Dispersion of Mobile Robots in the Global Communication Model

Ajay D. Kshemkalyani
University of Illinois at Chicago
Illinois 60607, USA
ajay@uic.edu

Anisur Rahaman Molla
Indian Statistical Institute
Kolkata 700108, India
molla@isical.ac.in

Gokarna Sharma
Kent State University
Ohio 44240, USA
sharma@cs.kent.edu

ABSTRACT

The dispersion problem on graphs asks $k \leq n$ robots placed initially arbitrarily on the nodes of an n-node anonymous graph to reposition autonomously to reach a configuration in which each robot is on a distinct node of the graph. This problem is of significant interest due to its relationship to other fundamental robot coordination problems, such as exploration, scattering, load balancing, and relocation of self-driven electric cars (robots) to recharge stations (nodes). In this paper, we consider dispersion in the *global communication* model where a robot can communicate with any other robot in the graph (but the graph is unknown to robots). We provide three novel deterministic algorithms, two for arbitrary graphs and one for arbitrary trees, in a synchronous setting where all robots perform their actions in every time step. For arbitrary graphs, our first algorithm is based on a DFS traversal and guarantees $O(\min(m, k\Delta))$ steps runtime using $\Theta(\log(\max(k, \Delta)))$ bits at each robot, where m is the number of edges and Δ is the maximum degree of the graph. The second algorithm for arbitrary graphs is based on a BFS traversal and guarantees $O(\max(D, k)\Delta(D + \Delta))$ steps runtime using $O(\max(D, \Delta \log k))$ bits at each robot, where D is the diameter of the graph. The algorithm for arbitrary trees is also based on a BFS travesal and guarantees $O(D \max(D, k))$ steps runtime using $O(\max(D, \Delta \log k))$ bits at each robot. Our results are significant improvements compared to the existing results established in the *local communication* model where a robot can communication only with other robots present at the same node.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Computing methodologies → Distributed algorithms**; • **Computer systems organization → Robotics**.

KEYWORDS

Multi-agent systems, Mobile robots, Dispersion, Collective exploration, Scattering, Uniform deployment, Load balancing, Distributed algorithms, Time and memory complexity

ACM Reference Format:
Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. 2020. Dispersion of Mobile Robots in the Global Communication Model. In *21st International Conference on Distributed Computing and Networking (ICDCN 2020), January 4–7, 2020, Kolkata, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3369740.3369775

1 INTRODUCTION

The dispersion of autonomous mobile robots to spread them out evenly in a region is a problem of significant interest in distributed robotics, e.g., see [15, 16]. Recently, this problem has been formulated by Augustine and Moses Jr. [1] in the context of graphs. They defined the problem as follows: Given any arbitrary initial configuration of $k \leq n$ robots positioned on the nodes of an n-node anonymous graph, the robots reposition autonomously to reach a configuration where each robot is positioned on a distinct node of the graph (which we call the Dispersion problem). This problem has many practical applications, for example, in relocating self-driven electric cars (robots) to recharge stations (nodes), assuming that the cars have smart devices to communicate with each other to find a free/empty charging station [1, 17]. This problem is also important due to its relationship to many other well-studied autonomous robot coordination problems, such as exploration, scattering, load balancing, covering, and self-deployment [1, 17]. One of the key aspects of mobile-robot research is to understand how to use the resource-limited robots to accomplish some large task in a distributed manner [11, 12].

In this paper, we continue our study on the trade-off between memory requirement and time to solve Dispersion on graphs. We consider for the very first time the problem of dispersion in the *global communication* model where a robot can communicate with any other robot in the system (but the graph structure is not known to robots). The previous work [1, 17, 18] (details in Tables 1, 2, and related work) on Dispersion considered the *local communication* model where a robot can only communicate with other robots that are present at the same node. Although the global communication model seems stronger than the local model in the first sight, many challenges that occur in the local model also arise in the global model. For example, two robots in two neighboring nodes of G cannot figure out just by communication which edge of the nodes leads to each other. Therefore, the robots still need to explore through the edges as in the local model. The global communication model has been considered heavily in the past in distributed robotics, e.g., see [7, 13, 22], in addition to the local model, and our goal is to explore how much global communication helps for Dispersion in graphs compared to the local model.

In this paper, we provide three new deterministic algorithms for Dispersion in the global communication model, two for arbitrary

Algorithm	Memory/robot (in bits)	Time (in rounds)	Comm. Model/ Initial Conf.
Lower bound	$\Omega(\log(\max(k, \Delta)))$	$\Omega(k)$	local
[1][1]	$O(\log n)$	$O(mn)$	local/general
[17]	$O(k \log \Delta)$	$O(m)$	local/general
[17]	$O(D \log \Delta)$	$O(\Delta^D)$	local/general
[17]	$O(\log(\max(k, \Delta)))$	$O(mk)$	local/general
[18]	$O(\log n)$	$O(\min(m, k\Delta) \cdot \log k)$	local/general
Lower bound	$\Omega(\log(\max(k, \Delta)))$	$\Omega(k)$	global
Thm. 1.1	$O(\log(\max(k, \Delta)))$	$O(\min(m, k\Delta))$	global/general
Thm. 1.2(a)	$O(\max(D, \Delta \log k))$	$O(D\Delta(D + \Delta))$	local/rooted
Thm. 1.2(b)	$O(\max(D, \Delta \log k))$	$O(\max(D, k)\Delta(D + \Delta))$	global/general

Table 1: The results on Dispersion **for $k \leq n$ robots on n-node arbitrary graphs with m edges, D diameter, and Δ maximum degree. [1]The results in [1] are only for $k = n$.**

Algorithm	Memory/robot (in bits)	Time (in rounds)	Comm. Model/ Initial Conf.
Lower bound	$\Omega(\log(\max(k, \Delta)))$	$\Omega(D^2)$	local
[1]	$O(\log n)$	$O(n)$	local/general
Lower bound	$\Omega(\log(\max(k, \Delta)))$	$\Omega(D^2)$	global
Thm. 1.3(a)	$O(\max(D, \Delta \log k))$	$O(D^2)$	local/rooted
Thm. 1.3(b)	$O(\max(D, \Delta \log k))$	$O(D \max(D, k))$	global/general

Table 2: The results on Dispersion **for $k \leq n$ robots on n-node trees.**

graphs and one for arbitrary trees. Our first algorithm for arbitrary graphs using a *depth first search* (DFS) traversal improves by a $O(\log n)$ factor on the state-of-the-art in the local communication model; see Table 1. The second algorithm for arbitrary graphs and the algorithm for arbitrary trees are the first algorithms designed for Dispersion using a *breadth first search* (BFS) traversal and provide different time-memory trade-offs. We also complement our algorithms by some lower bounds on time and memory requirement in the global model.

Overview of the Model and Results. We consider the same model (with a only difference as described in the next paragraph) as in Augustine and Moses Jr. [1], Kshemkalyani and Ali [17], and Kshemkalyani *et al.* [18] where a system of $k \leq n$ robots are operating on an n-node graph G. G is assumed to be a connected, undirected graph with m edges, diameter D, and maximum degree Δ. In addition, G is *anonymous*, i.e., nodes have no unique IDs and hence are indistinguishable but the ports (leading to incident edges) at each node have unique labels from $[1, \delta]$, where δ is the degree of that node. The robots are *distinguishable*, i.e., they have unique IDs in the range $[1, k]$. The robot activation setting is *synchronous* – all robots are activated in a round and they perform their operations simultaneously in synchronized rounds. Runtime is measured in rounds (or steps).

The only difference with the model in [1, 17, 18] is they assume the local communication model – the robots in the system can communicate with each other only when they are at the same node of G, whereas we consider in this paper the global communication model – the robots in the system can communicate with each other irrespective of their positions. Despite this capability, robots are still oblivious to G and they will not know the positions of the robots that they are communicating to.

We establish the following two results for Dispersion in an arbitrary graph. The second result differentiates the initial configurations of $k \leq n$ robots on G. We call the configuration *rooted* if all $k \leq n$ robots are on a single node of G in the initial configuration. We call the initial configuration *general*, otherwise.

Theorem 1.1. *Given any initial configuration of $k \leq n$ mobile robots in an arbitrary, anonymous n-node graph G having m edges and maximum degree Δ,* Dispersion *can be solved in $O(\min(m, k\Delta))$ time with $O(\log(\max(k, \Delta)))$ bits at each robot in the global communication model.*

Theorem 1.2. *Given $k \leq n$ mobile robots in an arbitrary, anonymous n-node graph G having m edges, diameter D, and maximum degree Δ:*

a. *For the rooted initial configurations,* Dispersion *can be solved in $O(D\Delta(D + \Delta))$ time with $O(\max(D, \Delta \log k))$ bits at each robot in the local communication model.*
b. *For the general initial configurations,* Dispersion *can be solved in $O(\max(D, k)\Delta(D + \Delta))$ time with $O(\max(D, \Delta \log k))$ bits at each robot in the global communication model.*

Theorem 1.1 improves by a factor of $O(\log k)$ over the $O(\min(m, k\Delta) \cdot \log k)$ time best previously known algorithm [18] in the local communication model (see Table 1). We also prove a time lower bound of $\Omega(k)$ and memory lower bound of $\Omega(\log(\max(k, \Delta)))$ bits at each robot for Dispersion on graphs in the global communication model. The implication is that, for constant-degree arbitrary graphs (i.e., when $\Delta = O(1)$), Theorem 1.1 is optimal with respect to both memory and time, first such result for arbitrary graphs. Theorem 1.2 improves significantly on the $O(\Delta^D)$ algorithm in the local communication model.

We establish the following theorem in an arbitrary tree.

Theorem 1.3. *Given $k \leq n$ mobile robots in an anonymous n-node arbitrary tree G with degree Δ and diameter D:*

a. *For the rooted initial configurations,* Dispersion *can be solved in $O(D^2)$ time with $O(\max(D, \Delta \log k))$ bits at each robot in the local communication model.*
b. *For the general initial configurations,* Dispersion *can be solved in $O(D \max(D, k))$ time with $O(\max(D, \Delta \log k))$ bits at each robot in the global communication model.*

We also prove a time lower bound of $\Omega(D^2)$ for Dispersion on trees in the global communication model. The implication is that the time in Theorem 1.3(a) is optimal. Also, the time in Theorem 1.2(a) is optimal for constant-degree arbitrary graphs.

Challenges and Techniques. The well-known DFS traversal approach [5] was used in the previous results on Dispersion [1, 17, 18]. If all k robots are positioned initially on a single node of G, then the DFS traversal finishes in $\min(4m - 2n + 2, k\Delta)$ rounds solving Dispersion. If k robots are initially on k different nodes of G, then Dispersion is solved in a single round. However, if not all of them are on a single node, then the robots on nodes with multiple robots need to reposition to reach to free nodes and settle. The natural approach is to run DFS traversals in parallel to minimize time.

The challenge arises when two or more DFS traversals meet before all robots settle. When this happens, the robots that have not settled yet need to find free nodes. For this, they may need to re-traverse the already traversed part of the graph by the DFS traversal. Kshemkalyani *et al.* [18] designed a smarter way to synchronize the parallel DFS traversals so that the total time increases only by a factor of $\log k$ to $\min(4m-2n+2, k\Delta)\cdot\log k$ rounds, in the worst-case, in the local communication model. However, removing the $O(\log k)$ factor seemed difficult due to the means of synchronization. We develop in this paper an approach that allows to synchronize DFS traversals without re-traversing the already traversed part of the graph giving us $\min(4m-2n+2,\ k\Delta)$ rounds, as if running DFS starting from all robots in the same node, in the global communication model. This is possible due to the information that can be passed to the robots to take their next actions, even if they do not known their positions on G. This was not possible in the local communication model and hence the synchronization incurred an $O(\log k)$ factor to synchronize $O(k)$ trees that might be formed in the dispersion process. For constant-degree arbitrary graphs, the time bound becomes $O(k)$, which is time-optimal.

Despite efficiency in merging the DFS traversal trees due to global communication, the time bound of $\min(4m-2n+2,\ k\Delta)$ seems to be inherent in algorithms based on a DFS traversal, even in the global model (consider, for example, a rooted initial configuration). The natural way to circumvent this limitation is to run BFS traversal to reach many nodes at once. A naive approach of running BFS gives exponential $O(\Delta^D)$ runtime. Here we design a smarter way of performing BFS so that we can achieve dispersion in arbitrary graphs in $O(D\Delta(D+\Delta))$ time and in arbitrary trees in $O(D^2)$ time in the rooted initial configurations. The general initial configurations introduced $\max(D,k)$ factor instead of $O(D)$ factor in the time bounds for both arbitrary graphs and trees.

Related Work. There are three previous studies focusing on Dispersion in the local communication model. Augustine and Moses Jr. [1] studied Dispersion assuming $k=n$. They proved a memory lower bound of $\Omega(\log n)$ bits at each robot and a time lower bound of $\Omega(D)$ ($\Omega(n)$ in arbitrary graphs) for any deterministic algorithm in any graph. They then provided deterministic algorithms using $O(\log n)$ bits at each robot to solve Dispersion on lines, rings, and trees in $O(n)$ time. For arbitrary graphs, they provided two algorithms, one using $O(\log n)$ bits at each robot with $O(mn)$ time and another using $O(n\log n)$ bits at each robot with $O(m)$ time.

Kshemkalyani and Ali [17] provided an $\Omega(k)$ time lower bound for arbitrary graphs for $k\le n$. They then provided three deterministic algorithms for Dispersion in arbitrary graphs: (i) The first algorithm using $O(k\log\Delta)$ bits at each robot with $O(m)$ time, (ii) The second algorithm using $O(D\log\Delta)$ bits at each robot with $O(\Delta^D)$ time, and (iii) The third algorithm using $O(\log(\max(k,\Delta)))$ bits at each robot with $O(mk)$ time. Recently, Kshemkalyani *et al.* [18] provided two algorithms: (i) the algorithm for arbitrary graph runs in $O(\min(m,k\Delta)\cdot\log k)$ time using $O(\log(\max(k,\Delta)))$ bits memory at each robot and (ii) the algorithm for grid graph runs in $O(\min(k,\sqrt{n}))$ time using $O(\log k)$ bits memory at each robot. Randomized algorithms are presented in [21] to solve Dispersion where the random bits are mainly used to reduce the memory requirement at each robot. In this paper,we present results in the global communication model. The previous results on arbitrary graphs and trees are summarized in Table 1.

One problem that is closely related to Dispersion is the graph exploration by mobile robots. The exploration problem has been quite heavily studied in the literature for specific as well as arbitrary graphs, e.g., [2, 4, 9, 14, 20]. It was shown that a robot can explore an anonymous graph using $\Theta(D\log\Delta)$-bits memory; the runtime of the algorithm is $O(\Delta^{D+1})$ [14]. In the model where graph nodes also have memory, Cohen *et al.* [4] gave two algorithms: The first algorithm uses $O(1)$-bits at the robot and 2 bits at each node, and the second algorithm uses $O(\log\Delta)$ bits at the robot and 1 bit at each node. The runtime of both algorithms is $O(m)$ with preprocessing time of $O(mD)$. The trade-off between exploration time and number of robots is studied in [20]. The collective exploration by a team of robots is studied in [13] for trees. Another problem related to Dispersion is the scattering of k robots in an n-node graph. This problem has been studied for rings [10, 24] and grids [3]. Recently, Poudel and Sharma [23] provided a $\Theta(\sqrt{n})$-time algorithm for uniform scattering in a grid [8]. Furthermore, Dispersion is related to the load balancing problem, where a given load at the nodes has to be (re-)distributed among several processors (nodes). This problem has been studied quite heavily in graphs, e.g., see [6]. We refer readers to [11, 12] for other recent developments in these topics.

Paper Organization. We discuss details of the model and some lower bounds in Section 2. We discuss the DFS traversal of a graph in Section 3. We present a DFS-based algorithm for arbitrary graphs in Section 4, proving Theorem 1.1. We then present a BFS-based algorithm for arbitrary graphs in Section 5, proving Theorem 1.2. We then present a BFS-based algorithm for arbitrary trees in Section 6, proving Theorem 1.3. Finally, we conclude in Section 7 with a short discussion. Due to space constraints, one algorithm and many proofs are deferred to the full version in Arxiv [19].

2 MODEL DETAILS AND PRELIMINARIES

Graph. We consider the same graph model as in [1, 17]. Let $G=(V,E)$ be an n-node graph with m edges, i.e., $|V|=n$ and $|E|=m$. G is assumed to be connected, unweighted, and undirected. G is *anonymous*, i.e., nodes do not have identifiers but, at any node, its incident edges are uniquely identified by a *label* (aka port number) in the range $[1,\delta]$, where δ is the *degree* of that node. The *maximum degree* of G is Δ, which is the maximum among the degree δ of the nodes in G. We assume that there is no correlation between two port numbers of an edge. Any number of robots are allowed to move along an edge at any time. The graph nodes do not have memory, i.e., they are not able to store any information.

Robots. We also consider the same robot model as in [1, 17, 18]. Let $\mathcal{R}=\{r_1,r_2,\ldots,r_k\}$ be a set of $k\le n$ robots residing on the nodes of G. For simplicity, we sometime use i to denote robot r_i. No robot can reside on the edges of G, but one or more robots can occupy the same node of G. Each robot has a unique $\lceil\log k\rceil$-bit ID taken from $[1,k]$. When a robot moves from node u to node v in G, it is aware of the port of u it used to leave u and the port of v it used to enter v. Furthermore, it is assumed that each robot is equipped with memory to store information, which may also be read and modified by other robots present on the same node.

Communication Model. We assume that robots follow the global communication model, i.e., a robot is capable to communicate with any other robot in the system, irrespective of their positions in the graph nodes. However, they will not have the position information as graph nodes are anonymous. This is in contrast to the local communication model where a robot can only communicate with other robots present on the same node.

Time Cycle. At any time a robot $r_i \in \mathcal{R}$ could be active or inactive. When a robot r_i becomes active, it performs the "Communicate-Compute-Move" (CCM) cycle as follows.

- *Communicate:* For each robot $r_j \in \mathcal{R}$ that is at node some node v_j, a robot r_i at node v_i can observe the memory of r_j. Robot r_i can also observe its own memory.
- *Compute:* r_i may perform an arbitrary computation using the information observed during the "communicate" portion of that cycle. This includes determination of a (possibly) port to use to exit v_i and the information to store in the robot r_j that is at v_i.
- *Move:* At the end of the cycle, r_i writes new information (if any) in the memory of a robot r_k at v_i, and exits v_i using the computed port to reach to a neighbor of v_i.

Time and Memory Complexity. We consider the synchronous setting where every robot is active in every CCM cycle and they perform the cycle in a synchrony. Therefore, time is measured in *rounds* or *steps* (a cycle is a round or step). Another important parameter is memory. Memory comes from a single source – the number of bits stored at each robot.

Mobile Robot Dispersion. The Dispersion problem can be formally defined as follows.

*Definition 2.1 (*Dispersion*).* Given any n-node anonymous graph $G = (V, E)$ having $k \leq n$ mobile robots positioned initially arbitrarily on the nodes of G, the robots reposition autonomously to reach a configuration where each robot is on a distinct node of G.

The goal is to solve Dispersion optimizing two performance metrics: (i) **Time** – the number of rounds (steps), and (ii) **Memory** – the number of bits stored at each robot.

2.1 Some Lower Bounds

We discuss here some time and memory lower bounds in the global communication model, which show the difficulty in obtaining fast runtime and lower memory algorithms. Consider the case of any rooted initial configuration of $k = n$ robots on a single node v_{root} of an arbitrary graph G with diameter D. A time lower bound of $\Omega(D)$ is immediate since a robot initially at v_{root} needs to traverse $\Omega(D)$ edges (one edge per time step) to reach a node that is $\Omega(D)$ away from v_{root}. For $k \leq n$, we present the following lower bound.

Theorem 2.2. *Any deterministic algorithm for* Dispersion *on graphs requires $\Omega(k)$ steps in the global communication model.*

For $k = n$, we present the following time lower bound for trees.

Theorem 2.3. *For $k = n$, there exists a tree T with n nodes and diameter (height) D such that any deterministic algorithm for* Dispersion *requires $\Omega(D^2)$ steps in the global communication model.*

We finally prove a lower bound of $\Omega(\log(\max(k, \Delta)))$ bits at each robot for any deterministic algorithm for Dispersion on graphs.

Theorem 2.4. *Any deterministic algorithm for* Dispersion *on n-node anonymous graphs requires $\Omega(\log(\max(k, \Delta)))$ bits at each robot in the global communication model, where $k \leq n$ is the number of robots and Δ is the maximum degree.*

3 DFS TRAVERSAL OF A GRAPH (ALGORITHM $DFS(k)$)

Consider an n-node arbitrary graph G as defined in Section 2. Let C_{init} be the initial configuration of $k \leq n$ robots positioned on a single node, say v, of G. Let the robots on v be represented as $N(v) = \{r_1, \ldots, r_k\}$, where r_i is the robot with ID i. We describe here a DFS traversal algorithm, $DFS(k)$, that disperses all the robots on the set $N(v)$ to the k nodes of G guaranteeing exactly one robot on each node. $DFS(k)$ will be heavily used in Section 4 as a basic building block.

Each robot r_i stores in its memory four variables $r_i.parent$ (initially assigned *null*), $r_i.child$ (initially assigned *null*), $r_i.treelabel$ (initally assigned $\top$), and $r_i.settled$ (initially assigned 0). $DFS(k)$ executes in two phases, $forward$ and $backtrack$ [5]. Variable $r_i.treelabel$ stores the ID of the smallest ID robot. Variable $r_i.parent$ stores the port from which r_i entered the node where it is currently positioned in the forward phase. Variable $r_i.child$ stores the smallest port of the node it is currently positioned at that has not been taken yet (while entering/exiting the node).

We are now ready to describe $DFS(k)$. In round 1, the maximum ID robot r_k writes $r_k.treelabel \leftarrow 1$ (the ID of the smallest robot in $N(v)$, which is 1), $r_k.child \leftarrow 1$ (the smallest port at v among $P(v)$), and $r_k.settled \leftarrow 1$. The robots $N(v)\backslash\{r_k\}$ exit v following port $r_k.child$; r_k stays (settles) at v. In the beginning of round 2, the robots $N(w) = N(v)\backslash\{r_k\}$ reach a neighbor node w of v. Suppose the robots entered w using port $p_w \in P(w)$. As w is free, robot $r_{k-1} \in N(w)$ writes $r_{k-1}.parent \leftarrow p_w$, $r_{k-1}.treelabel \leftarrow 1$ (the ID of the smallest robot in $N(w)$), and $r_{k-1}.settled \leftarrow 1$. If $r_{k-1}.child \leq \delta_w$, r_{k-1} writes $r_{k-1}.child \leftarrow r_{k-1}.child + 1$ if port $r_{k-1}.child + 1 \neq p_w$ and $r_{k-1}.child + 1 \leq \delta_w$, otherwise $r_{k-1}.child \leftarrow r_{k-1}.child + 2$. The robots $N(w)\backslash\{r_{k-1}\}$ decide to continue DFS in forward or backtrack phase as described below.

- (**forward phase**) if ($p_w = r_{k-1}.parent$ or p_w = old value of $r_{k-1}.child$) and (there is (at least) a port at w that has not been taken yet). The robots $N(w)\backslash\{r_{k-1}\}$ exit w through port $r_{k-1}.child$.
- (**backtrack phase**) if ($p_w = r_{k-1}.parent$ or p_w = old value of $r_{k-1}.child$) and (all the ports of w have been taken already). The robots $N(w)\backslash\{r_{k-1}\}$ exit w through port $r_{k-1}.parent$.

Assume that in round 2, the robots decide to proceed in forward phase. In the beginning of round 3, $N(u) = N(w)\backslash\{r_{k-1}\}$ robots reach some other node u (neighbor of w) of G. The robot r_{k-2} stays at u writing necessary information in its variables. In the forward phase in round 3, the robots $N(u)\backslash\{r_{k-2}\}$ exit u through port $r_{k-2}.child$. However, in the backtrack phase in round 3, r_{k-2} stays at u and robots $N(u)\backslash\{r_{k-2}\}$ exit u through port $r_{k-2}.parent$. This takes robots $N(u)\backslash\{r_{k-2}\}$ back to node w along $r_{k-1}.child$. Since r_{k-1} is already at w, r_{k-1} updates $r_{k-1}.child$ with the next port to take. Depending on whether $r_i.child \leq \delta_w$ or not, the robots $\{r_1, \ldots, r_{k-3}\}$ exit w using either $r_{k-1}.child$ (forward phase) or $r_{k-1}.parent$ (backtrack phase).

There is another condition, denoting the onset of a cycle, under which choosing backtrack phase is in order. When the robots enter x through p_x and robot r is settled at x,

- **(backtrack phase)** if ($p_x \neq r.parent$ and $p_x \neq$ old value of $r.child$). The robots exit x through port p_x and no variables of r are altered.

This process then continues for $DFS(k)$ until at some node $y \in G$, $N(y) = \{r_1\}$. The robot r_1 then stays at y and $DFS(k)$ finishes.

Lemma 3.1. *Algorithm* $DFS(k)$ *correctly solves* Dispersion *for* $k \leq n$ *robots initially positioned on a single node of a* n*-node arbitrary graph* G *in* $\min(4m - 2n + 2, k\Delta)$ *rounds using* $O(\log(\max(k, \Delta)))$ *bits at each robot.*

4 ALGORITHM FOR ARBITRARY GRAPHS (THEOREM 1.1)

We present and analyze *Graph_Disperse_DFS*, a DFS-based algorithm that solves Dispersion of $k \leq n$ robots on an arbitrary n-node graph in $O(\min(m, k\Delta))$ time with $O(\log(\max(k, \Delta)))$ bits of memory at each robot in the global communication model. This algorithm improves the $O(\min(m, k\Delta) \cdot \log k)$ time of the best previously known algorithm [17] for arbitrary graphs (Table 1) in the local communication model.

4.1 The Algorithm

The algorithm is based on DFS traversal. In general, a robot may operate in one of two interchangeable phases: GROW and COLLECT. As these are independent, a separate set of DFS variables: *parent*, *child*, is used for operating in the two phases. The following additional variables are used. (i) *TID_Grow*: Tree ID, of type robot identifier, is the ID of the DFS tree in the GROW phase with which the robot is associated. Initially, *TID_Grow* $\leftarrow$ minimum ID among the colocated robots. (ii) *TID_Collect*: Tree ID, of type robot identifier, is the ID of the DFS tree in the COLLECT phase with which the robot is associated. Initially, *TID_Collect* $\leftarrow\perp$. (iii) *CID*: for component ID, of type robot identifier, is used to denote the component associated with the GROW phase of the DFS. Initially, *CID* $\leftarrow$ *TID_Grow*. (iv) *CID_old*: for earlier component ID, of type robot identifier, is used to denote the earlier value of component ID just before the most recent component ID (*CID*) update, associated with the GROW phase of the DFS. Initially, *CID_old* $\leftarrow$ *TID_Grow*. (v) *winner*: of type robot identifier. When multiple components collide/merge, this is used to indicate the winning component ID that will subsume the other components. Initially, *winner* $\leftarrow\perp$. (vi) *leader*: of type boolean. This is set to 1 if the robot is responsible for collecting the various robots distributed in the component. Initially, *leader* $\leftarrow 0$. (vii) *home*: of type robot identifier. The robot identifier of a settled robot is used to identify the origin node of the leader robot that is responsible for collecting the scattered robots in the component back to this origin node. Initially, *home* $\leftarrow\perp$. (viii) *state*: denotes the state of the robot and can be one of $\{grow, collect, subsumed, settled\}$. Initially, *state* $\leftarrow$ *grow*.

In the initial configuration, there are groups of robots at different nodes. Each robot has its *TID_Grow* set to the minimum ID among the colocated robots, and its *state* = *grow*. The robots from a node move together in a DFS traversal, to locate free nodes and settle one by one. As they do the DFS traversal *Grow*(*TID_Grow*), they extend the DFS tree that is associated with the *TID_Grow*. Each growing DFS tree is also associated with a component ID, *CID*, that is initialized to the *TID_Grow*. Multiple DFS trees associated with

Algorithm 1: Algorithm *Graph_Disperse_DFS* to solve Dispersion in global model. Code for robot i in a round at any node. r denotes a settled robot, if any, at that node.

```
Initialize: $TID\_Grow, TID\_Collect, CID, CID\_old, winner,$
  $leader, home, state$
if $state = grow$ then
    if node is free then
        highest ID robot $x$ from highest $CID$ group having
          $state = grow$ settles; $x.state \leftarrow settled$
    if $CID \neq r.CID$ then
        one such robot from each $CID$ group broadcasts
          $Subsume(CID, r.CID)$
    Subsume_Graph_Processing
    if $CID$ is node in Subsume graph then
        $CID\_old \leftarrow CID$; $CID \leftarrow winner$ in my component of
          Subsume graph
        Let $x \leftarrow \min_j(j.TID\_grow = winner \wedge j.state =$
          $grow \wedge j$ is at same node as $i$)
        $x.home \leftarrow r.ID$; $x.leader \leftarrow 1$; $x.state \leftarrow collect$;
          $x.TID\_Collect \leftarrow x.TID\_Grow$
        $x$ begins DFS $Collect(TID\_Collect)$
        if $i \neq x \wedge state = grow$ then
            $state \leftarrow subsumed$; STOP
    else
        continue DFS $Grow(TID\_Grow)$
else if $state = collect$ then
    Subsume_Graph_Processing
    if $CID$ is node in Subsume graph then
        $CID\_old \leftarrow CID$; $CID \leftarrow winner$ in my component of
          Subsume graph
        $state \leftarrow subsumed$; STOP
        if $leader = 1$ then
            $leader \leftarrow 0$; $home \leftarrow\perp$
    else
        if node is free
          $\vee CID = r.CID = r.CID\_old \vee CID \neq r.CID$ then
            backtrack, as part of DFS $Collect(TID\_Collect)$
        else if $CID = r.CID \neq r.CID\_old$ then
            if $\exists x \mid x.leader = 1 \wedge x.home = r.ID \wedge$ all ports at $r$
              have been explored then
                if $x = i$ then
                    $x.leader \leftarrow 0; x.home \leftarrow\perp$
                $state \leftarrow grow$; $TID\_Grow \leftarrow x.TID\_Grow$
                $i$ continues DFS $Grow(TID\_Grow)$
            else
                $i$ continues DFS $Collect(TID\_Collect)$ of
                  $x \mid x.leader = 1$, along with $x$ if $x$ is backtracking
                  to its parent in DFS $Collect(x.TID\_Collect)$
```

different *CID*s may meet at a node in any round; specifically, a DFS tree for component *CID* may meet another component *r.CID* for some other DFS tree, where r is the robot that is settled at that node. In this case, one robot from the newly arrived robots of the DFS tree component *CID* broadcasts a *Subsume*(*CID*, *r.CID*) message. This is to indicate that the component *CID* is subsuming the component

```
35 else if $state = subsumed$ then
36     Subsume_Graph_Processing
37     if $CID$ is node in Subsume graph then
38         $CID\_old \leftarrow CID$; $CID \leftarrow winner$ in my component of Subsume graph
39     else
40         if $\exists$ arrived robot $x \mid x.state = collect \wedge x.leader = 1 \wedge$ $x$ is backtracking to its parent in DFS $Collect(TID\_Collect)$ then
41             $state \leftarrow collect$; $TID\_Collect \leftarrow x.TID\_Collect$
42             if $x.home = r.ID \wedge$ all ports at $r$ have been explored then
43                 $state \leftarrow grow$; $TID\_Grow \leftarrow x.TID\_Grow$
44                 $i$ continues DFS $Grow(TID\_Grow)$ along with $x$
45             else
46                 $i$ continues DFS $Collect(TID\_Collect)$ along with $x$
47 else if $state = settled$ then
48     Subsume_Graph_Processing
49     if $CID$ is node in Subsume graph then
50         $CID\_old \leftarrow CID$; $CID \leftarrow winner$ in my component of Subsume graph
51 Subsume_Graph_Processing
52 receive Subsume messages; build Subsume graph $S$
53 if node with no incoming edge in my component of $S$ exists then
54     $winner \leftarrow \min_{CID}(CID$ of nodes with no incoming edge in my component of $S)$
55 else
56     $winner \leftarrow \min_{CID}$(cycle of CIDs in my component)
```

$r.CID$. Multiple such *Subsume* messages may get broadcast from different robots in the graph in any particular round.

All the robots listen to all such broadcasts in each round, and build a directed graph, *Subsume*, $S = (C, E)$, where C is the set of component IDs, and edge (CID_j, CID_k) indicates that $Subsume(CID_j, CID_k)$ message has been received. In this graph, each node may have at most one outgoing edge but may have multiple incoming edges. The *winner* component ID corresponds to that node (in my connected component of S) that has the minimum CID among the nodes with no incoming edges (if such a node exists). Otherwise, there must exist a cycle with no incoming edges in the *Subsume* graph, and the lowest valued CID node in the cycle is chosen as *winner*. The significance of the *winner* is that its CID subsumes all other CIDs in its connected component of S; that is, all robots that are in the same connected component of S overwrite their current CID by *winner* in their connected component of S.

The robot with the minimum ID among those with $TID\_Grow = winner$ and $state = grow$ changes its *state* to *collect*, *leader* to 1, $TID\_Collect$ to TID_Grow, and embarks on the *Collect* phase. In the *Collect* phase, the leader does an independent DFS traversal $Collect(TID\_Collect)$ of the connected component of settled nodes of G which have settled robots which have newly changed their component ID CID to be the same as its own. And all (unsettled) robots which have newly changed their CID to that of the *winner* leader, or have $CID = winner$ but are not the leader, also change their *state*, whether *grow* or *collect*, to *subsumed* and stop movement until they are collected. In this DFS traversal *Collect*, the leader node collects all unsettled robots with $state = subsumed$ and brings them back to its home node from where it began the *Collect* DFS traversal, while the thus collected robots change their *state* to *collect* once they join the collection traversal. During the *Collect* traversal, if in some step the component gets subsumed by some other component, the unsettled robots reset their *state* to *subsumed*. If the *Collect* DFS traversal completes successfully, the collected robots and the leader change *state* to *grow*, set their TID_Grow to that of the leader, and resume DFS $Grow(TID\_Grow)$ after the leader resets its leader status.

Note that the DFS $Collect(TID\_Collect)$ is independent of the DFS $Grow(TID\_Grow)$, and thus an independent set of variables *parent*, *child*, need to be used for the two different types of DFSs. Further, when a new instance of a DFS *Grow*/DFS *Collect*, as identified by a new value of TID_Grow/ $TID\_Collect$, is detected at a settled robot (node), the robot switches to the new instance and resets the old values of *parent* and *child* for that DFS search.

In the DFS *Collect* phase, the leader visits all nodes in its connected component of settled nodes having a settled robot that changed its component ID $r.CID \leftarrow CID$. (These are the settled robots where $CID = r.CID \neq r.CID\_old$, where CID_old is the value of CID before the latest overwrite by *winner*.) This excludes the nodes already visited in the DFS *Grow* phase having settled robots with the same CID as that of the leader before it become the leader. To confine the DFS *Collect* to such nodes, note that the leader may have to backtrack from a node v if the node (i) is free or (ii) has $CID = r.CID = r.CID\_old$ or (iii) has $CID \neq r.CID$. If the CID of the leader changes at the beginning of this round (because it gets subsumed), before it can backtrack, the leader (and any accompanying robots having $state = collect$) simply changes *state* to *subsume* and stops. In cases (i) and (iii), there may thus be stopped robots at a free node, or at a node that belongs to an adjoining, independent component. Such robots may be later collected by (a) a leader from its old component, or (perhaps earlier than that) (b) by a leader from the component where they stop. In the former case (a), it is execution as usual. In the latter case (b), there is no issue of violating correctness even though the robots jump from one connected component sharing a common CID to an adjacent one with a different CID.

4.2 Correctness and Complexity

A robot may be in one of four states: *grow*, *collect*, *subsumed*, and *settled*. The state transition diagram for a robot is shown in Figure 1(a).

Lemma 4.1. *Once a robot enters state = grow for some value of TID_Grow, the DFS $Grow(TID\_Grow)$ completes within* $\min(4m - 2n + 2, 4k\Delta)$ *rounds, or the robot moves out of that state within* $\min(4m - 2n + 2, 4k\Delta)$ *rounds.*

Proof. A DFS (using $\log \Delta$ bits at a robot) completes within $4m - 2n + 2$ rounds. It also completes within $4k\Delta$ rounds, as in the DFS, each edge gets traversed a maximum of 4 times, and as at most k nodes need to be visited before all k robots get settled. Before this completion of the DFS, if the current component gets subsumed or

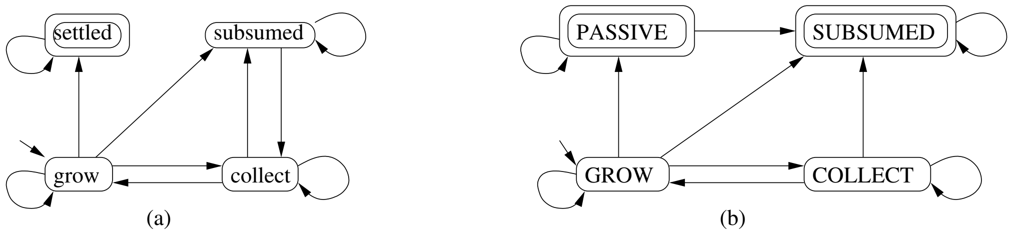

Figure 1: State transition diagrams. (a) Diagram for a robot's state, *state*. (b) Diagram for any value of *CID*.

subsumes another component, the robot moves to either *subsumed* or *collect* state. □

Lemma 4.2. *Once a robot enters state = collect for some value of TID_Collect, the DFS Collect(TID_Collect) completes in* $\min(4m - 2n + 2, 4k\Delta)$ *rounds or the robot moves out of that state within* $\min(4m - 2n + 2, 4k\Delta)$ *rounds.*

Proof. A DFS traversal of a component (using $\log \Delta$ bits at a robot) completes within $4m-2n+2$ rounds. It also completes within $4k\Delta$ rounds, as the collecting robot in the DFS traverses an edge at most 4 times, needs to visit each of the at most Δ neighbors of the at most k settled nodes in the component, until collection completes and the leader is back at the home node. (At the completion of the DFS, the robot moves to *grow* state; before this completion of the DFS, if the current component gets subsumed by another component, the robot moves to *subsumed* state.) □

Theorem 4.3. *The algorithm Graph_Disperse_DFS solves* Dispersion.

Proof. Each robot begins in $state = grow$. We make the following observations about the state transition diagram of a robot.

(1) A robot can enter *subsumed* state at most $k - 1$ times. In *subsumed* state, a robot can stay at most $\min(4m - 2n + 2, 4k\Delta) \cdot k$ rounds before it changes *state* to *collect*.
(2) From *collect* state, within $\min(4m - 2n + 2, 4k\Delta)$ rounds, a robot can go to *subsumed* state (which can happen at most $k - 1$ times), or go to *grow* state (Lemma 4.2).
(3) In *grow* state, a robot can remain for at most $\min(4m - 2n + 2, 4k\Delta)$ rounds, by when it may go to either *subsumed* or *collect* state for at most $k - 1$ times, or go to *settled* state (Lemma 4.1).

It then follows that within a finite, bounded number of rounds, a robot will be in *grow* state for the last time and within $\min(4m - 2n+2, 4k\Delta)$ further rounds, settle and go to *settled* state. This is and will be the only robot in *settled* state at the node. Thus, Dispersion is achieved within a finite, bounded number of rounds. □

We model the state of a particular value of *CID*. It can be either GROW, COLLECT, PASSIVE, or SUBSUMED. The state transition diagram for the state of a *CID* value is shown in Figure 1(b).

Theorem 4.4. *The algorithm Graph_Disperse_DFS terminates in* $\min(2 \cdot 4m, 2 \cdot 4k\Delta)$ *rounds.*

Theorem 4.5. *Algorithm 1 (*Graph_Disperse_DFS*) requires* $O(\log(\max(k, \Delta)))$ *bits memory.*

Proof. Each set of *parent*, *child*, *treelabel*, *settled* for the GROW and COLLECT phases takes $O(\log(\max(k, \Delta)))$ bits (follows from Theorem 3.1). *CID*, *CID_old*, *winner*, *TID_Grow*, *TID_Collect*, and *home* take $O(\log k)$ bits each. *leader* and *state* take $O(1)$ bits each. Thus, the theorem follows. □

Proof of Theorem 1.1: Follows from Theorems 4.3 – 4.5.

5 ALGORITHM FOR ARBITRARY GRAPHS (THEOREM 1.2)

We present and analyze *Graph_Disperse_BFS*, a BFS-based algorithm that solves Dispersion of $k \leq n$ robots on an arbitrary n-node graph in $O(\max(D, k)\Delta(D + \Delta))$ time with $O(\max(D, \Delta \log k))$ bits of memory at each robot in the global communication model. This algorithm improves the $O(\Delta^D)$ time of the best previously known algorithm [17] for arbitrary graphs (Table 1) in the local communication model.

We first discuss the rooted graph case, wherein all robots are on a single node initially. Here, we show that dispersion takes $O(D\Delta(D + \Delta))$ time with $O(\max(D, \Delta \log k))$ bits of memory at each robot in the local communication model. We then extend the result to the general case of robots on multiple nodes.

5.1 Rooted Case

In the initial configuration, all $k \leq n$ robots are at a single node v_{root}. The synchronous algorithm proceeds in rounds and is described in Algorithm 2. The algorithm induces a breadth-first search (BFS) tree, level by level, in the graph. There are two main steps in the algorithm when extending the tree from i levels to $i + 1$ levels: (i) the leaf nodes in the BFS tree at level i determine the number of edges going to level $i + 1$. This is done in procedure `DetermineLeafDemand(i)` and can be achieved in $O(\Delta^2)$ rounds as a 2-neighborhood traversal is performed. The level i robot sets its demand for robots equal to the number of edges (ports) going to level $i + 1$. (ii) The leaf nodes at level i then populate the level $i+1$ nodes in a coordinated manner, because there may be arbitrary number of edges and connectivity going from level i to level $i + 1$. This is done in procedure `PopulateNextLevel(i)` iteratively by borrowing robots for exploration from v_{root}. The number of robots assigned by the root may be up to the demand of that node. This movement takes $O(D)$ time. As there may be edges from multiple nodes at level i to a node at level $i + 1$, only one robot can be earmarked to settle at that node; others retrace their steps back to the root. The robot earmarked to settle at the level $i + 1$ node does a 1-neighborhood traversal and *invalidates* the ports of level i nodes leading to that level $i + 1$ node ($O(\Delta)$ time). The robot does not actually settle at the level $i + 1$ node but participates in further computation. It then returns to the level i node it arrived from and designates the port used to go to the level $i + 1$ node as a *valid* port. The settled robots at level i then re-evaluate the demand for robots, based on the number of *unfinalized* ports (i.e., not validated and not invalidated ports going to level $i + 1$ nodes). All unsettled robots (including those that had been "earmarked" to settle at a level $i + 1$ node) return to v_{root} and they are reassigned for the next iteration based on the renewed (and decreased) valued of net demand for exploratory robots. This movement takes $O(D)$ time. The algorithm guarantees that Δ iterations suffice for determining the *valid/invalid* status of all ports of level i nodes leading to level $i + 1$ nodes, after which a final iteration reassigns the final demand based on the number of *valid* ports (each of which leads to a unique level $i + 1$ node) and distributes up to those many robots among level $i+1$ nodes. The procedure `PopulateNextLevel(i)` thus takes $O(\Delta(\Delta + D))$ time.

Due to the BFS nature of the tree growth, D iterations of the outer loop of *Graph_Disperse_BFS* suffice. Hence, the running time is $O(D(\Delta^2 + \Delta(\Delta + D))$.

The following variables are used at each robot. (i) *nrobots* for the total number of robots at the root, v_{root}. Initialize as defined. (ii) *level* for the level of a robot/node in the BFS tree. Initialize to 0. (iii) i for the current maximum level of settled robots. Initialize to 0. (iv) *demand*$[1 \dots \Delta]$, where *demand*$[j]$ for a non-leaf node is the demand for robots to populate level $i + 1$ for sub-tree reachable via port j. Initialize to $\overline{0}$. *demand*$[j]$ for a leaf node at level i has the following semantics. *demand*$[j] = 0/1/2/3$ means the node reachable via port j does not go to level $i + 1$ node/goes to a level $i + 1$ node via an unfinalized edge/goes to a validated level $i + 1$ node/goes to an invalidated level $i + 1$ node. (v) *child_id*$[1 \dots \Delta]$, where *child_id*$[j]$ is the ID of the child node (if any,) reachable via port j. Initialize to $\overline{\perp}$. (vi) *parent_id* for the ID of the parent robot in the BFS tree. Initialize to $\perp$. (vii) *parent* to identify the port through which the parent node in the BFS tree is reached. Initialize to $\perp$. (viii) *winner* to uniquely select a robot among those that arrive at a level $i + 1$ node. Initialize to 0.

The variables *child_id*$[1 \dots \Delta]$ and *parent_id*, and the unique robot identifiers assumption, are strictly not necessary for the single-rooted case. Without *child_id*$[1 \dots \Delta]$ and *parent_id*, the broadcast function can be simulated by each settled robot moving up to its parent and back, to communicate the demand of its subtree. The unique robot identifiers assumption and *winner* help in determining which robot should settle at the root, for assigning robots as per the demands, and for selecting *winner*. Without these, a simple randomized scheme can be used for the above determinations.

Lemma 5.1. *The* **while** *loop of* `PopulateNextLevel(i)` *(line (11) in Algorithm 2) terminates within* Δ *iterations.*

Lemma 5.2. *A BFS tree in induced in the underlying graph by Algorithm* Graph_Disperse_BFS *given in Algorithm 2.*

Theorem 5.3. *Algorithm 2 (*Graph_Disperse_BFS*) solves* Dispersion *on single-rooted graphs in $O(D\Delta(\Delta + D))$ rounds and requires $O(\max(D, \Delta \log k))$ memory in the local communication model.*

Proof. There is one robot settled at each node of the BFS tree induced (Lemma 5.2); hence dispersion is achieved.

In one iteration of the main **while** loop:

1. `DetermineLeafDemand(i)` does 2-neighborhood traversals in parallel, and hence takes $O(\Delta^2)$ rounds.
2. In each of the Δ iterations of the **while** loop of `PopulateNextLevel(i)`, the upward movement and the downward movement of the robots in lines (12) and (17-18), respectively, takes i rounds; and the code block (20-25) takes 2Δ rounds.

So the time complexity is $\Delta^2 + \Delta(2\Delta + 2i)$. By Lemma 5.2, a BFS tree is induced and hence the maximum number of levels is D, which is the number of iterations of the **while** loop of *Graph_Disperse_BFS*. Thus the overall time time complexity is $\sum_{i=1}^{D} 1(\Delta^2 + \Delta(2\Delta + 2i)) = O(D\Delta(\Delta + D))$.

The variable *nrobots* takes $\log k$ bits, *level* and i take $\log D$ bits each, *demand*$[1 \dots \Delta]$ takes $\Delta \log k$ bits, *child_id*$[1 \dots \Delta]$ takes $\Delta \log k$ bits, *parent_id* takes $\log k$ bits, *parent* takes $\log \Delta$ bits, and *winner* takes 1 bit.

Note that the local communication model suffices because the broadcast function can be simulated by each settled robot moving up to its parent and back, to communicate the demand of its subtree. The theorem follows. □

5.2 General Case

We adapt the single-rooted algorithm to the multi-rooted case. From each root, a BFS tree is initiated in parallel, and is identified by the robot ID settled at the root. When two (or more) BFS trees meet at a node, a collision is detected; the tree with the higher depth (if unequal depths) subsumes the other tree(s) and collects the robots of the other tree(s) at its root. It then continues the BFS algorithm at the same depth in case $nrobots = 0$, i.e., level $i + 1$ may not be fully populated yet. A collision of two trees T_x and T_y is identified by a 4-tuple for each tree: $\langle root, depth, bordernode, borderport \rangle$. The changes to Algorithm 2 are given next, and the module for Collision processing is given in the full version attached in Appendix.

1. After Line 5, insert a line: Invoke a call to Collision processing.
2. Line 6: Conditionally increment i in line 6, as described in Step 3 of Collision processing.
3. Line 8: The case (iii) becomes case (iv) and the new case (iii) is: if v has a settled robot r' of another tree with root $root'$ and level lvl', r broadcasts Collide($\langle root, i + 1, u, out_u \rangle, \langle root', lvl', v, in_v \rangle$) – then discount v and backtrack.
4. Lines 19-24: are to be executed only with reference to robots belonging to my own tree (having same root).
5. Line 34: execute if no other robot from any tree arrives at the node v. Otherwise execute line 35.
6. Add new Line 35 in Algorithm 2: For each other robot r' of tree with root $root'$ and level $i+1$, broadcast Collide($\langle root, i+1, u, out_u \rangle, \langle root', i + 1, r', v_{in} \rangle$). Wait for SUBSUME messages broadcast in Collision processing. If x's tree subsumes other trees, x sets $level \leftarrow i + 1$, $parent\_id \leftarrow u$, $parent \leftarrow in$, x settles at v. (If x's tree is subsumed, x retraces its step back to u, then moves on to the root of the tree that subsumes its tree as described in Collision processing.)

Algorithm 2: Algorithm *Graph_Disperse_BFS* to solve Dispersion in global model. r denotes a settled robot, if any, at that node.

```
1  Initialize: $nrobots \leftarrow$ number of robots; $level$, $i$, $demand[1 \ldots \Delta]$, $child\_id[1 \ldots \Delta]$; $parent\_id$, $parent$, $winner$
2  robot with lowest ID settles at root, $level \leftarrow 0$
3  while $nrobots > 0$ do
4      DetermineLeafDemand($i$)
5      PopulateNextlevel($i$)
6      $i \leftarrow i + 1$
7  DetermineLeafDemand($i$)
8  Each settled robot $r$ at a leaf node $u$ at level $i$ does a 2-bounded DFS to count number of neighbors $v$ at level $i+1$. If on exploring $(u, v)$ via $out_u$, (i) $v$ is level $i-1$, then
     backtrack, (ii) else if $v$ has a level $i-1$ neighbor, then $v$ is level $i$ node - discount and backtrack, (iii) else $v$ is a level $i+1$ node, hence robot $r$ sets $demand_u[out_u] \leftarrow 1$.
9  Wait until $\Delta^2$ rounds are elapsed.
10 PopulateNextLevel($i$)
11 while $[\sum_{leaf\ u} \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = 1)] > 0 \bigwedge [\sum_{leaf\ u} \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = 2)] < nrobots$ do
12     All unassigned robots at level $i$ move upwards to root using $parent$ pointers in $i$ rounds
13     Leaf node $u$ broadcasts $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = 1))$
14     On receiving $B1(x, my\_id, y)$, if $child\_id^{-1}[x] = \theta$ then $demand[\theta] \leftarrow y$
15     On receiving $B1$ from all children ($\forall x \mid child\_id[x] \neq 0$), broadcast $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} demand[j])$
16     Wait until $i$ rounds are elapsed; synchronize()
17     When root receives $B1$ from all children, distribute $\min(nrobots, \sum_{j=1}^{\Delta} demand[j])$ robots among children after resetting $winner \leftarrow 0$ for each robot
18     Robots move down the tree to the leaf nodes at level $i+1$: On receiving $x$ robots, a node at level $< i$ ($= i$) distributes among children reachable via ports $p$ such that
         $demand[p] > 0$ ($demand[p] = 1$).
19     On arrival at level $i+1$ node $v$ from level $i$ node $u$ via $(out_u, in_v)$, robot with lowest ID sets $winner \leftarrow 1$
20     if $winner = 0$ then
21         retrace back via $in_v$ to $u$ and wait
22     else if $winner = 1$ then
23         visit each neighbor $w$ via $(out_v, in_w)$. If $w (\neq u)$ is at level $i$, $demand_w[in_w] \leftarrow 3$
24         retrace back from $v$ using $in_v$ to $u$; $demand_u[out_u] \leftarrow 2$
25     Wait until $2\Delta - 1$ rounds are elapsed since arriving at level $i+1$ (so all robots at level $i+1$ are back at level $i$); synchronize()
26 All unassigned robots at level $i$ move upwards to root using $parent$ pointers in $i$ rounds
27 Leaf node $u$ broadcasts $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = 2))$
28 On receiving $B1(x, my\_id, y)$, if $child\_id^{-1}[x] = \theta$ then $demand[\theta] \leftarrow y$
29 On receiving $B1$ from all children ($\forall x \mid child\_id[x] \neq 0$), broadcast $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} demand[j])$
30 Wait until $i$ rounds are elapsed.
31 When root receives $B1$ from all children, distribute $\min(nrobots, \sum_{j=1}^{\Delta} demand[j])$ robots among children; $nrobots \leftarrow nrobots - \sum_{j=1}^{\Delta} demand[j]$
32 Robots move down the tree to the leaf nodes at level $i$: On receiving $x$ robots, a node at level $< i$ distributes among children reachable via ports $p$ such that $demand[p] > 0$.
33 At a level $i$ node $u$, on receiving $\leq \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = 2)$ robots, send one robot $x$ on each port $out \mid demand_u[out] = 2$; $child\_id_u[out] \leftarrow x$
34 The robot $x$ reaches node $v$ at level $i+1$ via incoming port $in$, $level \leftarrow i + 1$, $parent\_id \leftarrow u$, $parent \leftarrow in$, $x$ settles at the node
```

Theorem 5.4. *Algorithm 2 (*Graph_Disperse_BFS*) along with the modifications given in this section solves* Dispersion *in multi-rooted graphs in $O(\max(D, k)\Delta(\Delta + D))$ rounds and requires $O(\max(D, \Delta \log k))$ memory in the global communication model.*

Proof of Theorem 1.2: Follows from Theorems 5.3 and 5.4.

6 ALGORITHM FOR ARBITRARY TREES (THEOREM 1.3)

We present and analyze a BFS-based algorithm that solves Dispersion of $k \leq n$ robots on an arbitrary n-node tree in $O(D \max(D, k))$ time with $O(\max(D, \Delta \log k))$ bits of memory at each robot in the global communication model. This algorithm improves the $O(n)$ time of the best previously known algorithm [1] for arbitrary trees (Table 2) in the local communication model.

In the rooted graph case, we first show that dispersion takes $O(D^2)$ time with $O(\max(D, \Delta \log k))$ bits of memory at each robot in the local communication model. We then extend the result to the general case of robots on multiple nodes.

6.1 Rooted Case

The rooted tree case, where all robots are initially colocated at one node, is a special case of Algorithm 2 adapted to the tree topology. `DetermineLeafDemand(i)` behaves as follows: instead of lines 8-9, $demand_u[out_u] \leftarrow 1$ for all out_u other than $parent$. In `PopulateNextLevel(i)`, the **while** loop (lines 11-25) is removed, and robots do not move upwards to root (line 26). A single iteration, as per the following modification of lines (27-34), and excluding line (30), suffices.

27: Leaf node u broadcasts $B1(my\_id, parent\_id, \delta - 1)$.
28: On receiving $B1(x, my\_id, y)$, if $child\_id^{-1}[x] = \theta$ then $demand[\theta] \leftarrow y$.
29: On receiving $B1$ from all children ($\forall x \mid child\_id[x] \neq 0$), broadcast $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} demand[j])$.
31: When root receives $B1$ from all children, distribute $\min(nrobots, \sum_{j=1}^{\Delta} demand[j])$ robots among children; $nrobots \leftarrow nrobots - \sum_{j=1}^{\Delta} demand[j]$.
32: Robots move down the tree to the leaf nodes at level i: On receiving x robots, a node at level $< i$ distributes among children reachable via ports p such that $demand[p] > 0$.
33: At a level i node u, on receiving $\leq \delta - 1$ robots, send one robot x on each port $out \mid demand_u[out] = 1$; $child\_id_u[out] \leftarrow x$.

34: The robot x reaches node v at level $i+1$ via incoming port in, $level \leftarrow i+1$, $parent\_id \leftarrow u$, $parent \leftarrow in$, x settles at the node.

Theorem 6.1. *Algorithm 2 (*Graph_Disperse_BFS*) with the changes described in this section solves* Dispersion *on a single-rooted tree in $O(D^2)$ rounds and requires $O(\max(D, \Delta \log\, k))$ memory in the local communication model.*

6.2 General Case

Each of the multiple roots on the tree topology initiate in parallel the execution of the rooted tree case, as described in Section 6.1, with the following additional changes.

(1) After Line 5 of Algorithm 2, insert a line: Invoke a call to Collision processing.
(2) Line 6 of Algorithm 2: Conditionally increment i in line 6, as described in Step 3 of Collision processing.
(3) Execute line 34 of Algorithm 2 only if no other robot from any other tree arrives at the node v nor is there a robot from another tree settled at v: "The robot x reaches node v at level $i+1$ via incoming port in, $level \leftarrow i+1$, $parent\_id \leftarrow u$, $parent \leftarrow in$, x settles at the node." Otherwise execute the following line 35.
(4) Add new line 35 in Algorithm 2: For each other robot r' of tree with root $root'$ and level $i+1$ that arrives at v, broadcast Collide($\langle root, i+1, u, out_u\rangle, \langle root', i+1, r', v_{in}\rangle$). If there is a robot r'' of tree with root $root''$ and level $i'' \leq i$ settled at v, broadcast Collide($\langle root, i+1, u, out_u\rangle, \langle root'', i'', r'', v_{in}\rangle$). Wait for SUBSUME messages broadcast in Collision processing. If x's tree subsumes other trees and there is no robot r'' settled at v, x sets $level \leftarrow i+1$, $parent\_id \leftarrow u$, $parent \leftarrow in$, x settles at v. Else if x's tree subsumes other trees and there is a robot r'' settled at v, x retraces back to u and up to its root, along with robots of other subsumed trees that relocate to $root$ (the root of the tree associated with x), and also resets $child\_id_u[out]$. (Else if x's tree is subsumed, x retraces its step back to u, then moves on to the root of the tree that subsumes its tree as described in Collision processing.)

Theorem 6.2. *Algorithm 2 (*Graph_Disperse_BFS*) along with changes described above and in Section 6.1 solves* Dispersion *on a multi-rooted tree in $O(D \max(D, k))$ rounds and requires $O(\max(D, \Delta \log\, k))$ memory in the global communication model.*

Proof of Theorem 1.3: Follows from Theorems 6.1 and 6.2.

7 CONCLUDING REMARKS

We have presented three results for solving Dispersion of $k \leq n$ robots on n-node graphs. The first two results are for arbitrary graphs and the third result is for arbitrary trees. The first result for arbitrary graphs is based on a DFS traversal and improves by $O(\log k)$ factor the best previously known algorithm in the local communication model. The second algorithm for arbitrary graphs is based on a BFS traversal and improves significantly on the $O(\Delta^D)$ time of the best previously known algorithm in the local communication model. The algorithm for arbitrary trees is also based on a BFS traversal and improves on the $O(n)$ time of best previously known algorithm in the local communication model.

For future work, it will be interesting to solve Dispersion on arbitrary graphs using a DFS-based algorithm with time $O(k)$ or improve the existing time lower bound of $\Omega(k)$ to $\Omega(\min(m, k\Delta))$. For BFS-based algorithms, it will be interesting to improve $\max(D, k)$ factor to $O(D)$ for both arbitrary graphs and trees. The third interesting direction will be to consider faulty robots. The fourth interesting direction will be to solve Dispersion in dynamic graphs. The fifth interesting direction will be to extend our algorithms to semi-synchronous and asynchronous settings.

REFERENCES

[1] John Augustine and William K. Moses Jr. Dispersion of mobile robots: A study of memory-time trade-offs. In *ICDCN*, pages 1:1–1:10, 2018.
[2] Evangelos Bampas, Leszek Gasieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, and Adrian Kosowski. Euler tour lock-in problem in the rotor-router model: I choose pointers and you choose port numbers. In *DISC*, pages 423–435, 2009.
[3] L. Barriere, P. Flocchini, E. Mesa-Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. In *IPDPS*, pages 1–8, 2009.
[4] Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. Algorithms*, 4(4):42:1–42:18, August 2008.
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
[6] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279–301, October 1989.
[7] Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration of trees by energy-constrained mobile robots. *Theory Comput. Syst.*, 62(5):1223–1240, 2018.
[8] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609:171–184, 2016.
[9] Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pajak, and Przemyslaw Uznański. Fast collaborative graph exploration. *Inf. Comput.*, 243(C):37–49, August 2015.
[10] Yotam Elor and Alfred M. Bruckstein. Uniform multi-agent deployment on a ring. *Theor. Comput. Sci.*, 412(8-10):783–795, 2011.
[11] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.
[12] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Mobile Entities*, volume 1 of *Theoretical Computer Science and General Issues*. Springer International Publishing, 2019.
[13] Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.
[14] Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, November 2005.
[15] Tien-Ruey Hsiang, Esther M. Arkin, Michael A. Bender, Sandor Fekete, and Joseph S. B. Mitchell. Online dispersion algorithms for swarms of robots. In *SoCG*, pages 382–383, 2003.
[16] Tien-Ruey Hsiang, Esther M. Arkin, Michael A. Bender, Sándor P. Fekete, and Joseph S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *WAFR*, pages 77–94, 2002.
[17] Ajay D. Kshemkalyani and Faizan Ali. Efficient dispersion of mobile robots on graphs. In *ICDCN*, pages 218–227, 2019.
[18] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Improved dispersion of mobile robots on arbitrary graphs. *CoRR*, abs/1812.05352, 2018.
[19] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots in the global communication model. *CoRR*, abs/1909.01957, 2019.
[20] Artur Menc, Dominik Pajak, and Przemyslaw Uznanski. Time and space optimality of rotor-router graph exploration. *Inf. Process. Lett.*, 127:17–20, 2017.
[21] Anisur Rahaman Molla and William K. Moses Jr. Dispersion of mobile robots: The power of randomness. In *TAMC*, pages 481–500, 2019.
[22] Christian Ortolf and Christian Schindelhauer. Online multi-robot exploration of grid graphs with rectangular obstacles. In *SPAA*, pages 27–36, 2012.
[23] Pavan Poudel and Gokarna Sharma. Time-optimal uniform scattering in a grid. In *ICDCN*, pages 228–237, 2019.
[24] Masahiro Shibata, Toshiya Mega, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Uniform deployment of mobile agents in asynchronous rings. In *PODC*, pages 415–424, 2016.