.*

Proof 5 *We say that a vertex $v \in N^+(v_0^b) \cap T^a$ is* informed *at round t if $w(v) = v_0^b$ at t, and define $Z_t \subseteq N^+(v_0^b) \cap T^a$ as the set of all informed vertices at t. Let $h = \lfloor (1/16)\sqrt{n\delta/\Delta} \rfloor$ for short. We first show that $|Z_t| \geq h$ holds for $t \geq t_a + 8\sqrt{n\Delta/\delta}\log n$.*

[§]Since the length of these shortest paths are at most two by the definition of $(a, \delta/8, 2)$-dense sets, the space for storing this information is asymptotically same as the space for the list of vertices.

Let t_i be the first time that $Z_{t_i} \geq i$ holds, and X_i be $X_i = t_i - t_{i-1}$ $(1 \leq i \leq h)$. By the assumption of $\delta > \sqrt{n}$, we have the following inequality.

$$\begin{aligned} h = \left\lfloor \frac{1}{16}\sqrt{\frac{n\delta}{\Delta}} \right\rfloor \leq \frac{1}{16}\sqrt{n} < \frac{1}{16}\delta \\ < |N^+(v_0^b) \cap T^a|. \end{aligned}$$

For any $1 \leq i \leq h$, the variable X_i follows the geometric distribution with success probability $p_i = (|N^+(v_0^b) \cap T^a| - i + 1)/|N^+(v_0^b)|$. Then we have

$$\begin{aligned} \mathbb{E}[X_i] &= \frac{|N^+(v_0^b)|}{|N^+(v_0^b) \cap T_a| - i + 1} \\ &\leq \frac{|N^+(v_0^b)|}{|N^+(v_0^b) \cap T^a| - h + 1}. \end{aligned}$$

This deduces the following bound.

$$\begin{aligned} \mathrm{E}\left[t_h\right] = t_a + \mathrm{E}\left[\sum_{i=2}^{\lfloor h \rfloor} X_i\right] &\leq t_a + \sum_{i=2}^{h} \frac{|N^+(v_0^b)|}{|N^+(v_0^b) \cap T^a| - h} \\ &\leq t_a + h\frac{(\Delta+1)}{\delta/16} \\ &\leq t_a + \left\lfloor \frac{1}{16}\sqrt{\frac{n\delta}{\Delta}} \right\rfloor \frac{16(\Delta+1)}{\delta} \\ &\leq t_a + 2\sqrt{\frac{n\Delta}{\delta}}. \end{aligned}$$

By Markov's inequality, the probability of $|Z_t| < h$ for $t = t_a + 4\sqrt{n\Delta/\delta}$ is at most $1/2$. *Thus the probability of $|Z_t| < h$ for $t = t_a + 8\sqrt{n\Delta/\delta}\log n$ is at most* $1/n^2$.

Assume that $|Z_t| \geq h$ holds for $t = t_a + 8\sqrt{n\Delta/\delta}\log n$. At t or later, the probability that agent a visits an informed vertex is at least $h/|T^a|$. Bounding the tail bound using Markov's inequality, we can conclude that agent a visits at least one informed vertex by the time $t_a + O\left(\sqrt{\frac{n\Delta}{\delta}}\log n\right)$ with probability $1 - 1/n^2$ *or more. That is, two agents meet within $t_a + O\left(\sqrt{\frac{n\Delta}{\delta}}\log n\right)$ rounds with probability at least* $1 - O(1/n^2)$. *Hence, the lemma is proven.* □

3.2.3 Construction of T^a

In what follows, we simply say that a vertex is heavy or light if it is $\delta/8$-heavy or $\delta/2$-light respectively. By Lemma 7, it suffices that agent a constructs a $(a, \delta/8, 2)$-dense set T^a to achieve rendezvous. The algorithm Construct takes the role of constructing T^a, which utilizes a subroutine called Sample. The pseudocode of Sample and Construct are presented in Algorithms 4 and 5 respectively. In algorithm Construct, agent a manages a set $S^a \subseteq N^+(v_0^a)$, and iteratively adds a vertex to S^a. In the following argument, we refer to the process of adding the i-th vertex to S^a as the *i-th iteration*. Eventually, the algorithm outputs $N^+(S^a)$ as the constructed set T^a when it satisfies the termination condition (which is explained later). Let S_i^a be the set stored in S^a at the beginning of the i-th iteration, and x_i be the vertex added in the i-th iteration. The principle of choosing x_i is very simple: Agent a selects a vertex x_i such that the volume of $N^+(x_i) \setminus N^+(S_i^a)$ is large. Specifically, it searches a vertex $w \in N^+(v_0^a)$ that is light for $N^+(S_i^a)$. If such a vertex exists, it is added to S_i^a as x_i. Otherwise, any vertex in $N^+(v_0^a)$ is heavy for $N^+(S_i^a)$, i.e., $N^+(v_0^a) \subseteq H_{\delta/8}(N^+(S_i^a))$. This implies that $N^+(S_i^a)$ satisfies $(a, \delta/8, 2)$-dense condition, and the algorithm can return it as T^a. Adding a light vertex to S_i^a increases the cardinality of $N^+(S^a)$ by at least $\Theta(\delta)$, and thus the algorithm Construct obviously terminates within $O(n/\delta)$ iterations (because if $N^+(S^a) = V$ holds, any vertex becomes heavy for $N^+(S^a)$).

For expanding S_i^a by adding a light vertex, the algorithm has to check the heaviness of each vertex in $N^+(v_0^a)$ (for $N^+(S_i^a)$). The algorithm Sample takes this role. More precisely, the run of Sample(Γ, α) probabilistically checks whether or not each vertex in $N^+(v_0^a)$ is α-heavy for Γ within $O(|\Gamma|/\alpha)$ rounds. The algorithm outputs the vertex set consisting of the vertices concluded as α-heavy for Γ. A straightforward approach of identifying x_i in the construction of T^a is to run Sample$(N^+(S_i^a), \delta/8)$ in every iteration. However, then the total running time of Construct becomes $O((n/\delta)^2)$ rounds. To save time, our algorithm finds a light vertex x_i using the following two-step strategy:

- (Step 1) **Optimistic decision**: In the i-th iteration, agent a runs Sample$(\Gamma, \delta/8)$ for $\Gamma = N^+(S_i^a) \setminus N^+(S_{i-1}^a)$. If it detects that a vertex $u \in N^+(v_0^a)$ is heavy for Γ, Proposition 1 guarantees that u is heavy for $N^+(S_i^a) \supseteq \Gamma$. On the other hand, vertex u can be heavy for Γ even if the algorithm says that u is light. Then adding a vertex u as x_i prevents the algorithm from working correctly as intended.

- (Step 2) **Strict decision**: To resolve the matter of step 1, agent a checks if the candidates of x_i are actually light for $N^+(S_i^a)$. More precisely, the agent samples $\Theta(\log n)$ vertices uniformly at random from the set output by the run of

Sample$(\Gamma, \delta/8)$, and then it checks the heaviness of each sample v by actually visiting there and computing $|N^+(S_i^a) \cap N^+(v)|$. If the agent finds a light vertex from the $\Theta(\log n)$ samples, that vertex is selected as x_i. Otherwise, it finds that a constant fraction of whole candidates for x_i in the optimistic decision is heavy for $N^+(S_i^a)$ with high probability. Then the agent runs Sample$(N^+(S_i^a), \delta/8)$ for strict checking. If a vertex u is found light for $N^+(S_i^a)$, the agent selects u as x_i. Otherwise, the algorithm terminates.

In the following argument, we refer to the runs of Sample in step 1 and 2 as *optimistic/strict runs* of Sample(Γ, α) respectively. Since the running time of each optimistic run depends on the size of difference set $N^+(S_i^a) \setminus N^+(S_{i-1}^a)$, the total sum of the running time incurred by optimistic runs is $O((n \log n)/\delta)$. While each strict run of Sample needs at most $O((n \log n)/\delta)$ rounds, we can show that strict runs are executed at most $O(\log n)$ times. It comes from the two facts that 1) one strict run corrects the identification of a constant fraction of heavy vertices in $N^+(S_i^a)$ which are wrongly identified as light ones, and 2) a vertex identified as a heavy one is never identified as light. Consequently the total running time of Construct is bounded by $O(n \log^2 n/\delta)$ steps. We explain the details of Sample(Γ, α) and Construct in the following paragraphs.

Sample(Γ, α)

For the decision of lightness/heaviness of each vertex in $N^+(v_0^a)$ for Γ, this algorithm conducts random samplings and visits. The agent uses an array $C \subseteq \mathbb{Z}^{|N^+(v_0^a)|}$, which counts for each $u \in N^+(v_0^a)$ the number of visited vertices having u as a neighbor. The initial value of $C[u]$ for each $u \in N^+(v_0^a)$ is $C[u] = 0$. Let l be a threshold value $l = \lceil 150 \ln n \rceil$. In the run of Sample$(\Gamma, \alpha)$, the agent repeatedly visits a vertex v in Γ chosen uniformly at random (with duplication) $96\lceil |\Gamma| (\ln n)/\alpha \rceil$ times. At the visited vertex v, it increments $C[u]$ for each vertex u in $N^+(v_0^a) \cap N^+(v)$ (for this process, the agent carries the information of $N^+(v_0^a)$). After processing all samples, the agent concludes that u is heavy for Γ if $C[u] \geq l$ holds, or light otherwise. The algorithm outputs the vertex set H' consisting of the vertices concluded as a heavy one.

Construct

In this algorithm agent a has the following sets as its internal variables: S_i^a, R_i, H_i, and NS_i^a. The subscript i corresponds to the number of iterations in the algorithm. The set R_i is a set of candidates for x_i. The set H_i stores the vertices that turned out to be $(\delta/8)$-heavy for $N^+(S_i^a)$ at the i-th iteration. The variable NS_i^a keeps track of the set $N^+(S_i^a)$.

Algorithm 4 Sample(Γ, α)

```
l: threshold value $l = \lceil 150 \ln n \rceil$
 1: for $i = 1$ to $96\left\lceil \frac{|\Gamma| \ln n}{\alpha} \right\rceil$ do
 2:     choose a vertex $v$ in $\Gamma$ uniformly at random
 3:     visit $v$
 4:     for all $u \in N^+(v) \cap N^+(v_0^a)$ do
 5:         $C[u]++$
 6:     end for
 7: end for
 8: for all $u \in N^+(v_0^z)$ do
 9:     if $C[u] \geq l$ then
10:         $H' \leftarrow H' \cup \{u\}$
11:     end if
12: end for
13: return $H'$
```

The initial value of these sets are $S_1^a = \{v_0^a\}$, $R_1 = N^+(v_0^a)$, $H_1 = \emptyset$, and $NS_i^a = N^+(v_0^a)$ respectively. The agent a iterates the following operations until $R_i = \emptyset$. First, the agent executes the optimistic run of Sample$(N^+(S_i^a) \setminus N^+(S_{i-1}^a), \delta/8)$, and for the returned set H' it updates H_i and R_i with $H_{i+1} \leftarrow H_i \cup H'$ and $R_i \leftarrow N^+(v_0^a) \setminus H_{i+1}$. Based on the updated set R_{i+1}, the agent randomly chooses $\lceil 4 \log n \rceil$ vertices from R_{i+1} and visits each sampled vertex. If a visited vertex is actually light for $N^+(S_i^a)$ (this is checked by using the information of NS_i^a), then the agent adds it to S_i^a as x_i. Otherwise, (i.e., all of the vertices are heavy for $N^+(v_0^a)$), then the agent executes the strict run of Sample$(N^+(S_i^a), \delta/8)$ and updates the set H_{i+1} and R_{i+1} in the same way as the optimistic run. After that, the agent selects any vertex in R_{i+1} and adds it to S_i^a.

3.2.4 Correctness Proof of Algorithm Sample(Γ, α)

Lemma 8 below shows that the algorithm Sample(Γ, α) probabilistically checks if a vertex $u \in N^+(v_0^a)$ is approximately heavy or light for Γ.

Lemma 8 *Let $\alpha > 0$ and $\Gamma \subseteq N^+(v_0^a)$ satisfy $|\Gamma| \geq \alpha$. The following statements hold for any $u \in N^+(v_0^a)$ and the output set H' of Sample(Γ, α) with probability at least $1 - 1/n^8$:*

1. *If $u \in H'$ then u is α-heavy for Γ.*

2. *if $u \in N^+(v_0^a) \setminus H'$ then u is 4α-light for Γ.*

Proof 6 *We prove that 1) if $u \in N^+(v_0^a)$ is α-light for Γ, then after the execution of the algorithm, $C[u] < l$ holds with high probability., and 2) if the vertex u is 4α-heavy then $C[u] \geq l$ with high probability. This trivially implies the lemma. Consider the proof of the first statement. Suppose that u is α-light for Γ. Then we have $|N^+(u) \cap \Gamma| < \alpha$. Let X_1 be the random variable corresponding to the value stored in $C[u]$ after the execution of* Sample(Γ, α). *Since X_1 follows the binomial distribution $B(m, p)$ with parameter $p = |N^+(u) \cap \Gamma|/|\Gamma| < \alpha/|\Gamma|$ and $m = 96\lceil(|\Gamma| \ln n)/\alpha\rceil$, $\mathbb{E}[X_1] \leq 96\lceil(|\Gamma| \ln n)/\alpha\rceil \cdot \alpha/|\Gamma| \leq (96 \ln n) + 1$ holds. Let $\mu_1 = (96 \ln n) + 1$ for short. Using Chernoff bound, we have*

$$\begin{aligned}\Pr[X_1 \geq l] &\leq \Pr[X_1 \geq (1 + 1/2)\mu_1] \\ &\leq \mathrm{e}^{-\mu_1/(3\cdot 2^2)} \leq 1/n^8.\end{aligned}$$

We next consider the second statement. Suppose that u is 4α-heavy for Γ. Then we have $|N^+(u) \cap \Gamma| \geq 4\alpha$. Similarly, with the first proof, we define the random variable X_2 corresponding the value of $C[u]$ after the execution of the algorithm. Since it follows the binomial distribution $B(m, p)$ with the same parameter as the first proof, we have $\mathbb{E}[X] \geq 96\lceil(|\Gamma| \ln n)/\alpha\rceil \cdot (4\alpha/|\Gamma|) \geq 96((|\Gamma| \ln n)/\alpha) \cdot (4\alpha/|\Gamma|) \geq 384 \ln n$. Letting $\mu_2 = 384 \ln n$, Chernoff bound provides the following inequality.

$$\begin{aligned}\Pr[X_2 \leq l] &\leq \Pr[X_2 \leq (1 - 1/2)\mu_2] \\ &\leq \mathrm{e}^{-\mu_2/(3\cdot 2^2)} \leq 1/n^8.\end{aligned}$$

Thus, the lemma is proven. □

The next corollary immediately implies the correctness of algorithm Sample(Γ, α), which is obtained by Lemma 9 and the standard union-bound argument.

Corollary 9 *Consider any call of* Sample(Γ, α). *If $|\Gamma| \geq \alpha$, then $H' \subseteq H_\alpha(\Gamma)$ and $N^+(v_0^a) \setminus H' \subseteq L_{4\alpha}(\Gamma)$ hold with probability at least $1 - 1/n^7$.*

Note that the running time of the algorithm Sample(Γ, α) is $O(\frac{\Gamma \ln n}{\alpha})$.

3.2.5 Correctness Proof of Algorithm Construct

Now we turn to the analysis of the algorithm Construct. Our first goal of this analysis is to show that the algorithm Construct constructs a desired $(a, \delta/8, 2)$-dense set T^a in $O(n/\delta)$ iterations. As we stated at the description of the algorithm (in section 3.2.3), the key observation for this goal is that in each iteration the algorithm adds a light vertex x_i to S_i. We show this observation in Lemma 11. Before proving Lemma 11, we state auxiliary lemma, which proves any strict run of the algorithm divides $N^+(v_0^a)$ into a set R_i of light vertices and a set H_i of heavy vertices with high probability. This lemma shows that the algorithm selects light vertex x_i in each strict run of the algorithm.

Lemma 10 *If the strict run occurs at the i-th iteration, $R_i \subseteq L_{\delta/2}(N^+(S_{i-1}^a))$ and $H_i \subseteq H_{\delta/8}(N^+(S_{i-1}^a))$ hold with probability at least $1 - O(1/n^7)$.*

Proof 7 *Since S_i^a is nonempty and its cardinality is monotonically increasing, we have $|S_i^a| \geq 1$, and thus $\Gamma = N^+(S_i^a) \geq \delta$ holds at the beginning of the strict run at the i-th iteration. This implies $|\Gamma| \geq \alpha = \delta/8$. By Corollary 9, $R_i \subseteq L_{\delta/2}(N^+(S_{i-1}^a))$ and $H_i \subseteq H_{\delta/8}(N^+(S_{i-1}^a))$ holds with probability at least $1 - 1/n^7$.* □

Lemma 11 *For any i, x_i is $\delta/2$-light for $N^+(S_i^a)$ with probability at least $1 - O(1/n^7)$.*

Proof 8 *We first consider the case that x_i is added without strict runs. In this case, agent a directly visits x_i and checks its heaviness. Hence, the lemma obviously holds. We next consider the case that x_i is added after the strict run. By Lemma 10, $R_{i+1} \subseteq L_{\delta/2}(N^+(S_i^a))$ holds with probability at least $1 - 1/n^7$. Thus any vertex $v \in R_{i+1}$ is $\delta/2$-light for $N^+(S_i^a)$. Hence, the lemma holds.* □

Now we show that in each iteration $H_{i+1} \subseteq H_{\delta/8}(N^+(S_i^a))$ holds.

Lemma 12 *For any $i \in [1, n-1]$, let Y_i be the indicator random variable taking $Y_i = 1$ if and only if $H_{i+1} \subseteq H_{\delta/8}(N^+(S_i^a))$ holds. Then we have $\Pr\left[\bigcap_{i=1}^n Y_i = 1\right] \geq 1 - O(1/n^6)$.*

Proof 9 *Since S_i^a is nonempty and its cardinality is monotonically increasing, we have $|S_i^a| \geq 1$, and thus $\Gamma = N^+(S_i^a) \geq \delta$ holds at the beginning of the strict run in the i-th iteration. It implies $|\Gamma| \geq \alpha = \delta/8$. By Lemma 11, $|N^+(S_{i-1}^a) \cap N^+(x_i)| < \delta/2$ holds, and then we have $|N^+(S_{i-1}^a) \setminus N^+(x_i)| \geq \delta/2 > \alpha$. Hence any call of Sample satisfies the assumption of Corollary 9 with probability at least $1 - 3/n^7$. Since Sample is called at most $O(n)$ times, a standard union-bound argument provides the lemma.* □

By using Lemma 12, we prove that the algorithm eventually finds a $(a, \delta/8, 2)$-dense set T^a in Lemma 13. We also prove the upper bound for the number of iterations of the algorithm.

Lemma 13 *Algorithm Construct outputs a $(a, \delta/8, 2)$-dense set T^a within $O(n/\delta)$ iterations with probability at least $1 - O(1/n^5)$.*

Proof 10 *Let $T^a = N^+(S^a_j)$. That is, the algorithm terminates at the j-th iteration. First we show that T^a is $(a, \delta/8, 2)$-dense. Since $S^a_i \subseteq N^+(v^a_0)$ holds, the first and second conditions of $(a, \delta/8, 2)$-dense condition are obviously satisfied. Consider the third condition. By definition, two sets R_i and H_i are always a partition of $N^+(v^a_0)$. Thus we obtain $H_j = N^+(v^a_0)$ because $R_j = \emptyset$ holds. Lemma 12 implies that $N^+(v^a_0) = H_j \subseteq H_{\delta/8}(N^+(S^a_j))$ holds. That is, $T^a = N^+(S^a_j)$ satisfies the third condition.*

We next show that the event $R_i = \emptyset$ occurs within $O(n/\delta)$ iterations. By Lemma 11, $|N^+(x_i) \setminus N^+(S^a_{i-1})| \geq \delta/2$ holds for any x_i. Then we have $|N^+(S^a_j)| \geq j\delta/2$. Due to the trivial upper bound of $|N^+(S^a_j)| \leq n$, we obtain $j \leq 2n/\delta = O(n/\delta)$. The success probability of the lemma is derived from taking the union bound for at most $O(n)$ applications of Lemmas 11 and 12. $\square$

We analyse the time complexity of the algorithm Construct.

Lemma 14 *The total running time of Construct is $O(n \log^2 n/\delta)$ time with probability at least $1 - O(1/n^3)$.*

Proof 11 *We first bound the total running time incurred by the part of optimistic decision. Assume that T^a is constructed at the j-th iteration. For each $1 \leq i \leq j-1$, the optimistic run of Sample$(N^+(x_i) \setminus N^+(S^a_i), \delta/8)$ takes $96\lceil |(N^+(x_i) \setminus N^+(S^a_i)| \ln n/\delta \rceil$ rounds. Hence, the total running time is bounded by*

$$\sum_{i=1}^{r} 96 \left\lceil \frac{|N^+(x_i) \setminus N^+(S^a_i)| \ln n}{\delta} \right\rceil \leq O\left(\frac{N^+(S^a_j) \log n}{\delta}\right)$$
$$= O\left(\frac{n \log n}{\delta}\right).$$

We next consider the time complexity caused by the part of strict decision. We show that Sample is executed as a strict run at most $O(\log n)$ times. It is sufficient to prove that at least a constant fraction of R_i is moved to H_{i+1} with high probability if the strict run occurs at the i-th iteration. In each i-th iteration, let g_i be the number of $(\delta/8)$-heavy vertices for $N^+(S^a_i)$. We show that $g_i/|R_i| \geq 1/2$ holds if the agent samples no light vertex from R_i in the strict run of Sample. Consider the case of $g_i/|R_i| < 1/2$. Then the probability that the agent samples a $\delta/8$-heavy vertex is at most $1/2$. Thus, the probability that all of the sampled vertices are $\delta/8$-heavy is at most $(1/2)^{\lceil 4 \log n \rceil} \leq 1/n^4$. Conversely, if all of the sampled vertices are $\delta/8$-heavy, $g_i/|R_i| \geq 1/2$ holds with probability at least $1 - 1/n^4$. By Lemma 10, the strict run of Sample in the i-th iteration moves all the $\delta/8$-heavy vertices in R_i to H_{i+1} with high *probability. Then at least a half of the elements in R_i are deleted. Since the cardinality of R_i never increases, the number of calls to* Sample *as a strict run is at most $O(\log n)$ times with high probability. Each strict run takes $O((|N^+(S_i^a)| \log n)/\delta)$ rounds, and thus the total running time of* Construct *is bounded by $O((n \log^2 n)/\delta)$. The success probability of the lemma is obtained by taking union bounds on $O(\log n)$ applications of Lemma 10.* □

Finally, we obtain the main lemma of Construct.

Lemma 15 *Algorithm* Construct *outputs T^a satisfying $(a, \delta/8, 2)$-dense condition in $O(n \log^2 n/\delta)$ rounds with probability at least $1 - O(1/n^3)$.*

The combination of this lemma and Lemma 7 deduces the correctness of our rendezvous algorithm.

Theorem 16 *Let $G = (V, E)$ be any graph such that $\delta_G \geq \sqrt{n}$ holds. There is an algorithm that completes rendezvous within $O\left(\frac{n}{\delta} \log^2 n + \sqrt{\frac{n\Delta}{\delta}} \log n\right)$ rounds with high probability.*

3.3 Discussion

3.3.1 Removing the Assumption of Min-Degree Knowledge

In the algorithm presented in Subsection 3.2.3, we suppose that agents know a constant factor approximation of δ. This assumption can be easily removed by a simple doubling-estimation mechanism. Precisely, in the construction of T^a (which is the only part of the algorithm using δ), agent a initially sets δ' to the half of the degree of v_0^a. If the agent visits a vertex whose degree is less than δ', then it restarts the procedure of *Construct* after halving δ'. Note that we do not have to restart agent b for synchronization because its behavior (in Main-rendezvous) is inherently oblivious (i.e., iteratively marking neighbors). Eventually the procedure terminates without restarting when $\delta' < \delta_G$ is satisfied. Since the running time of Construct is $O((n \log^2 n)/\delta')$, the doubling update of δ' does not incur any extra asymptotic cost. That is, if the estimation of δ' starts from a range $[2^j, 2^{j+1}]$, the total running time is bound as follows:

$$\begin{aligned}\sum_{\lfloor \log \delta \rfloor \le j' \le j} & O(n \log^2 n / 2^{j'}) \\ &= O(n \log^2 n/\delta) \cdot \left(1 + \frac{1}{2} + \cdots + \frac{1}{2^{j - \lfloor \log \delta \rfloor}}\right) \\ &= O\left(\frac{n \log^2 n}{\delta}\right).\end{aligned}$$

Corollary 17 *The modified algorithm stated above outputs T^a (equivalently, $N^+(S_i^a)$) satisfying $(a, \delta'/8, 2)$-dense set in $O(n \log^2 n/\delta')$ rounds with probability at least $1 - O(1/n^3)$.*

3.3.2 Algorithm without Using Whiteboards

In this subsection, we present a rendezvous algorithm Rendezvous-without-Whiteboard that does not use whiteboards, under the assumption that nodes are tightly named (that is, $n' = O(n)$). We present the pseudo-code of the algorithm in Algorithm 6. This algorithm assumes that agents know the value of n' and the minimum degree δ, but the minimum-degree assumption can be removed by the technique in Section 3.3.1. In this algorithm, agent a first constructs a set $T^a \subseteq N^+(N^+(v_0^a))$ in the same way as the original one (recall that Construct does not use whiteboards). In order to synchronize the iterative probings of vertices by both agents, they start Rendezvous-without-Whiteboard at round $t' = c_1 n' \log^2 n/\delta$ for sufficiently large constant c_1 such that the construction of T^a finishes by round t'.

We define several notations. We denote the ID space $\{1, \ldots, n'\}$ by $\mathbb{S}_{ID}$. For any integer β, we define the β-partition $\{\mathbb{I}_1 \ldots, \mathbb{I}_{\lceil n/\beta \rceil}\}$ of $\mathbb{S}_{ID}$ as $\mathbb{I}_i = [(i-1)\beta + 1, i\beta]\}$ for all i. The goal of the algorithm is that for an appropriate β, the agents a and b respectively construct $\Phi_a \subseteq T^a$ and $\Phi_b \subseteq N^+(v_0^b)$ satisfying the following properties with high probability:

- (intersection) $|\Phi_a \cap \Phi_b| \ge 1$.
- (sparseness) There exists some constant c_2 such that $|\Phi_a \cap \mathbb{I}_i| \le c_2 \log n$ and $|\Phi_b \cap \mathbb{I}_i| \le c_2 \log n$ hold for any $i \in [1, \lceil n/\beta \rceil]$.

We first present the construction of Φ_a and Φ_b satisfying the properties above. For each $v \in T^a$, agent a adds v into Φ_a with probability $4 \ln n/\sqrt{\delta}$. Similarly, for each $v \in N^+(v_0^b)$, agent b adds v into Φ_b with probability $4 \ln n/\sqrt{\delta}$. Then we can guarantee with high probability that Φ_a and Φ_b satisfy the intersection property, and also satisfy the sparseness property for $\beta = \lceil\sqrt{\delta}\rceil$ and $c_2 = 18$.

We explain how rendezvous is achieved by using two sets Φ_a and Φ_b. The agents a and b iterate the following operations for all $i = 1, 2, \ldots, \lceil n/\sqrt{\delta}\rceil$ (referred as i-th *phase* of agents a and b). The i-th phase consists of $\lceil 4c_2 \ln n\rceil^2$ rounds, and starts at round $t' + (i-1)\lceil 4c_2 \ln n\rceil^2 + 1$. In the i-th phase, agent a visits each vertex $v_j \in \Phi_a \cap \mathbb{I}_i$ in ascending order of its ID, and waits $\lceil 4c_2 \ln n\rceil$ rounds at each visited vertex. After visiting all the vertices in $\Phi_a \cap \mathbb{I}_i$, the agent waits at the initial position until round $t' + i\lceil 4c_2 \ln n\rceil^2$ to synchronize the next phase. The behavior of agent b is similar to that of a. It visits each $v_k \in \Phi_b \cap \mathbb{I}_i$ in ascending order of its ID. The agent b waits at each visited vertex for two rounds. Agent b repeats this process $\lceil 4c_2 \ln n\rceil$ times. Then it waits on the initial position until $t' + i \cdot \lceil 4c_2 \ln n\rceil^2$ rounds. We can show that agents a and b attain rendezvous in $\mathbb{I}_l$ such that $\Phi_a \cap \Phi_b \cap \mathbb{I}_l \neq \emptyset$ holds. The total time complexity is $O((n/\beta) \cdot \log^2 n) = O((n \log^2 n)/\sqrt{\delta})$ rounds.

Theorem 18 *Algorithm Rendezvous-without-Whiteboard achieves rendezvous in* $O\left(t' + \frac{n}{\sqrt{\delta}} \log^2 n\right)$ *rounds with probability at least* $1 - O(1/n^2)$.

Proof 12 *First, we show that Φ_a and Φ_b satisfy the intersection property. By the independence of the probabilistic choices of agents a and b, any node in $T^a \cap N^+(v_0^b)$ is contained in both Φ_a and Φ_b with probability* $(4\ln n/\sqrt{\delta})^2 = (4 \ln n)^2/\delta$. *Hence the probability p that $|\Phi_a \cap \Phi_b| = 0$ is upper bounded by* $p \leq \left(1 - \frac{(4\ln n)^2}{\delta}\right)^{\delta/8} \leq \mathrm{e}^{-2\ln^2 n} \leq \frac{1}{n^2}$. *That is, the intersection property is satisfied with high probability. Next, we show that Φ_a and Φ_b satisfy the sparseness property. For any $i \in [1, \lceil n/\sqrt{\delta}\rceil]$, let Y_i^a be the number of vertices in $N^+(v_0^a) \cap \mathbb{I}_i$. Then we have* $\mathbb{E}[Y_i^a] \leq \lceil\sqrt{\delta}\rceil \cdot 4\ln n/\sqrt{\delta} \leq 9 \ln n$. *Applying the Chernoff bound, the probability* $\Pr[Y_i^a \geq 18 \log n]$ *is upper bounded by* $\Pr[Y_i^a \geq 18 \ln n] \leq \Pr[Y_i^a \geq (1+1)9\ln n] \leq e^{-3\ln n} \leq \frac{1}{n^3}$. *By taking union bound over all $i \in [1, \lceil n/\sqrt{\delta}\rceil]$, a, and b, the probability that Φ_a and Φ_b do not satisfy the sparseness property is at most* $3/n^2$.

Finally, we show that if Φ_a and Φ_b satisfy the two properties, then rendezvous is achieved within $O((n \log^2 n)/\sqrt{\delta})$ rounds. We consider the l-th part such that $|\mathbb{I}_l \cap \Phi_a \cap \Phi_b| \geq 1$ holds. Let r be any vertex in $|\mathbb{I}_l \cap \Phi_a \cap \Phi_b|$, and s be the order of r in $\Phi_a \cap \mathbb{I}_l$ following IDs. By the definition of the algorithm, both a and b starts phase l at round $t' + (l-1)\lceil 4c_2 \ln n\rceil^2 + 1$. *In addition, the time when agent a stays at r is from round* $t' + (i-1)\lceil 4c_2 \ln n\rceil^2 + (s-1)\lceil 4c_2 \ln n\rceil - 2$ *to* $t' + (i-1)\lceil 4c_2 \ln n\rceil^2 + s\lceil 4c_2 \ln n\rceil - 2$. *During that period, agent b visits all the nodes in $\Phi_b \cap \mathbb{I}_l$. That is, rendezvous is achieved.* □

3.3.3 Achieving Rendezvous for $d \geq 2$

Since both Main-Rendezvous and Rendezvous-without-Whiteboard presented above work correctly under the assumption of $d = 1$, there is no guarantee to achieve rendezvous when $d > 1$. Fortunately, we can make these algorithms work for general $d \geq 1$ by combining the algorithms with the following termination detection scheme and a graph exploration algorithm. Roughly speaking, when one of the agents (more precisely, agent b in our algorithms) detects the fail of the algorithm of $d = 1$, it conducts the standard DFS algorithm taking $O(n)$ rounds, for finding the initial location of another agent (i.e., agent a). Since agent a periodically moves back its initial location. the rendezvous is achieved by making b stay at the initial location of a. The whole algorithm achieves the sublinear-time rendezvous for $d = 1$, and also achieves the rendezvous in $O(n \log^2 n)$ rounds for general $d > 1$ with high probability. Note that these algorithms are nearly optimal up to poly-logarithmic factor because we prove $\Omega(n)$-round lower bound for the case of $d = 2$ in Section 3.4.

The termination detection of the algorithms are as follows. Recall that in the algorithm, the agent b repeatedly visits neighbors of initial position until rendezvous. We add operations that the agent memorizes these visited neighbors in the whiteboard of the initial location, and the agent checks if it visits all neighbors. When the agent finds that it visits all neighbors (and does not achieved rendezvous), then it concludes that $d > 1$, and proceeds to the graph exploration. The time spent in this detection process is $O(|N(v)| \log n) = O(n \log n)$ rounds with high probability, which is obtained by the standard coupon collector argument.

For exploring the graph by agent b, we apply the Depth-First Search (DFS) algorithm, which roughly described as follows: The exploring agent at current vertex v searches an unvisited neighbor in $N(v)$, and if it is found, the agent moves to the vertex (i.e., *forward*); otherwise, the agent returns to the (neighboring) vertex from which the agent visits v first (i.e., *backtrack*). Obviously, the number of forward and backtrack movements are respectively upper bounded by n. Hence the crucial point of time complexity is that spent for checking if an unvisited vertex exists in $N(v)$ or not. In our setting, each agent has enough memory to memorize the whole visited vertices, and it also has the capability of knowing neighborhood IDs. Therefore the exploring agent can search an unvisited vertex locally by storing all visited vertices in its memory. Thus the time complexity of the DFS algorithm is $O(n)$ rounds in our setting. The precise implementation of the DFS algorithm in the setting of mobile agent systems is given in [35].

3.4 Impossibility for Sub-linear Time Rendezvous

In this section, we show four impossibility results for sublinear-time rendezvous, which respectively concern the four unconventional assumptions of our algorithm, namely, bounded minimum degrees, accessibility to neighborhood IDs, initial distance one, and randomization. In each proof, we show the impossibility results in the models relaxing the corresponding assumption. We define some terminologies used in the proofs. Given a graph G and an algorithm $\mathcal{A}$, let $\hat{X}(G, a, v, f(n))$ be the random variable representing the set of vertices visited by agent a initially at vertex v in G in the first consecutive $f(n)$ rounds. While this is an illegal run because b is not in the graph, but can identify the (probabilistic) set of vertices a visits. Also, we define $X(G, a, v, f(n))$ to be the vertex set defined as $X(G, a, v, f(n)) = \{x \in V(G) \mid \Pr[x \in \hat{X}(G, a, v, f(n))] \leq 1/4\}$.

3.4.1 Lower bound in the Case of Bounded Minimum Degrees

First, we show that there is a graph instance with minimum degree $\delta = o(\sqrt{n})$ and $\Delta = \omega(\sqrt{n})$ such that any algorithm needs $\Omega(\Delta)$ rounds for neighborhood rendezvous. Precisely, the $\Omega(n/\delta)$-round lower bound is obtained in the graphs with $\delta = o(\sqrt{n})$ and $\Delta = \Omega(\sqrt{n})$.

Theorem 19 *Letting $\delta = o(\sqrt{n})$ and $\Delta = \omega(\sqrt{n})$, the $(\Delta, \delta, 1)$-rendezvous problem has a class of instances where any rendezvous algorithm takes $\Omega(\Delta)$ rounds with a constant probability. In particular, the $(n/2, 1, 1)$-rendezvous problem has a class of instances where any rendezvous algorithm takes $\Omega(n)$ rounds with a constant probability.*

Proof 13 *We first consider the case of $\Delta = n/2$ and $\delta = 1$ for simplicity of argument. Suppose for contradiction that an algorithm $\mathcal{A}$ achieves rendezvous within $f(n) = o(n)$ rounds with high probability for the $(n/2, 1, 1)$-rendezvous problem. Assume that n is a multiple of 4 for simplicity, and let $[1, n]$ be the domain of vertex IDs. First, we consider a star graph $S_1(j)$ of $n/2 + 1$ vertices, where the ID of the center is $j \in [n/2 + 1, n]$, and IDs of all leaves are from $[1, n/2]$. In this graph we put agent a at the center vertex j, and run $\mathcal{A}$ during $f(n)$ rounds. It is easy to verify $|X(S_1(j), a, j, f(n))| > n/4$ because $f(n)$ is sublinear of n. Next, we consider a star graph $S_2(k)$ of $n/2 + 1$ vertices that consists of the center vertex with ID $k \in [1, n/2]$ and leaf sets with IDs $[n/2 + 1, n]$. It also satisfies $|X(S_2(k), b, k, f(n))| > n/4$. Now we consider a directed bipartite graph $G' = ([1, n/2], [n/2 + 1, n], E)$. The edge set E is defined as $E = \{(h, i) \mid h \in X(S_1(i), a, i, f(n)) \vee h \in X(S_2(i), b, i, f(n))\}$.*

Since we have $|X(S_1(i), a, i, f(n))| > n/4$ and $|X(S_2(i), b, i, f(n))| > n/4$ for all i, the total number of directed edges is more than $(n/2 \cdot n/4) \cdot 2 = n^2/4$. This means that there exists at least one pair (j, k) such that both (j, k) and (k, j) are contained in E. We consider the graph that consists of two star graphs of $n/2 + 1$ vertices sharing an edge (Fig. 3.1 (a)). The IDs of the two center vertices are j and k, and the IDs of j's leaves are from $[n/2 + 1, n] \setminus \{k\}$, and those of k's leaves are from $[1, n/2] \setminus \{j\}$. The edge (j, k) connects the two centers. In this graph, when we execute the algorithm $\mathcal{A}$ locating the two agents at j and k respectively, it is guaranteed that each agent does not pass through edge (j, k) in the first consecutive $f(n)$ rounds with probability at least $1/4$. That is, the algorithm does not achieve rendezvous within $f(n)$ rounds with probability at least $1/2$. This is a contradiction.

The general case can be proven in the same way as the argument above. The only difference is to change the degree of the center vertex to Δ and replace all the leaves of star graphs with a clique of size $s = \frac{n-2}{2\Delta} = \Omega(n/\Delta) = \Omega(\delta)$ where exactly one vertex is adjacent to the center (Fig. 3.1 (b)). That graph obviously satisfies the constraint of min/max degrees, and the proof above also applies to it. □

3.4.2 Lower bound in the Case of the No Accessibility to IDs of Neighborhood Vertices

Next, we show that any algorithm solving the ($\Theta(n)$,$\Theta(n)$,1)-rendezvous problem requires $\Omega(n)$ rounds in the worst case if agents have no access to IDs of neighborhood vertices.

Theorem 20 *Let n be even, $n \geq 6$, $\delta = n/2 - 1$ and $\Delta = n/2 - 1$, and assume that any agent has no access to neighborhood IDs. Then there exists an instance of $(\Delta, \delta, 1)$-rendezvous problem where any rendezvous algorithm takes $\Omega(\Delta)$ rounds with a constant probability.*

Proof 14 *Suppose for contradiction that an algorithm $\mathcal{A}$ achieves rendezvous within $f(n) = o(n)$ rounds with high probability for the $(n/2 - 1, n/2 - 1, 1)$-rendezvous problem. We first consider two cliques C_1 and C_2 of $n/2$ vertices where each vertex has an arbitrary ID. Let agent a be located at v_0^a in the clique C_1, and let agent b be located at v_0^b in the clique C_2. As the proof of Theorem 19, we make agents a and b execute algorithm $\mathcal{A}$ in each clique. By the assumption of $f(n) = o(n)$, it is easy to verify that $|X(C_1, a, v_0^a, f(n))| > n/4$ and $|X(C_2, b, v_0^b, f(n))| > n/4$ holds. Now we select vertices $x_1 \in X(C_1, a, v_0^a, f(n))$ and $x_2 \in X(C_2, b, v_0^b, f(n))$. Let $j = \hat{P}^{-1}_{v_0^a}(x_1)$, $k = \hat{P}^{-1}_{v_0^b}(x_2)$, $\bar{j} = \hat{P}^{-1}_{x_1}(v_0^a)$, and $\bar{k} = \hat{P}^{-1}_{x_2}(v_0^b)$. We construct a graph G by removing*

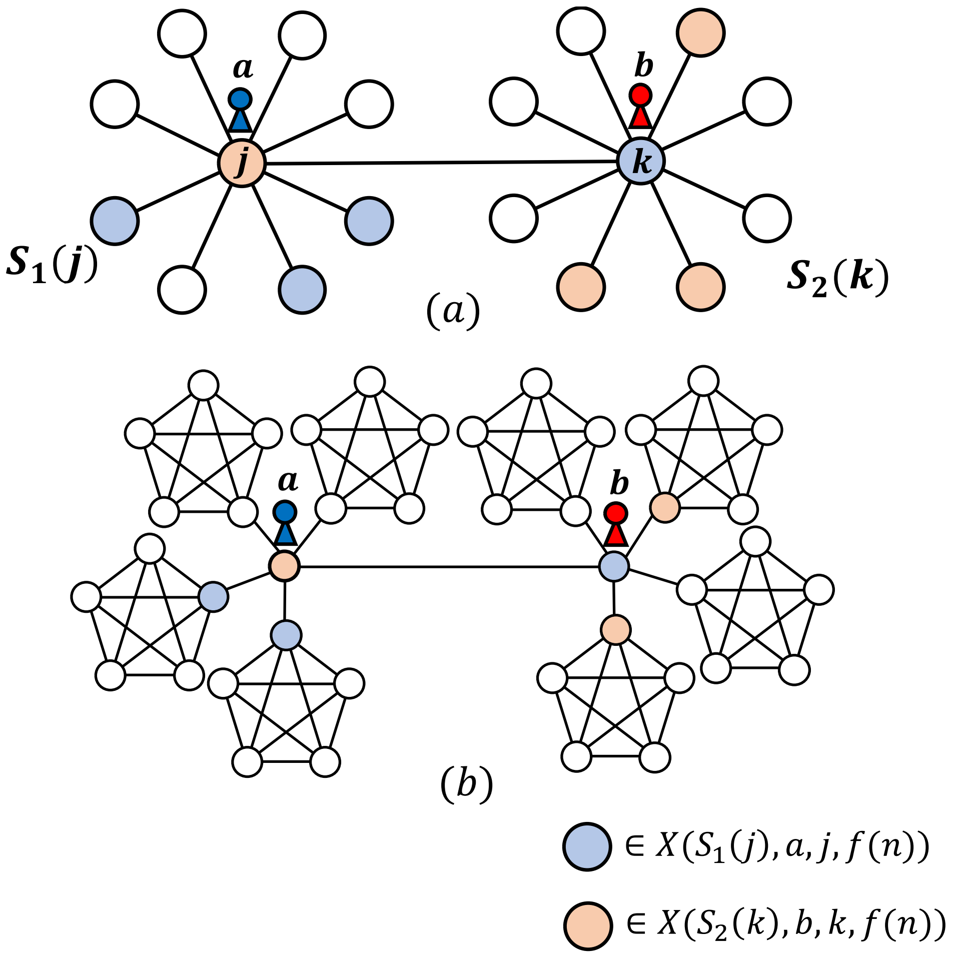

Figure 3.1: Proof of Theorem 19

edges (v_0^a, x_1) *and* (v_0^b, x_2) *from* C_1 *and* C_2 *respectively, and adding the edges* (v_0^a, v_0^b) *and* (x_1, x_2). *The local port number of those edges are defined as* $\hat{P}^{-1}_{v_0^a}(v_0^b) = j$, $\hat{P}^{-1}_{v_0^b}(v_0^a) = k$, $\hat{P}^{-1}_{x_1}(x_2) = \bar{j}$, *and* $\hat{P}^{-1}_{x_2}(x_1) = \bar{k}$. *The construction is illustrated in Fig. 3.2. Consider the* $f(n)$*-round run of* $\mathcal{A}$ *in* G *where two agents* a *and* b *start from* v_0^a *and* v_0^b *respectively. Since* v_0^a *and* v_0^b *are connected by an edge, this is an instance of the* $(n/2-1, n/2-1, 1)$*-rendezvous problem. Since* $x_1 \in X(C_1, a, v_0^a, f(n))$, *agent* a *visits* x_1 *or* v_0^b *with probability at most* $1/4$. *Similarly,* b *also visits* x_2 *or* v_0^a *with probability at most* $1/4$. *This implies that with probability at least* $1/2$ *no agent moves along edge* (v_0^a, v_0^b) *or* (x_1, x_2), *that is, rendezvous is not achieved at round* $n/2$ *with* *a constant probability. This is a contradiction.* □

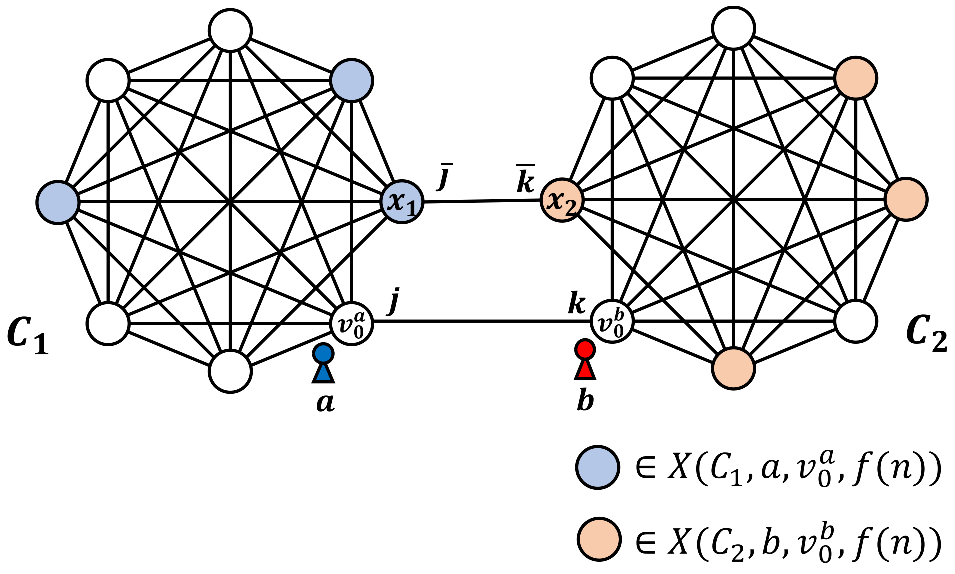

Figure 3.2: Proof of Theorem 20

3.4.3 Lower bound in the Case of the Distance Two of Initial Locations

Next, we show that the lower bound for the $(\Theta(n), \Theta(n), 2)$-rendezvous problem.

Theorem 21 *Let n be odd, $\Delta = n - 1$ and $\delta = (n-1)/2$. $(\Delta, \delta, 2)$-rendezvous problem has a graph instance where any algorithm takes $\Omega(\Delta)$ rounds with a constant probability.*

Proof 15 *Suppose for contradiction that an algorithm $\mathcal{A}$ achieves rendezvous within $f(n) = o(n)$ rounds with high probability for the $(n-1, (n-1)/2, 2)$-rendezvous problem. We first consider $(n+1)/2$ cliques $C_1, C_2, \ldots, C_{(n+1)/2}$ of $(n+1)/2$ vertices, where the i-th vertex set is $V(C_i)$. The IDs of the vertices of each clique C_i are assigned from $\left[\frac{n+1}{2}(i-1)+1, \frac{n+1}{2}i\right]$ respectively for all $i \in [1, \frac{n+1}{2}]$. Suppose that agent a executes algorithm $\mathcal{A}$ in each clique C_i with an arbitrary initial location $c_i \in V(C_i)$. By the assumption of $f(n) = o(n)$, it is easy to verify that $|X(C_i, a, c_i, f(n))| > (n+1)/4$. Let V' a vertex set that obtained by picking up one vertex $w_i \in |X(C_i, a, c_i, f(n))|$ for all $i \in [1, \frac{n+1}{2}]$, and we construct a clique*

C' consisting of $(n+1)/2$ vertices whose IDs come from V'. Suppose that agent b executes algorithm $\mathcal{A}$ in C' with an arbitrary initial location $c' \in V(C')$. It also satisfies $|X(C', b, c', f(n))| > (n+1)/4$ because $f(n)$ is sublinear of n. We pick up any vertex $x \in X(C', b, c', f(n))$. Letting C_k be the clique containing the vertex x, we construct the graph G consisting of two cliques C' and C_k sharing x (Fig. 3.3). Consider the $f(n)$-round run of $\mathcal{A}$ in G where a and b respectively start from c_k and c'. This is an instance of $(n-1, (n-1)/2, 2)$-rendezvous problem. Since $x \in X(C_k, a, c_k, f(n)) \cap X(C', b, c', f(n))$ holds, a and b do not visit x with probability at least $1/4$. That is, they cannot attain the rendezvous within $f(n)$ rounds at least with probability $1/2$. This is a contradiction. $\square$

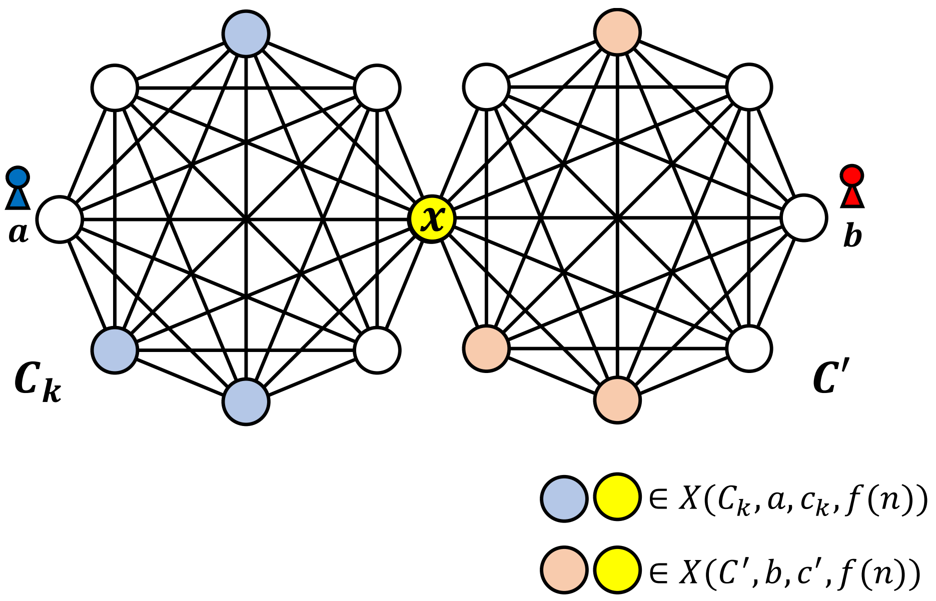

Figure 3.3: Proof of Theorem 21

3.4.4 Lower Bound for Deterministic Algorithms

We show that any deterministic algorithm solving the $(\Theta(n), \Theta(n), 1)$-rendezvous problem requires $\Omega(n)$ rounds in the worst case. First, we outline the proof strategy. Suppose for contradiction that an algorithm $\mathcal{A}$ solves $(\Theta(n), \Theta(n), 1)$-rendezvous problem within $o(n)$ rounds. In the proof, we adaptively construct the hard-core instance according to the behavior of $\mathcal{A}$: We start the construction with the two star graphs whose centers are the initial locations of two agents, and consider the run of

$\mathcal{A}$ in that graph. When the agent moves to an unvisited vertex, we adaptively fix its neighborhood vertices. More precisely, the graph construction roughly follows the process below: We select in advance $\Omega(n)$ vertices as a pool, and if an agent moves to an unvisited vertex with degree $o(n)$, we select $\Omega(n)$ vertices from the pool as neighbors. This construction provides two independent graphs respectively associated with two agents. Finally, we carefully glue them in the way of guaranteeing the initial distance one and minimum degree $\Omega(n)$, which becomes the instance yielding $\Omega(n)$-round lower bound.

We define some notations for explaining the details. Let n be a multiple of 32 for simplicity. As we stated, our proof first constructs two instances (for two agents) separately. By symmetry we only focus on the instance for agent a. We select an arbitrary ID space ID_a whose size is $n/2+1$ for the instance of agent a, and fix an initial vertex $v_0^a \in ID_a$. Let $Q_t^a(\mathcal{A}, G, v_0^a) = \{v_0^a, v_1^a, \ldots, v_t^a\}$. That is, $Q_t^a(\mathcal{A}, G, v_0^a)$ is the set of vertices visited by agent a in the execution of $\mathcal{A}$ starting from v_0^a in G up to round t. We also define the sequence $S_t^a(\mathcal{A}, G, v_0^a) = (v_0^a, v_1^a, \ldots, v_t^a)$ of the vertices in $Q_t^a(\mathcal{A}, G, v_0^a)$ with order. Given $\mathcal{A}$, $G = (V, E)$, v_0^a and a round $r \geq 0$, we can construct the *execution spanning subgraph* $\hat{G}_r^a(\mathcal{A}, G, v_0^a) = (\hat{V}, \hat{E})$ such that $\hat{V} = N_G^+(Q_r^a(\mathcal{A}, G, v_0^a))$ and $\hat{E} = \{(u, v) \mid u \in Q_r^a(\mathcal{A}, G, v_0^a) \wedge (u, v) \in E\}$. Intuitively, $\hat{G}_r^a(\mathcal{A}, G, v_0^a)$ represents the substructure of G seen by agent a in the execution of $\mathcal{A}$ starting from v_0^a up to round r. Now we assume any graph G' such that $\hat{G}_r^a(\mathcal{A}, G, v_0^a) = \hat{G}_r^a(\mathcal{A}, G', v_0^a)$ holds. It is obvious that the behavior of a in G' starting from v_0^a is completely same as that in G up to round $r+1$, and thus we obtain the following proposition.

Proposition 2 *Assume for any G, G', we have $\hat{G}_r^a(\mathcal{A}, G, v_0^a) = \hat{G}_r^a(\mathcal{A}, G', v_0^a)$. Then, $S_{r+1}^a(\mathcal{A}, G, v_0^a) = S_{r+1}^a(\mathcal{A}, G', v_0^a)$ holds.*

We show the lemma below, which is a key observation of our lower bound proof.

Lemma 22 *Let $\mathcal{A}$ be any algorithm terminating within $t \leq n/32$ rounds. Suppose that ID_a and v_0^a is given. There exists a graph G containing a vertex subset $W \subseteq N_G(v_0^a)$ of size at least $13n/32$ such that (i) $(Q_t^a(\mathcal{A}, G, v_0^a) \setminus \{v_0^a\}) \cap N_G^+(W) = \emptyset$ holds, and (ii) for each vertex $w \in V(G) \setminus (N_G^+(W) \setminus \{v_0^a\})$, $|N_G(w)| = \Theta(n)$ holds.*

Proof 16 *We adaptively construct the graph G according to the agent a's movement. Precisely, we incrementally fix the sequence of graphs $G_0, G_1, \ldots, G_t$ such that for each $r \in [0, t-1]$, $S_{r+1}^a(\mathcal{A}, G_r, v_0^a) = S_{r+1}^a(\mathcal{A}, G_{r+1}, v_0^a)$ is guaranteed. The vertex set of each G_i is common, which is denoted by V, and equal to ID_a (i.e., $V = ID_a$). Let $P \subseteq V \setminus \{v_0^a\}$ be an arbitrary subset of size $7n/16$, and $\overline{P} = V \setminus P$. We also define*

$E_0 = \{(v_0^a, u) \mid u \in ID_a \setminus \{v_0^a\}\} \cup \{(u, v) \mid u, v \in \overline{P} \wedge u \neq v\}$. For all $r \geq 0$, the algorithm $\mathcal{A}$ outputs the vertex $v_{r+1}^a \in N_{G_r}(v_r^a)$, as the destination of the movement at round r. Let $Q_r = Q_r^a(\mathcal{A}, G_r, v_0^a)$ for short. There are following two cases:

- *$v_{r+1}^a \in Q_r \cup \overline{P}$.*
- *$v_{r+1}^a \notin Q_r \cup \overline{P}$ (that is, $v_{r+1}^a \in P \setminus Q_r$).*

If $v_{r+1}^a \in Q_r \cup \overline{P}$ holds, we simply fix $G_{r+1} = G_r$ (i.e., $E_{r+1} = E_r$). Otherwise, we construct E_{r+1} by adding to E_r the edges from v_{r+1}^a to all the vertices in $\overline{P} \setminus Q_r$. In the following argument, we show $S_{r+1}^a(\mathcal{A}, G_r, v_0^a) = S_{r+1}^a(\mathcal{A}, G_{r+1}, v_0^a)$ holds for any $r \in [0, t-1]$ by the induction on r. In the base case of $r = 0$, we have $Q_0 = \{v_0^a\}$ and $S_0^a(\mathcal{A}, G_0, v_0^a) = (v_0^a)$. The algorithm outputs the vertex v_1^a as the destination of the movement in G_0 at round $r = 0$. In any case of updating rules, we can confirm that $\hat{G}_0^a(\mathcal{A}, G_0, v_0^a) = \hat{G}_0^a(\mathcal{A}, G_1, v_0^a)$. Therefore the vertex v_1^a in G_0 coincides with the one in G_1 and we have $S_1^a(\mathcal{A}, G_0, v_0^a) = S_1^a(\mathcal{A}, G_1, v_0^a)$. In the case of $r > 1$, assume that we are given G_r. The algorithm outputs the vertex v_{r+1}^a as the destination of the movement in G_r at round r. If $v_{r+1}^a \in Q_r \cup \overline{P}$, then $G_r = G_{r+1}$ holds, we have $S_{r+1}(\mathcal{A}, G_r, v_0^a) = S_{r+1}(\mathcal{A}, G_{r+1}, v_0^a)$. Otherwise, since we add edges between unvisited vertices (from $v_{r+1}^a \in P \setminus Q_r$ to each $u \in \overline{P} \setminus Q_r$), it follows $\hat{G}_r^a(\mathcal{A}, G_r, v_0^a) = \hat{G}_r^a(\mathcal{A}, G_{r+1}, v_0^a)$. Then by proposition 2, $S_{r+1}^a(\mathcal{A}, G_r, v_0^a) = S_{r+1}^a(\mathcal{A}, G_{r+1}, v_0^a)$ hold.

We set $P \setminus Q_t = W$ (that is, the vertices in P not visited by round t). Finally, we show that G_t has the desired property of the lemma. Since the agent visits to the vertices in P at most $t = n/32$ times, the size of W is at least $7n/16 - n/32 = 13n/32$. Since W is the set of vertices which are unvisited by agent a in the execution of $\mathcal{A}$ in G_t, by the updating rules of the graphs, each vertex in W is only connected to v_0^a. Therefore we have $(Q_t^a(\mathcal{A}, G, v_0^a) \setminus \{v_0^a\}) \cap N_G^+(W) = \emptyset$. Since $\overline{P}$ is a clique in G_0 (and thus in G_t), for each vertex $u \in \overline{P}$, we have $|N_{G_t}(u)| \geq n/16 - 1 = \Theta(n)$. For each vertex $u \in P \cap Q_r$, the size of $\overline{P} \setminus Q_r$ is at least $n/16 - n/32 = n/32$ at any round $r \in [0, t]$, and thus we have $|N_{G_t}(u)| \geq n/32 = \Theta(n)$. □

By the proposition and the lemma, we can construct the hard-core instance for the deterministic algorithm. In the proof, we apply Lemma 22 several times according to the agent IDs and initial positions v_0^a, v_0^b. Therefore in the proof we add subscripts of agent IDs and initial vertices to G and W constructed by the lemma, as $G_{(a, v_0^a)}$ and $W_{(a, v_0^a)}$.

Theorem 23 *For $\Delta = \Theta(n)$ and $\delta = \Theta(n)$, the $(\Delta, \delta, 1)$-rendezvous problem has a graph instance where any deterministic algorithm takes $\Omega(\Delta)$ rounds with probability one.*

Proof 17 *Suppose for contradiction that a deterministic algorithm $\mathcal{A}$ achieves rendezvous within $f(n) = n/32$ rounds for the $(\Delta, \delta, 1)$-rendezvous problem of $\Delta = \Theta(n)$ and $\delta = \Theta(n)$. Let $[1, n]$ be the domain of vertex IDs.*

We select $[1, n/2]$ and $j \in [n/2 + 1, n]$ as the ID space of the execution of the agent a, denoted by ID_a. We choose $v_0^a = j$ as the initial vertex of a, and construct $G_{(a,j)}$ by using Lemma 22. Similarly, we adaptively construct the graph instance according to the agent b's moves alone. We select $[n/2 + 1, n]$ and $k \in [1, n/2]$ as the ID space, denoted by ID_b. We choose $v_0^b = k$ as the initial vertex of b, and construct $G_{(b,k)}$ by also using Lemma 22.

Now we consider a directed bipartite graph $G' = ([1, n/2], [n/2 + 1, n], E)$. The edge set E is defined as $E = \{(x, y) \mid (x = j \wedge y \in W_{(a,j)}) \vee (x = k \wedge y \in W_{(b,k)}))\}$ for all j and k. Since we have $|W_{(a,j)}| \geq (13/32)n > n/4$ and $|W_{(b,k)}| \geq (13/32)n > n/4$ for all j and k, the total number of directed edges is more than $(n/2 \cdot n/4) \cdot 2 = n^2/4$. This means that there exists at least one pair (j, k) such that both (j, k) and (k, j) are contained in E. Finally we construct the whole graph instance. Prepare $G_{a,j}$ and G_{b_k} as the subgraphs of the constructed instance. Then we add an edge between j and k. We augment edges between any vertices in $W_{(a,j)} \setminus \{k\}$ and in $W_{(b,k)} \setminus \{j\}$ respectively. By the condition (ii) of Lemma 22, it is easy to verify that the minimum degree of the constructed instance is $\Theta(n)$. In this graph, consider the execution of $\mathcal{A}$ where two agents a and b are respectively located at j and k. By the condition (i) of Lemma 22, it is guaranteed that each agent does not pass through edge (j, k) in the first consecutive $n/32$ rounds. That is, the algorithm does not achieve rendezvous within $f(n)$ rounds. This is a contradiction. □

Algorithm 5 Construct

```
1: while $R_i \neq \emptyset$ do
2:     $H' \leftarrow \mathsf{Sample}(N^+(S_i^a) \setminus N^+(S_{i-1}^a), \delta/8)$;
3:     $H_{i+1} \leftarrow H_i \cup H'$;
4:     $R_{i+1} \leftarrow N^+(v_0^a) \setminus H_{i+1}$;
5:     if $R_{i+1} \neq \emptyset$ then
6:         for $j = 1$ to $\lceil 4 \log n \rceil$ do
7:             choose $u_j \in R_{i+1}$ uniformly at random;
8:             visit $u_j$;
9:             compute $|N^+(S_i^a) \cap N^+(u_j)|$ using $NS_i^a$;
10:            if $u_j$ is $\delta/2$-light for $N^+(S_i^a)$ then
11:                $x_i \leftarrow u_j$
12:                $S_{i+1}^a \leftarrow S_i^a \cup \{x_i\}$;
13:                $R_{i+1} \leftarrow R_{i+1} \setminus \{x_i\}$;
14:                break;
15:            end if
16:        end for
17:        if Each $u_j$ is $\delta/2$-heavy for $N^+(S_i^a)$ then
18:            $H' \leftarrow \mathsf{Sample}(N^+(S_i^a), \delta/8)$;
19:            $H_{i+1} \leftarrow H_{i+1} \cup H'$;
20:            $R_{i+1} \leftarrow N(v_0^a) \setminus H_{i+1}$;
21:            if $R_{i+1} \neq \emptyset$ then
22:                choose any vertex $x_i \in R_{i+1}$;
23:                $S_{i+1}^a \leftarrow S_i^a \cup \{x_i\}$;
24:                $NS_i^a \leftarrow NS_i^a \cup N^+(x_i)$;
25:                $R_{i+1} \leftarrow R_{i+1} \setminus \{x_i\}$;
26:            end if
27:        end if
28:    end if
29:    $i \leftarrow i + 1$
30: end while
31: return $N^+(S_i^a)$
```

Algorithm 6 Rendezvous-without-Whiteboards

Operations of Agent A

```
 1: Construct
 2: wait until $t = c_1\left(\frac{n'\log^2 n}{\delta}\right)$
 3: for all $u \in T^a$ do
 4:     $\Phi^a \leftarrow \Phi^a \cup \{u\}$ with probability $\frac{4\log n}{\sqrt{\delta}}$
 5: end for
 6: for $i = 1$ to $\lceil n/\sqrt{\delta}\rceil$ do
 7:     for all $u \in \Phi^a \cap \mathbb{I}_i$ do
 8:         visit $u$
 9:         wait on $u$ until $\lceil 4c_2 \log n\rceil$ time (including the round moving to $u$)
10:         return to $v_0^a$
11:     end for
12:     wait on $v_0^a$ until time $c_1\left(\frac{n'\log^2 n}{\delta}\right) + i\lceil 4c_2\log n\rceil^2$
13: end for
```

Operations of Agent B

```
 1: for all $u \in N^+(v_0^b)$ do
 2:     $\Phi^b \leftarrow \Phi^b \cup \{u\}$ with probability $\frac{4\log n}{\sqrt{\delta}}$
 3: end for
 4: wait until $t = c_1\left(\frac{n'\log^2 n}{\delta}\right)$
 5: for $i = 1$ to $\lceil n/\sqrt{\delta}\rceil$ do
 6:     for $j = 1$ to $\lceil 4c_2\log n\rceil$ do
 7:         for all $u \in \Phi^b \cap \mathbb{I}_i$ do
 8:             visit $u$
 9:             wait two time units on $v_0^b$
10:             return to $v_0^b$
11:         end for
12:     end for
13:     wait on $v_0^b$ until $t = c_1\left(\frac{n'\log^2 n}{\delta}\right) + i\lceil 4c_2\log n\rceil^2$
14: end for
```

Chapter 4

Concluding Remarks

In this dissertation, we considered the sub-linear time solvability of the two problems in autonomous and passively mobile agent systems. Specifically, for the passively mobile agent system, we considered the aggregation problem in probabilistic population protocol model, and we propose a sufficient condition for the sub-linear time solvability of the problem in the model. On the other hand, for the autonomous mobile agent system we consider the rendezvous problem in graphical mobile agent system, and we propose the minimal condition for the sub-linear time solvability of the problem. Our result is summarized as follows.

Sub-linear time Aggregation in Probabilistic Population Protocol Model

In chapter 2, we proposed a new aggregation algorithm that converges in $O(\sqrt{n}\log^2 n)$ parallel time with high probability and uses $O(|X|^2)$ states per ordinary agent. To the best of our knowledge, this is the first algorithm attaining a sub-linear running time which does not depends on the alphabet size $|X|$. We should point out that our algorithm does not guarantee the *stability* of executions. That is, though the failure probability is quite small, the system converges to a wrong output and never recovers.

A future research direction is to design a much faster convergence algorithm achieving polylogarithmic convergent time. Another challenging problem is to design a (relatively) faster algorithm not relying on the existence of the base station or a leader.

Fast Neighborhood Rendezvous

In chapter 3, we consider the neighborhood rendezvous problem, and propose two randomized algorithms for solving it. The first algorithm achieves rendezvous in

$O\left(\frac{n}{\delta}\log^3 n + \sqrt{\frac{n\Delta}{\delta}}\log n\right)$ rounds with high probability for graphs of minimum degree $\delta = \omega(\sqrt{n}\log n)$. The second algorithm achieves rendezvous in $O\left(\frac{n}{\delta}\log^2 n + \frac{n}{\sqrt{\delta}}\log^2 n\right)$ rounds with high probability. It does not use whiteboards. We also presented four impossibility results for sub-linear time rendezvous, where each result respectively considers four unconventional assumptions of our algorithm, that is, bounded minimum degrees, accessibility to neighborhood IDs, initial distance one, and randomization. One can obtain the $\Omega(n)$-round lower bound if either of them is removed. Therefore we conclude that our algorithms run under a minimal assumption.

Acknowledgements

I would like to express my gratitude to the people involved with my research for this dissertation. Especially, I am indebted to my supervisors Professor Yoshiaki Katayama and Assistant Professor Yonghwan Kim of Nagoya Institute of Technology, and my past supervisor Associate Professor Taisuke Izumi of Osaka University for their valuable guidance, support, and advice. I am also appreciative of all the staff and the students of Katayama Kim Laboratory and past Izumi Laboratory of Naogya Institute of Technology.

Publications

Journal Papers

- Ryota Eguchi, and Taisuke Izumi, "Sub-Linear Time Aggregationin Probabilistic PopulationProtocol Model", IEICE Transactions on Fundamentals of Electronics, Vol.9, p.1187-1194, 2019
- Ryota Eguchi, Naoki Kitamura, and Taisuke Izumi, "Fast Neighborhood Rendezvous", IEICE Transactions on Fundamentals of Electronics (to appear)

Conference Papers

- Ryota Eguchi, and Taisuke Izumi, "Brief Announcement: Fast Aggregation in Probabilistic PopulationProtocol Model", 31st International Symposium on Distributed Computing (DISC 2016), pp.49:1-49:3, Austria, October, 2018
- Ryota Eguchi, Naoki Kitamura, and Taisuke Izumi, "Fast Neighborhood Rendezvous", 40th IEEE International Conference on Distributed ComputingSystems (ICDCS 2020), pp.168-178, Singapore(Online), December, 2020

Bibliography

[1] Shehla Abbas, Mohamed Mosbah, and Akka Zemmari. A probabilistic model for distributed merging of mobile agents. In *2nd international workshop on verification and evaluation of computer and communication systems (VeCOS ' 08)*, 2008.

[2] D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, and R. L. Rivest. Time-space trade-offs in population protocols. In *Proc. of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2560–2579. SIAM, 2017.

[3] D. Alistarh, J. Aspnes, and R. Gelashvili. Space-optimal majority in population protocols. In *Proc. of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2221–2239. SIAM, 2018.

[4] D. Alistarh and R. Gelashvili. Polylogarithmic-time leader election in population protocols. In *International Colloquium on Automata, Languages, and Programming*, pages 479–491, 2015.

[5] D. Alistarh, R. Gelashvili, and M. Vojnović. Fast and exact majority in population protocols. In *Proc. of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 47–56. ACM, 2015.

[6] Dan Alistarh, Martin Töpfer, and Przemysław Uznański. Comparison dynamics in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 55–65, New York, NY, USA, 2021. Association for Computing Machinery.

[7] Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002.

[8] Steve Alpern. Rendezvous search on labeled networks. *Naval Research Logistics (NRL)*, 49(3):256–274, 2002.

[9] Steve Alpern, Robbert Fokkink, L Gasieniec, Roy Lindelauf, and VS Subrahmanian. *Search theory*. Springer, 2013.

[10] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International Series in Operations Research & Management Science*. Springer Science & Business Media, 2006.

[11] Edward J Anderson and RR Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability*, 27(4):839–851, 1990.

[12] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[13] D Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, September 2008.

[14] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[15] D. Angluin, J. Aspnes, M. J. Fischer, and H Jiang. Self-stabilizing population protocols. *ACM Transactions on Autonomous Adaptive Systtems*, 3(4):1–28, 2008.

[16] J. Aspnes, J. Beauquier, J. Burman, and D. Sohier. Time and space optimal counting in population protocols. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proc. in Informatics (LIPIcs)*, pages 13:1–13:17, 2017.

[17] Daisuke Baba, Tomoko Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Linear time and space gathering of anonymous mobile agents in asynchronous trees. *Theoretical Computer Science*, 478:118–126, 2013.

[18] J. Beauquier, P. Blanchard, and J. Burman. Self-stabilizing seader election in population protocols over arbitrary communication graphs. In *Proc. of 17th International Conference on Principle of Distributed Systems(OPODIS)*, pages 38–52, 2013.

[19] J. Beauquier, J. Burman, and S. Kutten. Making population protocols self-stabilizing. In *Proc. of 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems(SSS)*, pages 90–104, 2009.

[20] J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks with a base station. In *Proc. of 21st International Symposium on Distributed Computing(DISC)*, pages 63–76. Springer Berlin Heidelberg, 2007.

[21] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A Population Protocol for Exact Majority with O(log5/3 n) Stabilization Time and Theta(log n) States. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[22] Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct Population Protocols for Presburger Arithmetic. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[23] Michael Blondin, Javier Esparza, and Stefan Jaax. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[24] Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc, and Franck Petit. On deterministic rendezvous at a node of agents with arbitrary velocities. *Information Processing Letters*, 133:39–43, 2018.

[25] Nader H. Bshouty, Lisa Higham, and Jolanta Warpechowska-Gruca. Meeting times of random walks on graphs. *Information Processing Letters*, 69(5):259 – 265, 1999.

[26] Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 33–44, New York, NY, USA, 2021. Association for Computing Machinery.

[27] S. Cai, T. Izumi, and K. Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory of Computing Systems*, 50(3):433–445, 2012.

[28] Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theor. Comp. Sys.*, 50(3):433–445, apr 2012.

[29] Andrew Collins, Jurek Czyzowicz, Leszek Gąsieniec, Adrian Kosowski, and Russell Martin. Synchronous rendezvous for location-aware agents. In *International Symposium on Distributed Computing*, pages 447–459. Springer, 2011.

[30] Philipp Czerner and Javier Esparza. Lower bounds on the state complexity of population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 45–54, New York, NY, USA, 2021. Association for Computing Machinery.

[31] Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25(2):165–178, 2012.

[32] Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. Time versus space trade-offs for rendezvous in trees. *Distributed Computing*, 27(2):95–109, 2014.

[33] Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms (TALG)*, 8(4):37, 2012.

[34] Varsha Dani, Thomas P Hayes, Cristopher Moore, and Alexander Russell. Codes, lower bounds, and phase transitions in the symmetric rendezvous problem. *Random Structures & Algorithms*, 49(4):742–765, 2016.

[35] Shantanu Das. *Graph Explorations with Mobile Agents*, pages 403–422. Springer International Publishing, Cham, 2019.

[36] Shantanu Das, Dariusz Dereniowski, Adrian Kosowski, and Przemysław Uznański. Rendezvous of distance-aware mobile agents in unknown graphs. In *International Colloquium on Structural Information and Communication Complexity*, pages 295–310. Springer, 2014.

[37] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

[38] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

[39] Dariusz Dereniowski and Andrzej Pelc. Drawing maps with advice. *Journal of Parallel and Distributed Computing*, 72(2):132–143, 2012.

[40] D. Doty and D. Soloveichik. Stable leader election in population protocols requires linear time. In *International Symposium on Distributed Computing*, pages 602–616. Springer, 2015.

[41] Robert Elsasser, Tomasz Radzik, et al. Recent results in population protocols for exact majority and leader election. *Bulletin of EATCS*, 3(126), 2018.

[42] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Nicola Santoro, and Cindy Sawchuk. Multiple mobile agent rendezvous in a ring. In *Latin American Symposium on Theoretical Informatics*, pages 599–608. Springer, 2004.

[43] Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *International Symposium on Distributed Computing*, pages 242–256. Springer, 2008.

[44] L. Gαsieniec and G. Stachowiak. Fast space optimal leader election in population protocols. In *Proc. of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 265–266. Society for Industrial and Applied Mathematics, 2018.

[45] T. Izumi, K. Kinpara, T. Izumi, and K. Wada. Space-efficient self-stabilizing counting population protocols on mobile sensor networks. *Theoretical Computer Science*, 552:99–108, 2014.

[46] Taisuke Izumi. On space and time complexity of loosely-stabilizing leader election. In Christian Scheideler, editor, *Structural Information and Communication Complexity*, pages 299–312, Cham, 2015. Springer International Publishing.

[47] Nan Kang, Frederik Mallmann-Trenn, and Nicolás Rivera. Diversity, fairness, and sustainability in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 67–76, New York, NY, USA, 2021. Association for Computing Machinery.

[48] Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *International Colloquium on Structural Information and Communication Complexity*, pages 1–9. Springer, 2006.

[49] Evangelos Kranakis, Nicola Santoro, Cindy Sawchuk, and Danny Krizanc. Mobile agent rendezvous in a ring. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 592–599. IEEE, 2003.

[50] G. B Mertzios, S. E Nikoletseas, C. L. Raptopoulos, and P. G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *International Colloquium on Automata, Languages, and Programming*, pages 871–882. Springer, 2014.

[51] Avery Miller and Andrzej Pelc. Fast rendezvous with advice. *Theoretical Computer Science*, 608:190–198, 2015.

[52] Avery Miller and Andrzej Pelc. Tradeoffs between cost and information for rendezvous and treasure hunt. *Journal of Parallel and Distributed Computing*, 83:159–167, 2015.

[53] Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. *Distributed Computing*, 29(1):51–64, 2016.

[54] Andrzej Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.

[55] David Peleg. *Distributed computing: A locality-sensitive approach*. Society for Industrial and Applied Mathematics, 2000.

[56] Yuichi Sudo, Ryota Eguchi, Taisuke Izumi, and Toshimitsu Masuzawa. Time-Optimal Loosely-Stabilizing Leader Election in Population Protocols. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[57] Yuichi Sudo and Toshimitsu Masuzawa. Leader election requires logarithmic time in population protocols. *Parallel Processing Letters*, 30(01):2050005, 2020.

[58] Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election in a population protocol model. *Theoretical Computer Science*, 444:100–112, 2012.

[59] Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, Toshimitsu Masuzawa, Ajoy K Datta, and Lawrence L Larmore. Loosely-stabilizing leader election with

polylogarithmic convergence time. *Theoretical Computer Science*, 806:617–631, 2020.

[60] Prasad Tetali and Peter Winkler. On a random walk problem arising in self-stabilizing token management. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '91, pages 273–280, New York, NY, USA, 1991. ACM.

[61] Richard Weber. Optimal symmetric rendezvous search on three locations. *Mathematics of Operations Research*, 37(1):111–122, 2012.

[62] Hiroto Yasumi, Naoki Kitamura, Fukuhito Ooshita, Taisuke Izumi, and Michiko Inoue. A population protocol for uniform k-partition under global fairness. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 813–819. IEEE, 2018.

[63] Hiroto Yasumi, Fukuhito Ooshita, and Michiko Inoue. Uniform partition in population protocol model under weak fairness. *arXiv preprint arXiv:1911.04678*, 2019.

[64] Hiroto Yasumi, Fukuhito Ooshita, Ken'ichi Yamaguchi, and Michiko Inoue. Constant-space population protocols for uniform bipartition. In *21st International Conference on Principles of Distributed Systems (OPODIS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[65] Xiangdong Yu and Moti Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *International Colloquium on Automata, Languages, and Programming*, pages 610–621. Springer, 1996.