Journal of Parallel and Distributed Computing 161 (2022) 100–117

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Dispersion of mobile robots using global communication ☆

Ajay D. Kshemkalyani [a], Anisur Rahaman Molla [b], Gokarna Sharma [c,*]

[a] University of Illinois at Chicago, Chicago, USA
[b] Indian Statistical Institute, Kolkata, India
[c] Kent State University, Kent, USA

ARTICLE INFO

Article history:
Received 21 August 2020
Received in revised form 23 August 2021
Accepted 30 November 2021
Available online 7 December 2021

Keywords:
Multi-agent systems
Autonomous mobile robots
Dispersion
Distributed algorithms
Time and memory complexity

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 using 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 two novel deterministic algorithms for arbitrary graphs in a synchronous setting where all robots perform their actions in every time step. Our first algorithm is based on a DFS traversal and guarantees (i) $O(k\Delta)$ steps runtime using $O(\log(k+\Delta))$ bits at each robot and (ii) $O(\min(m, k\Delta))$ steps runtime using $O(\Delta + \log k)$ bits at each robot, where m is the number of edges and Δ is the maximum degree of the graph. The second algorithm is based on a BFS traversal and guarantees $O((D+k)\Delta(D+\Delta))$ steps runtime using $O(\log D + \Delta \log k))$ bits at each robot, where D is the diameter of the graph. Our results complement the existing results established using the *local communication* model where a robot can communication only with other robots present at the same node.

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 [16,17]. 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 [1,19]. 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 [1,19]. 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 [12,13].

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 using 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 [1,19,20] (details in Table 1 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 using the local model also arise using 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 using

☆ A preliminary version of this article appears in the Proceedings of ICDCN'20 [21].

* Corresponding author.
E-mail address: sharma@cs.kent.edu (G. Sharma).
URL: https://www.isical.ac.in/~molla (A.R. Molla).

https://doi.org/10.1016/j.jpdc.2021.11.007
0743-7315/

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.

Algorithm	Memory/robot (in bits)	Time (in rounds)	Comm. Model/ Initial Conf.
Lower bound	$\Omega(\log k)$	$\Omega(k)$	local
[1][1]	$O(\log n)$	$O(mn)$	local/general
[19]	$O(k \log \Delta)$	$O(m)$	local/general
[19]	$O(D \log \Delta)$	$O(\Delta^D)$	local/general
[19]	$O(\log(k+\Delta))$	$O(mk)$	local/general
[20]	$O(\log n)$	$O(\min(m, k\Delta) \cdot \log k)$	local/general
Lower bound 1	$\Omega(\log k)$	$\Omega(k)$	global
Lower bound 2 (for trees)	$\Omega(\log k)$	$\Omega(D^2)$	global
Thm. 1.1(a)	$O(\log(k+\Delta))$	$O(k\Delta)$	global/general
Thm. 1.1(b)	$O(\Delta + \log k)$	$O(\min(m, k\Delta))$	global/general
Thm. 1.2(a)	$O(\log D + \Delta \log k)$	$O(D\Delta(D+\Delta))$	local/rooted
Thm. 1.2(b)	$O(\log D + \Delta \log k)$	$O((D+k)\Delta(D+\Delta))$	global/general

[1] The results in [1] are only for $k = n$.

the local model. The global communication model has been considered heavily in the past in distributed robotics, e.g., see [8,15,25], 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 two new deterministic algorithms for Dispersion using the global communication model for arbitrary graphs. Our first algorithm using a *depth first search* (DFS) traversal performs better than the state-of-the-art using the local communication model by a $O(\log k)$ factor; see Table 1. The second algorithm is the first algorithm designed for Dispersion using a *breadth first search* (BFS) traversal and provides different time-memory trade-offs. We also complement our algorithms by some lower bounds on time and memory requirement using the global model.

Overview of the Model and Results. We consider the same model (with the only difference as described in the next paragraph) as in Augustine and Moses Jr. [1], Kshemkalyani and Ali [19], and Kshemkalyani et al. [20] where a system of $k \leq n$ robots is 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 $[0, \delta - 1]$, 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 [1,19,20] 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 with, except of those they are colocated with.

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:*

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

Theorem 1.1 performs better than the $O(\min(m, k\Delta) \cdot \log k)$ time best previously known algorithm [20] using the local communication model by a factor of $O(\log k)$ with an additional $O(\Delta)$ bits (see Table 1). We also prove a time lower bound of $\Omega(k)$ and memory lower bound of $\Omega(\log k)$ bits at each robot for Dispersion on graphs using the global communication model. The implication is that, for constant-degree arbitrary graphs (i.e., when $\Delta = O(1)$), Theorem 1.1 is asymptotically optimal with respect to both memory and time, the first such result for arbitrary graphs. Theorem 1.2 gives significantly better run-time than the $O(\Delta^D)$ algorithm [19] using the local communication model. We also prove a time lower bound of $\Omega(D^2)$ for Dispersion on trees using the global communication model. This implies that the time in Theorem 1.2(a) is asymptotically optimal for constant-degree arbitrary graphs.

Although Theorem 1.2(b) has higher time and space complexity than Theorem 1.1, the algorithm for Theorem 1.2(b) is significant because: (i) it is the first algorithm that is based on a BFS approach (rather than DFS) to solve Dispersion and illustrates a new technique, (ii) it performs better than the $O(\Delta^D)$ algorithm [19] using the local communication model, and (iii) it directly contributes to solving the

problem of merging concurrently initiated BFS tree constructions and traversals in a distributed setting, which is a very broad problem with many applications.

Challenges and Techniques. The well-known DFS traversal approach [5] was used in the previous results on Dispersion [1,19,20]. If all k robots are positioned initially on a single node of G, then the DFS traversal finishes in $\min(4m-2n+2, 4k\Delta)$ 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. [20] designed a smarter way to synchronize the parallel DFS traversals so that the total time increases only by a factor of $\log k$ to $\min(4m-2n+2, 4k\Delta)\cdot\log k$ rounds, in the worst-case, using 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 (Algorithm 2) that allows to synchronize DFS traversals without re-traversing the already traversed part of the graph giving us $\min(4m-2n+2, 4k\Delta)$ rounds, as if running DFS starting from all robots in the same node, using the global communication model. This gives an additive, not multiplicative, cost in synchronizing different DFS traversals. This is possible due to the information that can be passed to the robots to take their next actions, even if they do not know their positions on G. This passing of information among remotely located robots has not been possible using the local communication model in the literature, and does not seem to be possible because of the very nature of the local communication model. The best mechanism for synchronization in the local communication model incurred an $O(\log k)$ factor to synchronize $O(k)$ trees that might be formed in the dispersion process. Our time bound for the global communication model becomes $O(k)$ for constant-degree arbitrary graphs, which is asymptotically time-optimal. Our proposed technique in Algorithm 2 can be generalized and applied to any problem which requires merging concurrently-initiated and concurrently-growing DFS tree components into a single DFS component at an additive cost.

Despite efficiency in merging the DFS traversal trees due to global communication, the time bound of $\min(4m-2n+2, 4k\Delta)$ seems to be inherent in algorithms based on a DFS traversal [5,19], and even if using the global model. A 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 (Algorithms 3 and 4) of performing BFS so that we can achieve dispersion in arbitrary graphs in $O(D\Delta(D+\Delta))$ time in the rooted initial configuration. The general initial configuration introduces a $(D+k)$ factor instead of the $O(D)$ factor in the time bound. Our proposed technique in Algorithms 3 and 4 can be generalized and applied to any problem which requires merging concurrently initiated and concurrently growing BFS tree components into a single BFS component.

Related Work. There are three previous studies focusing on Dispersion using 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 [19] provided an $\Omega(k)$ time lower bound for arbitrary graphs for $k\leq n$. 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 $O(\log(k+\Delta))$ bits at each robot with $O(mk)$ time. Recently, Kshemkalyani et al. [20] provided an algorithm for arbitrary graph that runs in $O(\min(m, k\Delta)\cdot\log k)$ time using $O(\log n)$ bits memory at each robot. For grid graphs, Kshemkalyani et al. [22] provided an algorithm that runs in $O(\min(k, \sqrt{n}))$ time using $O(\log k)$ bits memory at each robot using the local model and an algorithm that runs in $O(\sqrt{k})$ time using $O(\log k)$ bits memory at each robot using the global model. Randomized algorithms are presented in [24] to solve Dispersion from rooted initial configurations where the random bits are mainly used to reduce the memory requirement at each robot. In this paper, we present results using the global communication model. The previous results on arbitrary graphs 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., [2,4,9,14,18,23]. 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 [23]. The collective exploration by a team of robots is studied in [15] 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 [11,27] and grids [3]. Recently, Poudel and Sharma [26] provided a $\Theta(\sqrt{n})$-time algorithm for uniform scattering in a grid [7]. 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 [12,13] 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 rooted arbitrary graphs in Section 5, proving Theorem 1.2(a). We then present a BFS-based algorithm for arbitrary graphs in Section 6, proving Theorem 1.2(b). Finally, we conclude in Section 7 with a short discussion.

2. Model details and preliminaries

Graph. We consider the same graph model as in [1,19]. 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* (a.k.a. port number) in the range $[0,\delta-1]$, 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 [1,19,20]. Let $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$ be a set of $k \leq n$ robots residing on the nodes of G. For simplicity, we sometimes 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, except for co-located robots, 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.

The global communication abstraction can be implemented by message-passing or through memory. In some of our algorithms, we also use message-passing for non-co-located robots, wherein a directed message to a destination robot can be sent or a broadcast can be performed. This can be simulated in the memory model as follows. If robot i wants to send to j, it writes "broadcast/j:⟨message content⟩" in its own memory and the receiver(s) j read this content from i's memory.

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 some node v_j, the robot r_i at node v_i can observe the memory of r_j. In some of our algorithms, this communication is done by message-passing if $v_i \neq v_j$ (r_i does a send or broadcast and r_j does a receive) which can be simulated in the memory model. 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_j.
- *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.

A cycle may have $O(1)$ CCM sub-cycles.

Time and Memory Complexity. We consider the synchronous setting where every robot is active in every CCM cycle and sub-cycle and they perform the cycle and sub-cycles in 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. The memory we count is persistent memory, which is the memory that needs to survive across rounds. This includes the read-only memory for the robot identifier.

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

Definition 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 using the global communication model, which show the difficulty in obtaining fast runtime and low 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 2.1. *Any deterministic algorithm for* Dispersion *on graphs requires $\Omega(k)$ steps using the global communication model.*

Proof. Consider a line graph G and a rooted initial configuration of $k \leq n$ robots on a single node v_{root} of G. In order for the robots to solve Dispersion, they need to settle at k distinct nodes of G, exactly one on each node. To reach a node to settle, some robot must travel $k - 1$ edges of G, taking $k - 1$ time steps. □

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

Theorem 2.2. *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 using the global communication model.*

Proof. We use the lower bound proof for exploration due to Disser et al. [10] to prove this lower bound. It has been argued in [1] that a lower bound for exploration applies to Dispersion. We argue here that the lower bound of [10] applies for Dispersion using the global communication model. Disser et al. [10] proved a lower bound for exploration assuming a rooted initial configuration in which $k = n$ robots are on a single node v_{root} of tree T. Moreover, they assumed that the nodes of tree T have unique identifiers and the robots have global communication. Specifically, they showed that: Using $k = n$ robots, there exists a tree T on n vertices and with diameter (height) $D = \omega(1)$ such that any deterministic exploration strategy requires at least $D^2/3 = \Omega(D^2)$ steps to explore T. As our model is weaker

because the nodes are indistinguishable, the $\Omega(D^2)$ steps lower bound applies to DISPERSION in trees using the global communication model. □

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

Theorem 2.3. *Any deterministic algorithm for* DISPERSION *on n-node anonymous graphs requires* $\Omega(\log k)$ *bits at each robot using the global communication model, where* $k \leq n$ *is the number of robots.*

Proof. From the system model, each robot must have its identifier in the range $[1, k]$ and we count this memory space for the identifier which must survive across rounds as the space complexity. □

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 $R(v) = \{r_1, \ldots, r_k\}$, where r_i is the robot with ID i. We describe here a DFS traversal algorithm, $DFS(k)$, that disperses all the robots in the set $R(v)$ to 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.

1. *parent* (initially assigned $\perp$), for a settled robot denotes the port through which it first entered the node it is settled at.
2. *child* (initially assigned -1), stores the port of the node where r_i is currently located, that it has last taken (while entering/exiting the node).
3. *treelabel* (initially assigned $\min(R(v))$) stores the ID of the smallest ID robot the tree is associated with.
4. $r_i.state \in \{forward, backward, settled\}$ (initially assigned $forward$). $DFS(k)$ executes in two phases, $forward$ and *backtrack* [5].

The algorithm pseudo-code is shown in Algorithm 1. The robots in $R(v)$ move together in a DFS, leaving behind the highest ID robot at each newly discovered node. They all adopt the ID of the lowest ID robot in $R(v)$ which is the last to settle, as their *treelabel*. Let the node visited have degree δ. When robots enter a node through port *child* either for the first time in forward mode or at any time in backtrack mode, the unsettled robots move out using port $(child + 1) \mod \delta$. However, when robots enter a node through port *child* in forward mode for the second or subsequent time, they change phase from forward to backtrack and move out through port *child*. (It is straight-forward to modify Algorithm 1 so that the settled robot r also tracks in $r.child$ the port through which the other robots last left the node except when they entered the node in forward mode for the second or subsequent time and hence backtracked through the port through which they entered that time.)

Algorithm 1: Algorithm *DFS(k)* for DFS traversal of a graph by k robots from a rooted initial configuration. Code for robot i.

```
1  Initialize: child ← −1, parent ←⊥, state ← forward, treelabel ← min(R(v))
2  for round = 1 to min(4m − 2n + 2, 4kΔ) do
3      child ← port through which node is entered
4      if state = forward then
5          if node is free then
6              if i is the highest ID robot on the node then
7                  state ← settled, i settles at the node (does not move henceforth), parent ← child, treelabel ← lowest ID robot at the node
8              else
9                  child ← (child + 1) mod δ
10                 if child = parent of robot settled at node then
11                     state ← backtrack
12         else
13             state ← backtrack
14     else if state=backtrack then
15         child ← (child + 1) mod δ
16         if child ≠ parent of robot settled at node then
17             state ← forward
18     move out through child
```

Theorem 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* $\min(4m - 2n + 2, 4k\Delta)$ *rounds using* $O(\log(k + \Delta))$ *bits at each robot.*

Proof. We first show that DISPERSION is achieved by $DFS(k)$. Because every robot starts at the same node and follows the same path as other not-yet-settled robots until it is assigned to a node, $DFS(k)$ resembles the DFS traversal of an anonymous port-numbered graph [1] with all robots starting from the same node. Therefore, $DFS(k)$ visits k different nodes, where each robot is settled.

We now prove time and memory bounds. The DFS traversal may take up to $4m - 2n + 2$ rounds, with each forward edge being traversed twice and each backward edge being traversed 4 times (once in either direction in the forward phase and once in either direction in the backward phase) [19]. However, in $4k\Delta$ rounds, $DFS(k)$ is guaranteed to visit at least k different nodes of G because in the DFS, each edge

can be traversed at most 4 times and hence at most 4Δ traversals can visit a particular node [19]. If $4m - 2n + 2 < 4k\Delta$, $DFS(k)$ visits all n nodes of G. Therefore, it is clear that the runtime of $DFS(k)$ is $\min(4m - 2n + 2, 4k\Delta)$ rounds. Regarding memory, variable *treelabel* takes $O(\log k)$ bits, *state* takes $O(1)$ bits, and *parent* and *child* take $O(\log \Delta)$ bits. The k robots can be distinguished through $O(\log k)$ bits since their IDs are in the range $[1, k]$. Thus, each robot requires $O(\log(k + \Delta))$ bits. □

4. DFS-based 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 (i) $O(k\Delta)$ time with $O(\log(k + \Delta))$ bits of memory at each robot and (ii) $O(\min(m, k\Delta))$ time with $O(\Delta + \log k)$ bits of memory at each robot using the global communication model. This algorithm has better run-time than the $O(\min(m, k\Delta) \cdot \log k)$ time of the best previously known algorithm [19] for arbitrary graphs (Table 1) using the local communication model. We mainly discuss result (i) in Sections 4.1– 4.3. The last subsection, Section 4.4, discusses how to modify the approach of result (i), to obtain result (ii). Specifically, result (ii) achieves improved time of $O(\min(m, k\Delta))$ in the expense of additional $O(\Delta)$ bits per robot.

4.1. Basic idea

When k robots are located at more than one node in the initial configuration, multiple concurrent DFS traversals described in Section 3 are initiated. Two or more such DFSs collide when one DFS visits a node where another DFS has settled a robot or another DFS is also visiting in this round. The challenge is to combine these DFSs efficiently. Our algorithm operates in two phases: DFS *Grow*, and DFS *Collect*. The $DFS(k)$ described in Algorithm 1 is the DFS *Grow* and is associated with a component ID, CID, which is the *treelabel*. When two or more DFSs collide and form a connected component of DFSs, using global communication, a *Subsume* graph is constructed to represent which DFS has collided into which other one. One of the DFSs called the *winner* DFS is chosen to subsume the others in the connected component of CIDs and collapses the others. An unsettled robot from the *winner* DFS is chosen as leader and it traverses all the other connected DFS components of subsumed DFSs in G by entering the DFS *Collect* phase to collect all the settled and unsettled robots from those DFS components using a DFS traversal. Then they all switch to the DFS *Grow* phase of the *winner* and continue its DFS *Grow* from the home node where it was interrupted by the collision. This process can repeat up to $k - 1$ times.

The algorithm has two nice properties based on how the *winner* component is chosen after each collision.

1. All the *settled nodes* (defined as nodes with a settled robot) in all the subsumed components of DFSs (which are the nodes in the connected component of the *Subsume* graph) form a connected component in the network graph G.
2. Hence it is possible for the leader to successfully traverse these nodes, and only these nodes (i.e., not traverse the nodes in the *winner*'s component or any other nodes) in the DFS *Collect* phase, and collect their settled and unsettled robots to the home node from which the leader started the DFS *Collect* traversal.

A component CID may get subsumed multiple times before/while being collected because of possibly multiple collisions and multiple *winner*s over time. However, the collection time of a subsumed DFS component is linear in the number of settled nodes in it. To see this, consider a component CID that is always a *winner* across the lifetime of the execution.

1. From the above properties, all the robots in all the components subsumed by the *winner* over time get collected by the *winner*'s leaders without interruption of their DFS *Collect* phases. The total time cost of the DFS *Collect* phases of the *winner* component is linearly additive, each term being the time for a DFS traversal of only the settled nodes in the subsumed components each time the leader of that *winner* switches to DFS *Collect* phase. This is bounded by $O(\min(m, k\Delta))$.
2. In addition, the time cost of the DFS *Grow* phases of the *winner* is additive and hence the total cost is that for a DFS traversal of the maximum number of robots in that *winner*, which is bounded by $O(\min(m, k\Delta))$.

The overall time cost is additive and hence bounded by $O(\min(m, k\Delta))$.

4.2. The algorithm

The algorithm is based on DFS traversal. In general, a robot may operate in one of two interchangeable DFS phases: DFS *Grow* and DFS *Collect*. As these are independent, a separate set of DFS variables: *parent*, *child*, *state* is used for operating in the two phases. The following additional variables are used.

1. TID_Grow: Tree ID, of type robot identifier, is the ID, i.e., *treelabel*, of the DFS tree in the GROW phase with which the robot is associated. Initially, $TID\_Grow \leftarrow$ minimum ID among the colocated robots.
2. $TID\_Collect$: Tree ID, of type robot identifier, is the ID of the DFS tree in the DFS *Collect* phase with which the robot is associated. Initially, $TID\_Collect \leftarrow \perp$.
3. CID: for component ID, of type robot identifier, is used to denote the component associated with the DFS *Grow* phase of the DFS. Initially, $CID \leftarrow TID\_Grow$.
4. 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 DFS *Grow* phase of the DFS. Initially, $CID\_old \leftarrow TID\_Grow$.
5. *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, $winner \leftarrow \perp$.
6. *leader*: of type boolean. This is set to 1 if the robot is responsible for collecting the various robots distributed in the subsumed components. Initially, $leader \leftarrow 0$.

Algorithm 2: 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 ($prevcurr\_settled = 1$), if any, at that node.

```
1  Initialize: TID_Grow, TID_Collect, CID, CID_old, winner, leader, home, state, prevcurr_settled
2  if state = grow then
3      if node is free then
4          highest ID robot y from highest CID group having state = grow settles; y.state ← settled; y.prevcurr_settled ← 1
5      if CID ≠ r.CID then
6          lowest ID robot from each CID group broadcasts Subsume(CID, r.CID)
7      Subsume_Graph_Processing
8      if CID is a node in Subsume graph then
9          CID_old ← CID; CID ← winner in my connected component of Subsume graph
10         Let x ← min_j(j.TID_Grow = winner)
11         x.home ← r.ID; x.leader ← 1; x.state ← collect; x.TID_Collect ← x.TID_Grow
12         x begins DFS Collect(TID_Collect)
13         if i ≠ x ∧ state = grow on entering this round then
14             state ← subsumed; STOP moving until collected
15     else
16         continue DFS Grow(TID_Grow)
17 else if state = collect then
18     Subsume_Graph_Processing
19     if CID is a node in Subsume graph then
20         CID_old ← CID; CID ← winner in my connected component of Subsume graph
21         state ← subsumed; STOP moving until collected
22         if leader = 1 then
23             leader ← 0; home ←⊥
24     else
25         if node is free ∨CID = r.CID = r.CID_old ∨ CID ≠ r.CID then
26             backtrack, as part of DFS Collect(TID_Collect)
27         else if CID = r.CID ≠ r.CID_old then
28             if ∃ arrived robot x at my node | x.leader = 1 ∧ x.home = r.ID∧ all ports at r have been explored then
29                 if x = i then
30                     x.leader ← 0; x.home ←⊥
31                 state ← grow; TID_Grow ← x.TID_Grow
32                 i continues DFS Grow(TID_Grow)
33             else
34                 i continues DFS Collect(TID_Collect) of x | x.leader = 1, along with x if x is backtracking to its parent in DFS Collect(x.TID_Collect)
35 else if state = subsumed then
36     Subsume_Graph_Processing
37     if CID is a node in Subsume graph then
38         CID_old ← CID; CID ← winner in my connected component of Subsume graph
39     else
40         if ∃ arrived robot x at my node | x.state = collect ∧ x.leader = 1 ∧ x is backtracking to its parent in DFS Collect(TID_Collect) then
41             state ← collect; TID_Collect ← x.TID_Collect
42             if x.home = r.ID∧ all ports at r have been explored then
43                 state ← grow; TID_Grow ← x.TID_Grow; prevcurr_settled ← 0
44                 i continues DFS Grow(TID_Grow) along with x
45             else
46                 prevcurr_settled ← 0; i continues DFS Collect(TID_Collect) along with x
47 else if state = settled then
48     Subsume_Graph_Processing
49     if CID is a node in Subsume graph then
50         CID_old ← CID; CID ← winner in my connected component of Subsume graph
51         if CID_old ≠ CID then
52             state ← subsumed, do not move until collected
53 Subsume_Graph_Processing
54 receive Subsume messages; build Subsume graph S
55 if exists node with no incoming edge in my connected component of S then
56     winner ← min_CID(CIDs of nodes with no incoming edge in my component of S)
57 else if all nodes in my connected component of S are in a cycle then
58     winner ← min_CID(CIDs of nodes in cycle in my connected component)
```

7. *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 subsumed components back to this origin node. Initially, $home \leftarrow \perp$.
8. *state*: denotes the state of the robot and can be one of $\{grow, collect, subsumed, settled\}$. Initially, $state \leftarrow grow$.
9. *prevcurr_settled*: of type boolean. A robot sets this when entering $state = settled$ and resets it when leaving $state = subsumed$. A robot is considered settled if $prevcurr\_settled = 1$. Initially, $prevcurr\_settled \leftarrow 0$.

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. A free node is one where there is no robot having $prevcurr\_settled = 1$. As the robots 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 different CIDs 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 or settles there in this round, defined as having $prevcurr\_settled = 1$. 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 $r.CID$. Multiple such *Subsume* messages may get broadcast from different robots in different parts of the graph in any particular round.

All the robots listen to all such broadcasts in each round, and build a directed graph, *Subsume*, $S = (C, L)$, where C is the set of component IDs, and edge $(CID_j, CID_k) \in L$ 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, all nodes (in my connected component) of the *Subsume* graph must exist in one cycle, 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 and get collected by a leader robot from the *winner* component.

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 (settled and unsettled) robots which have newly changed their CID to that of the *winner* leader (hence $CID \neq CID\_old$, where CID_old is the value of CID before the latest overwrite by *winner*), also change their *state*, whether *grow*, *collect*, *settled*, or *subsumed* to *subsumed* and stop movement until they are collected. Also, all unsettled robots with $state = grow$ and $CID = winner = CID\_old$ but are not the leader change their *state* to *subsumed* and stop movement until they are collected. In this DFS *Collect* traversal, the leader node collects all settled and 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 being traversed gets subsumed by some other component, the robots in the component being traversed reset their *state* to *subsumed*. If the DFS *Collect* 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. If the DFS *Collect* does not complete, it is because the component got subsumed by another *winner* component; the robots in the interrupted DFS *Collect* change *state* to *subsumed* and stop until they are themselves collected by the leader of the new *winner*.

Note that the DFS $Collect(TID\_Collect)$ is independent of the DFS $Grow(TID\_Grow)$, and thus an independent set of variables *parent*, *child*, *state* 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 by a robot, the robot switches to the new instance and resets the old values of *parent*, *child*, *state* 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 $CID = r.CID \neq r.CID\_old$.) 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 in which it was to backtrack (because it gets subsumed), before it can backtrack, the leader (and any accompanying robots having $state = collect$) simply changes *state* to *subsumed* 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.

4.3. 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 Fig. 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. From Theorem 3.1, the DFS $Grow(TID\_Grow)$ can complete within $\min(4m - 2n + 2, 4k\Delta)$ rounds and the robot goes to *settled* state. Before this completion of the DFS, if the current component gets subsumed or subsumes another component, the robot moves to either *subsumed* or *collect* state. □

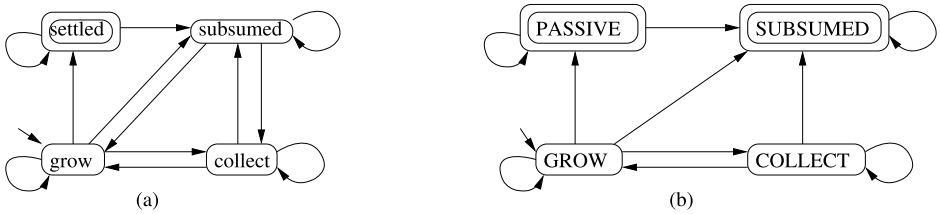

Fig. 1. State transition diagrams. (a) Diagram for a robot's state, *state*. (b) Diagram for any value of CID.

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. The DFS traversal of a component 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. □

Lemma 4.3. *For a DFS* Collect *for a CID CID_x:*

1. *The nodes in the DFS components of G subsumed by winner CID_x, corresponding to the nodes of S (in a connected component of S) subsumed by winner CID_x, form a connected component of G and for such nodes $CID = CID_x \neq CID\_old$.*
2. *The DFS* Collect *by the leader of winner CID_x traverses all these nodes having $CID = CID_x \neq CID\_old$ and their neighbors, and only these nodes.*

Proof. (Part 1). There are two possibilities from *Subsume_Graph_Processing*. (a) All the nodes in the connected component of $S' = (C', L')$ are in a simple cycle. All the nodes in the cycle minus the CID_x node form a connected component (C'', L''). (b) There exists a node in the connected component of $S' = (C', L')$ that has no incoming edges, and CID_x is the CID of such a node. $C' \setminus \{CID_x\}$ is a connected component (C'', L'').

In either possibility, an edge (CID_y, CID_z) in L'', by definition of a *Subsume* broadcast, indicates the nodes in G belonging to the two DFS components of CID_y and CID_z are reachable from one to the other. The connected component (C'', L'') therefore implies that the union of the nodes contained in G in each of these nodes in C'' is all reachable from one another. Furthermore, all these are subsumed nodes by CID_x and from the algorithm, they set $CID = CID_x$ and $CID\_old \neq CID_x$. The lemma Part (1) follows.

(Part 2). From the algorithm pseudocode, observe that the DFS *Collect* by the leader of winner CID_x traverses all these nodes and their neighbors, and no other nodes. The lemma Part (2) follows. □

Theorem 4.4. *The algorithm $Graph\_Disperse\_DFS$ solves* Dispersion.

Proof. Each robot begins in $state = grow$. Let α denote $\min(4m - 2n + 2, 4k\Delta)$. We make the following observations about the state transition diagram of a robot given in Fig. 1(a).

1. A robot can enter *subsumed* state at most $k - 1$ times. In *subsumed* state, a robot can stay at most $\alpha \cdot k$ rounds before it changes *state* to *collect*. This follows from Lemma 4.3 – the leader of the winner CID_{winner} will complete its DFS *Collect* of the connected component having $CID = CID_{winner} \neq CID\_old$ and collect the subsumed robot to its home node. This DFS *Collect* completes within α unless it gets serially subsumed by another winner, but such subsumptions can happen at most $k - 1$ times. In DFS *Collect*, robots from the subsumed component move along with the leader only when the leader is backtracking. So even if the DFS is interrupted by another winner $winner'$, the partially completed DFS still belongs to a connected component having $CID = CID_{winner'} \neq CID\_old$. Hence the new DFS *Collect* for $winner'$ can complete, and as per Lemma 4.3, collect *all* subsumed robots.
   Note that from *subsumed* state, a robot x that has the same TID_Grow as the leader at the time the leader was so selected and is at the leader's home node, but then stays back at the home node, changes state to *grow* when the leader returns to the home node and has finished exploring all ports at the home node in DFS *Collect*($TID\_Collect$) (hence is virtually backtracking to its "parent" in DFS *Collect*). This transition happens within α rounds unless x gets serially subsumed by another winner, but such subsumptions can happen at most $k - 1$ times.
2. From *collect* state, within α rounds, a robot can go to *subsumed* state (which can happen at most $k - 1$ times), or go to *grow* state (which can happen at most $k - 1$ times) (Lemma 4.2).
3. In *grow* state, a robot can remain for at most α rounds, by when it may go to either *subsumed*, *collect*, or *settled* state for at most k times (Lemma 4.1).
4. From *settled* state, a robot goes to *subsumed* at most $k - 1$ times, and each such transition must happen within what we denote as β rounds, or else the robot remains permanently in *settled* state. We now bound β. If the robot makes a transition to *subsumed*, it is because it gets subsumed by some CID CID_y.
   (a) The maximum time that different robots with $TID\_Grow = CID_y$ spend in *grow* state is $k\alpha$, from (3) above.
   (b) The maximum time that different robots with $TID\_Collect = CID_y$ spend in *collect* state is $2(k-1)\alpha$, from (2) above.
   (c) The maximum time that different robots with $CID = CID_y$ spend in *subsumed* state is $(k-1)\alpha$, from (1) above.
   Thus, the sum of the sojourns in *settled* state is bounded by $(k-1)\beta$ which is $(k-1) \cdot (k\alpha + 2(k-1)\alpha + (k-1)\alpha)$.

Although there are cycles in the state transition diagram, a robot must exit *subsumed* at most $k-1$ times, *collect* state at most $k-1$ times, *grow* state at most k times, and each stay has a bounded sojourn. Also, a robot must exit *settled* state at most $k-1$ times with the bounded sojourn in each, or else it remains permanently in *settled* state. It then follows that within a finite, bounded number of rounds, a robot will be in *settled* state permanently. It will be settled as part of the DFS TID_Grow tree traversal it was last associated with within α further rounds (follows from Theorem 3.1). 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 CID_x as follows, depending on the state of the least ID robot whose TID_Grow value equals CID_x.

Definition 2.

$$state(CID_x) = \begin{cases} GROW & \text{if } min_j(j.TID\_Grow = CID_x).state = grow \\ COLLECT & \text{if } min_j(j.TID\_Grow = CID_x).state = collect \\ SUBSUMED & \text{if } min_j(j.TID\_Grow = CID_x).state = subsumed \text{ or} \\ & \nexists j \mid j.TID\_Grow = CID_x \\ PASSIVE & \text{if } min_j(j.TID\_Grow = CID_x).state = settled \end{cases}$$

The state transition diagram for the state of a CID value is shown in Fig. 1(b).

Theorem 4.5. *The algorithm Graph_Disperse_DFS terminates in* $2 \cdot 4k\Delta$ *rounds.*

Proof. By Theorem 4.4, each robot settles permanently within a finite, bounded number of rounds. We now determine the number of rounds more precisely.

If the state of CID CID_y ever enters SUBSUMED, the robots with $CID = CID_y$ just before the transition would be subsumed, collected, and assigned a new CID $CID_{x'}$ possibly $k-1$ times. Let the final such CID assigned be $CID_{x''}$. Note that $CID_{x''}$ would never have entered SUBSUMED state. The robots in question get associated with CID $CID_{x''}$, and the termination time is that of such a $CID_{x''}$ that never entered $SUBSUMED$ state.

Let CID_x denote the CID of any robot that settles in the last round of Algorithm *Graph_Disperse_DFS*. This CID_x has never entered SUBSUMED state and therefore its state has shuttled between GROW and COLLECT before reaching and ending in PASSIVE. We separately bound the number of rounds spent by CID_x in GROW state and in COLLECT state.

Let the DFS tree in GROW state be associated with TID_Grow_x. Observe that multiple sojourns of CID_x in GROW state are associated with the same TID_Grow_x. The DFS data structures associated with TID_Grow_x are never overwritten by another DFS in GROW state as the component CID_x is never subsumed (and independent DFS traversal data structures are maintained for the DFS *Grow* and DFS *Collect* phases). Within $4m - 2n + 2$ rounds, possibly spread across multiple sojourns in GROW state, the DFS associated with TID_Grow_x completes and every robot associated with it gets settled. Every robot associated with TID_Grow_x also gets settled within $4k\Delta$ rounds, as the DFS *Grow* visits each edge at most 4 times, and hence within $4k\Delta$ rounds, at least k nodes get visited.

CID_x can transit from GROW to COLLECT and back at most $k-1$ times because that is the maximum number of times CID_x can subsume another CID. Let the transition to COLLECT state occur $l, 0 \leq l \leq k-1$ times, let the number of rounds spent in COLLECT state on the jth transition to it, $1 \leq j \leq l$, be s_j. Each transition to COLLECT is followed by a successful DFS *Collect* traversal of the connected component C_j of nodes having $CID = CID_x \neq CID\_old$ (from Lemma 4.3); denote by N_j^{int}, the set of such nodes. These are the nodes in the subsumed DFS components that form the connected component C_j, from Lemma 4.3.

In the DFS *Collect* traversal of C_j by the leader for the jth transition to COLLECT state, the leader visits from each node in N_j^{int}, each adjacent edge at most 4 times. As $\sum_{j=1}^{l} |N_j^{int}| \leq k$, it follows that at most $4k\Delta$ edges are visited in DFS $Collect(TID\_Collect = CID_x)$ across all transitions to COLLECT state. Hence,

$$\sum_{j=1}^{l} s_j \leq 4k\Delta$$

The theorem follows by separately combining the number of rounds in DFS *Grow* and DFS *Collect* phases in terms of m, and separately combining the number of rounds in DFS *Grow* and DFS *Collect* phases in terms of $k\Delta$. □

Theorem 4.6. *Algorithm 2 (*Graph_Disperse_DFS*) requires* $O(\log(k+\Delta))$ *bits memory.*

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

Proof of Theorem 1.1(a): Follows from Theorems 4.4 – 4.6.

4.4. *A more time-efficient* DFS_Collect

The DFS *Collect* traversal of C_j is a naive DFS traversal. There are four types of edges traversed in it.

1. e_j^{int}: Edge between two nodes in C_j. These can be either forward tree edges or back edges.
2. e_j^{win}: Edge from a node in C_j to a node having $CID = CID_x = CID\_old$, i.e., back to the *winner*'s component.
3. e_j^{free}: Edge from a node in C_j to a free node.
4. e_j^{adj}: Edge from a node in C_j to a node in an adjacent component (having $CID_y \neq CID_x$).

Let the numbers of these four types of edges be m_j^{int}, m_j^{win}, m_j^{free}, m_j^{adj}. In the DFS *Collect* of C_j, each e_j^{int} is traversed at most 4 times, whereas each e_j^{win}, e_j^{free}, and e_j^{adj} is traversed exactly 2 times (once in the forward mode and once in the backtrack mode). This gives:

$$s_j \leq 4m_j^{int} + 2(m_j^{win} + m_j^{free} + m_j^{adj})$$

As C_j and $C_{j'}$ are not disjoint, it does not seem possible to bound $\sum_{j=1}^{l} s_j$, the cost of the *Collect* phases of a single *winner*, by $O(m)$. As a result, time complexity of Algorithm *Graph_Disperse_DFS* is $O(k\Delta)$. We now describe a more efficient DFS *Collect* procedure having time complexity $O(k)$, leading to the algorithm having $O(\min(m, k\Delta))$ time. The $O(m)$ is obtained because $m_j^{win} = m_j^{free} = m_j^{adj} = 0$ and only internal tree edges of C_j are traversed. The key idea is that only DFS tree edges of each subsumed component are traversed in collecting the robots in that tree. This lower time complexity comes at the cost of a space complexity of $O(\Delta + \log k)$ because Δ bits are required at a settled robot to mark (in the DFS *Grow* phase) whether the Δ neighbors of the node are tree edges or not.

Let a *junction node* be one which has robots having two or more TID_Grow values. When a leader x of the winner component is selected following a *Subsume* graph processing, it (serially) does a DFS *Collect* of each DFS tree of robot y having $y.TID\_Grow \neq x.TID\_Grow$. This DFS *Collect* is outlined as follows.

1. Go from the junction node to the root of the tree of y.
2. Perform a DFS traversal of the tree (Algorithm 1) with a different set of variables than those used in the *Grow* phase of that tree, with the following main changes.
   (a) Explore an edge only if it marked as a tree edge or is marked by the *child* pointer of r in the DFS *Grow* phase.
   This ensures that only tree edges are visited.
   (b) When backtracking to a node along an edge pointed to by *child* pointer of r in the DFS *Grow* phase, do not visit other edges in forward mode, but backtrack to the *parent*.
   This ensures that only e_j^{int} edges are visited.
   (c) Collect all robots to the parent node when backtracking to the parent, except along the path from the junction node to the root node.
   This ensures that the tree with TID $y.TID\_Grow$ remains a tree. If this traversal is interrupted due to another *Subsume_Graph_Processing*, the next leader traversing the tree will use the *child* pointer of the DFS *Collect* phase to continue the DFS traversal from where it left off.
3. When the DFS traversal returns to the root node, collect all robots from the root to the junction node, round-by-round, setting the current root as active node in each round.
   This preserves the tree structure in $y.TID\_Grow$ from root to the junction node in case of interruption by another *Subsume_Graph_Processing*.

During the DFS *Collect* traversal of a tree $y.TID\_Grow$, if a junction node (where $y.TID\_Grow \neq z.TID\_Grow$) is encountered, a DFS *Collect* traversal of tree $z.TID\_Grow$ is begun *recursively*. Cycles in these recursive calls are broken by broadcasting $z.TID\_Grow$ whenever the DFS *Collect* of $z.TID\_Grow$ is begun (and setting, using, and resetting the new *being_collected* variable appropriately).

Proof of Theorem 1.1(b): For the modified DFS *Collect* algorithm outlined in this section, only tree edges of the subsumed DFS trees are visited. A DFS tree having n_1 nodes is traversed in $2n_1$ rounds, collecting at least n_1 robots to its root. Plus at most $2n_1$ rounds traversing from the junction node to the root node of the tree, and back. With a maximum of k robots being collected by the various C_j, at most $4k$ rounds are required by the *Collect* phases for the various C_j. From Theorem 4.5, the *Grow* phases of CID_x take $O(\min(m, k\Delta))$ time. This is the overall time complexity.

The modified algorithm requires Δ bits to mark tree edges in the *Grow* phase. The extra variables needed to implement the modified *Collect* phase require $O(\log(k + \Delta))$ space. Combining with this same space complexity shown in Theorem 4.6, the total space complexity of the modified algorithm is $O(\Delta + \log k)$ bits per robot.

The theorem follows from the above reasoning and Theorems 4.4 – 4.6.

5. BFS algorithm for rooted arbitrary graphs (Theorem 1.2(a))

In this section, we present and analyze *Rooted_Graph_Disperse_BFS*, a BFS-based algorithm that solves Dispersion of $k \leq n$ robots on an arbitrary rooted n-node graph in $O(D\Delta(D + \Delta))$ time with $O(\log D + \Delta \log k)$ bits of memory at each robot using the local communication model. This algorithm assumes all $k \leq n$ robots are at a single node v_{root} in the initial configuration. In the next section, we show how to adapt this algorithm to the general case of robots on multiple nodes in the initial configuration.

5.1. Basic idea

The algorithm begins with all the robots at a single root node. A BFS tree is built level by level with the root at level 0. The graph is explored to identify the nodes at level $i + 1$. This exploration is initially carried out by doing a 2-neighborhood search by the robots settled at level i (Δ^2 rounds). This gives a count of the "demand" for robots to settle at level $i + 1$. The demand number of unsettled robots are borrowed from the root to try to settle at level $i + 1$. Unfortunately the demand is likely to be an overcount because a level

$i+1$ node may be reachable independently from multiple level i nodes. Yet that is not a big problem if the number of robots at the root exceeds this (overcount) demand.

The real problem arises if the number of robots at the root is less than this demand. Not having any topology information, the available robots at the root are distributed to the level i leaf nodes to only partially fulfill their demands. Now, (a) some level $i+1$ nodes may not be reached at all because of the partial fulfillment – leading to a deficit of robots for level $i+1$ nodes, while on the other hand, (b) some level $i+1$ nodes may be reached via multiple level i nodes – leading to a surplus of robots for level $i+1$ nodes. The biggest challenge is how to transfer robots efficiently from the surplus to the deficit without having any topology information. The procedure of returning the surplus robots to the root and redetermining demands by level i nodes, and borrowing robots from the root to meet these demands can be repeated multiple times but the problem of partially fulfilling the demands and resulting in problems (a) and (b) again, may occur again.

The algorithm cleverly bounds the number of iterations of this procedure to transfer robots from surplus to deficit to $\Delta+2$. In this process, the algorithm also simultaneously reduces the overcount demand to the actual demand. It does this by having the exploratory robots borrowed from the root perform some added rounds (2Δ) in each iteration to classify edges (u, v) from level i nodes to level $i+1$ as (i) to be uniquely taken to reach the level $i+1$ node v (type V for "valid"), or (ii) not to be taken in order that the level $i+1$ node v is uniquely reachable from another level i node (type I for "invalid"), or (iii) it is yet undetermined whether the level $i+1$ node v is uniquely reachable from u (type U for "unfinalized"). The movement of robots up and down the tree for meeting the demand and rebalancing the surplus/deficit to meet this goal takes $2i$ rounds. So the time complexity for extending the level i tree to level $i+1$ is $O((\Delta+2)(2\Delta+2i))$. This gives a $O(D\Delta(D+\Delta))$ time algorithm.

The algorithm is presented assuming a global communication model for two reasons. It is more compact, and it can be directly adapted in the next section to the multi-rooted case for the global communication model. While proving the correctness of the algorithm, we show how the global communication can be replaced by the local communication model without increasing the asymptotic complexity.

5.2. The algorithm – 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 3. 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 level i to level $i+1$: (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 *Determine_Leaf_Demand(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 *Populate_Next_Level*(i) iteratively by borrowing robots for exploration from v_{root}.

The iterative borrowing of robots in *Populate_Next_Level(i)* is done as follows. In each iteration, the leaf nodes' demands are accumulated up the tree in a convergecast-like manner by doing a broadcast $B1$ as detailed in the algorithm. In this process, each node in the tree and the root learns of the demands of the sub-trees rooted at each of its children. In parallel, the unsettled robots, if any, at the leaf nodes move up to the root. When this completes in i rounds, the root redistributes the available robots among its children's sub-trees as per their accumulated demands, and so on, down the tree for rebalancing. The downward motion of the robots from root to the leaf nodes takes another i rounds. The number of robots assigned by the root to a leaf node at level i may be up to the demand of that node, which is the number of its incident edges going to level $i+1$ nodes. 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. The robot earmarked to settle at the level $i+1$ node does a 1-neighborhood traversal and *invalidates* (I) the ports of all other 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* (V) port. The settled robots at level i then re-evaluate the demand for robots, based on the number of *unfinalized* (U) 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} in the next iteration and they are reassigned for the next iteration based on the renewed (and decreased) values of net demand for exploratory robots and move to the level i leaf nodes. The upward and downward movement takes $O(D)$ time. The algorithm guarantees that $\Delta+1$ iterations suffice for setting to *valid* status a sufficient number of ports of level i nodes leading to level $i+1$ nodes (sufficient to fill level $i+1$ using the available number of robots), after which a final iteration of robot movements up and down the tree reassigns the final demand based on the number of *valid* ports (each of which leads to an unique level $i+1$ node) and distributes up to those many robots among level $i+1$ nodes. The procedure *Populate_Next_Level*(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.

1. $nrobots$: the total number of robots at the root, v_{root}. Initialize as defined.
2. $level$: the level of a robot/node in the BFS tree. Initialize to 0.
3. i: the current maximum level of settled robots. Initialize to 0.
4. $demand[1 \ldots \Delta]$, where
   $demand_u[j]$ for a non-leaf node u is the demand for robots to populate level $i+1$ for sub-tree reachable via port j. Initialize to $\overline{0}$.
   $demand_u[j]$ for a leaf node u at level i has the following semantics.

$$demand_u[j] = \begin{cases} 0 & \text{if port } j \text{ does not go to a level } i+1 \text{ node} \\ U & \text{if port } j \text{ goes to a level } i+1 \text{ node via an unfinalized edge} \\ V & \text{if port } j \text{ goes to a level } i+1 \text{ node via an validated edge} \\ I & \text{if port } j \text{ goes to a level } i+1 \text{ node via an invalidated edge} \end{cases}$$

   Initialize to $\overline{0}$.

Algorithm 3: Algorithm *Rooted_Graph_Disperse_BFS* to solve DISPERSION in global model. r denotes a settled robot, if any, at that node.

Initialize: $nrobots \leftarrow$ number of robots; $level$, i, $demand[1 \dots \Delta]$, $child\_id[1 \dots \Delta]$; $parent\_id$, $parent$, $winner$, $lvlfull$
robot with lowest ID settles at root, $nrobots \leftarrow nrobots - 1$
while $nrobots > 0$ **do**
  Determine_Leaf_Demand(i)
  Populate_Next_Level(i)
  $i \leftarrow i + 1$

Determine_Leaf_Demand(i)

Each settled robot r at a leaf node u at level i does a 2-bounded DFS to count the 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 $demand_u[out_u] \leftarrow U$.
Wait until Δ^2 rounds are elapsed or synchronize

Populate_Next_Level(i)

while $[\sum_{leaf\ u} \sum_{j=1}^{\Delta} (1\ if\ demand_u[j] = U)] > 0 \bigwedge [\sum_{leaf\ u} \sum_{j=1}^{\Delta} (1\ if\ demand_u[j] = V)] \leq nrobots$ **do**
  if $i = 0$ **then**
    $\forall j, demand_{root}[j] \leftarrow 1$
  else if $i > 0$ **then**
    unsettled robots at level i move upwards to root using *parent* pointers of settled nodes along the path, in i rounds. In parallel, **Convergecast** from leaf nodes at level i to root is performed in i rounds as follows. Leaf node u broadcasts $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} (1\ if\ demand_u[j] = U))$. On receiving $B1(x, my\_id, y)$, my_id sets $demand[child\_id^{-1}[x]] \leftarrow y$. On receiving $B1$ from all children, my_id broadcasts $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} demand[j])$. Wait until i rounds are elapsed
  When root has updated $demand[j]$ for all children j, distribute $\min(nrobots, \sum_{j=1}^{\Delta} demand[j])$ robots among children after resetting $winner \leftarrow 0$ for each robot
  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 $demand[p] > 0$ $(demand[p] = U)$.
  On arrival at level $i+1$ node v from level i node u via (out_u, in_v), robot with lowest ID sets $winner \leftarrow 1$
  if $winner = 0$ **then**
    retrace back via in_v to u and wait
  else if $winner = 1$ **then**
    visit each neighbor w via (out_v, in_w). If $w (\neq u)$ is at level i, $demand_w[in_w] \leftarrow I$
    retrace back from v using in_v to u; $demand_u[out_u] \leftarrow V$
  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) or synchronize()
if $i = 0$ **then**
  $\forall j, demand_{root}[j] \leftarrow 1$ if $demand_{root}[j] = V$
else if $i > 0$ **then**
  unsettled robots at level i move upwards to root using *parent* pointers of settled nodes along the path, in i rounds. In parallel, **Convergecast** from leaf nodes at level i to root is performed in i rounds as follows. Leaf node u broadcasts $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} (1\ if\ demand_u[j] = V))$. On receiving $B1(x, my\_id, y)$, my_id sets $demand[child\_id^{-1}[x]] \leftarrow y$. On receiving $B1$ from all children, my_id broadcasts $B1(my\_id, parent\_id, \sum_{j=1}^{\Delta} demand[j])$. Wait until i rounds are elapsed
When root has updated $demand[j]$ for all children j: set $lvlfull \leftarrow 0$; If $nrobots \geq \sum_{j=1}^{\Delta} demand[j]$ then $lvlfull \leftarrow 1$. Distribute $\min(nrobots, \sum_{j=1}^{\Delta} demand[j])$ robots among children; $nrobots \leftarrow nrobots - \min(nrobots, \sum_{j=1}^{\Delta} demand[j])$
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 $demand[p] > 0$.
At a level i node u, on receiving $\leq \sum_{j=1}^{\Delta} (1\ if\ demand_u[j] = V)$ robots, send one robot x on each port $out \mid demand_u[out] = V$; $child\_id_u[out] \leftarrow x$
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

If the node u being referred to is clear from context, we sometimes omit the subscript u.

5. $child\_id[1 \dots \Delta]$, where $child\_id[j]$ is the ID of the child node (if any), reachable via port j. Initialize to $\overline{\perp}$.
6. $parent\_id$: the ID of the parent robot in the BFS tree. Initialize to $\perp$.
7. $parent$: to identify the port through which the parent node in the BFS tree is reached. Initialize to $\perp$.
8. $winner$: to uniquely select a robot among those that arrive at a level $i+1$ node, to mark the edge/outgoing port along which it arrived as V and all other edges/outgoing ports from level i to that level $i+1$ node as I. Initialize to 0.
9. $lvlfull$: a boolean to track whether all level $i+1$ nodes can have a settled robot. (Used in Algorithm 4.) 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* Populate_Next_Level(i) *(in Algorithm 3) terminates within* $\Delta + 1$ *iterations.*

Proof. Let e_i denote $\sum_{leaf\ u} \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = U)$, the number of unfinalized edges (i.e., not validated and not invalidated edges) going from level i to level $i+1$.

1. If $nrobots \geq e_i$, then each of the e_i unfinalized edges can be explored and all nodes in level $i+1$ accounted for by validating exactly one edge each among the ports at level i nodes (and invalidating all other unfinalized edges at level i nodes) – thus, the **while** loop can be exited after one iteration as $demand_u[j] \neq U$ for any u at level i and for any j and the first clause of the **while** loop condition is falsified.
2. If $nrobots < e_i$, then $nrobots$ robots will be pressed into service for exploration of level $i+1$, at least $\lceil nrobots/\Delta \rceil$ nodes at level $i+1$ will be visited uniquely in this iteration (i.e., not visited in earlier iterations) via unfinalized edges, and hence at least $\lceil nrobots/\Delta \rceil$ ports (edges) at level i, that are currently marked as $demand_u[j] = U$ will be validated with change $demand_u[j] = V$. For the next iteration of the **while** loop, e_i will be decreased by at least this amount and $\lceil nrobots/\Delta \rceil$ ports (edges) at level i will change $demand_u[j]$ to V. It follows that within $\Delta + 1$ iterations, one of the following will occur.
   (a) $>$ $nrobots$ ports at level i will be validated. The second clause of the **while** loop condition is falsified and the loop is exited.
   (b) $=$ $nrobots$ ports at level i will be validated and $\sum_{leaf\ u} \sum_{j=1}^{\Delta}(1 \text{ if } demand_u[j] = U)] = 0$. This happens because no ports were validated in iteration $\Delta + 1$ which is because in Δ iterations at least $nrobots$ ports could have been validated and if exactly $nrobots$ ports were validated, there were no remaining unfinalized ports in iteration $\Delta + 1$. The first clause of the **while** loop condition is falsified and the loop is exited.
   (c) $nrobots \geq e_i$ within the first Δ iterations. By the reasoning given above in part (1), in one additional iteration, the loop is exited.

In all cases, the loop is exited within $\Delta + 1$ iterations. □

Lemma 5.2. *A BFS tree is induced in the underlying graph by Algorithm* Rooted_Graph_Disperse_BFS *given in Algorithm 3.*

Proof. We show by induction on the hypothesis that "all nodes at distance i (along shortest path) from the root have a settled robot that is assigned $level = i$, or there are no more robots to assign to some such nodes." The hypothesis is clearly true for $level = 0$ and can be seen to be true for $level = 1$ by following the execution of the algorithm.

We now assume the hypothesis for $level = x$ $(x \geq 1)$ and prove it true for $level = x+1$. If level x is filled for iteration with $level = x-1$ and some robots are left over, the algorithm moves to $level = x$ iteration of the main loop, otherwise we are done with the proof. Procedure *Determine_Leaf_Demand(x)* correctly identifies all nodes at level $x+1$ and the number of unfinalized edges going to such nodes from level x nodes is set to $e_x = \sum_{u\ at\ level\ x} \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = U)$.

1. If $nrobots \geq e_x$, then after one iteration of the **while** loop of *Populate_Next_Level*(x), all nodes of level $x+1$ are assigned robots (i.e., all ports from level x leading to level $x+1$ are marked V or I) as $\sum_{u\ at\ level\ x} \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = U) = 0$. After one traversal of robots up and down the tree for correct distribution of robots, robots settle at all the level $x+1$ nodes. Each node at level $x+1$ has its corresponding outgoing port j from level x parent u set to $demand_u[j] = V$, with more robots left at v_{root} for populating higher levels if $nrobots > \sum_{u\ at\ level\ x} \sum_{j=1}^{\Delta}(1\ if\ demand_u[j] = V)$.
2. If $nrobots < e_x$, then (as argued in the proof of Lemma 5.1), it follows that within $\Delta + 1$ iterations of the **while** loop of *Populate_Next_Level*(x), one of the following will occur.
   (a) $>$ $nrobots$ ports at level x will be validated. The second clause of the **while** loop condition is falsified and the loop is exited. All the remaining robots ($nrobots$) can be accommodated at level $x+1$ nodes and some nodes at level $x+1$ will not be assigned any robots because the algorithm has run out of robots. There will be no further levels in the BFS tree.
   (b) $=$ $nrobots$ ports at level i will be validated and $\sum_{leaf\ u} \sum_{j=1}^{\Delta}(1 \text{ if } demand_u[j] = U)] = 0$. The first clause of the **while** loop condition is falsified and the loop is exited. Level $x+1$ is filled and there are no remaining robots.
   (c) e_x gets decreased and $nrobots \geq e_x$ within the first Δ iterations. By the reasoning given above in part (1) for $nrobots \geq e_x$, in one additional iteration, the loop is exited. Now either
      i. $\geq$ $nrobots$ edges have been validated (adding those validated before this iteration to those validated in this iteration), in which case $nrobots$ robots are accommodated at level $x+1$, and some nodes at level $x+1$ remain free because the algorithm has run out of robots (if $\geq$ is strictly $>$), or
      ii. $<$ $nrobots$ edges have been validated (adding those validated before this iteration and in this iteration), in which case all the nodes at level $x+1$ will be assigned and settled with robots, with more robots left at v_{root} for populating higher levels.

The correctness of the induced BFS tree follows. □

Theorem 5.3. *Algorithm 3 (*Rooted_Graph_Disperse_BFS*) solves* Dispersion *on single-rooted graphs in* $O(D\Delta(\Delta + D))$ *rounds and requires* $O(\log D + \Delta \log k))$ *memory using 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. *Determine_Leaf_Demand(i)* does 2-neighborhood traversals in parallel, and hence takes $O(\Delta^2)$ rounds.
2. In each of the $\Delta + 1$ iterations of the **while** loop of *Populate_Next_Level*(i) (Lemma 5.1), the upward movement and the downward movement of the robots takes $2i$ rounds and the following code block based on the value of $winner$ takes 2Δ rounds. This is followed by one upward and downward movement of the robots outside the **while** loop, which takes $2i$ rounds.

So the time complexity for each iteration of the main **while** loop is $\Delta^2 + (\Delta+1)(2\Delta+2i)+2i$. 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 *Rooted_Graph_Disperse_BFS*. Thus the overall time complexity is $\sum_{i=1}^{D} 1(\Delta^2+(\Delta+1)(2\Delta+2i)+2i) = O(D\Delta(\Delta+D))$.

The variable *nrobots* takes $\log k$ bits, *level* and i take $\log D$ bits each, *demand*$[1\ldots\Delta]$ takes $\Delta \log k$ bits, *child_id*$[1\ldots\Delta]$ takes $\Delta \log k$ bits, *parent_id* takes $\log k$ bits, *parent* takes $\log\Delta$ bits, and *winner* takes 1 bit. The synchronize() can be implemented by a $\log k$ bit counter.

The local communication model suffices because we can simulate the global communication in the pseudocode by local communication without increasing the asymptotic complexity.

1. The directed broadcasts $B1$ can be simulated by each settled robot moving up to its parent and back, to communicate the demand of its sub-tree.
2. Also, to evaluate the condition in line (11), leaf nodes trigger a convergecast up the tree wherein each settled robot moves one level up to its parent to convey its subtree variables; it waits there until the root evaluates the condition; the result is read in a tree broadcast-like action down the tree wherein each waiting child at the parent's original position locally reads the result and moves down one level to its original position to convey the result to its children who are waiting there.
3. The *synchronize* can be implemented using a mechanism similar to that in point 2 above.

The theorem follows. □

6. BFS algorithm for arbitrary graphs (Theorem 1.2(b))

In this section, we adapt the single-rooted algorithm *Rooted_Graph_Disperse_BFS* (Algorithm 3) to the multi-rooted case. We present and analyze *Graph_Disperse_BFS* (Algorithm 4), which is a BFS-based algorithm that solves Dispersion of $k \leq n$ robots on an arbitrary n-node graph in $O((D+k)\Delta(D+\Delta))$ time with $O(\log D + \Delta \log k)$ bits of memory at each robot using the global communication model. This algorithm has lower run-time than the $O(\Delta^D)$ time of the best previously known algorithm [19] for arbitrary graphs (Table 1) using the local communication model. Furthermore, it is the first such algorithm using the BFS approach to solve Dispersion. This algorithm also contributes to BFS tree creation in distributed systems with concurrently initiated BFS tree creations from different nodes.

6.1. The algorithm – general case

We adapt the single-rooted algorithm to the multi-rooted case, see Algorithm 4. From each root, a BFS tree is initiated in parallel. A tree is identified by *root*, the robot ID settled at the root. When two (or more) BFS trees meet at a node, a collision is detected and a *Collide* message is broadcast. This collision detection may happen in *Determine_Leaf_Demand* or in *Populate_Next_Level*. In *Collision_Processing*, a tree with the highest depth among the colliding trees is chosen to *subsume* the other tree(s) it has directly collided with, and a corresponding *Subsume* message(s) is broadcast. Settled robots in a subsumed tree change their state and are not settled any more, and all the trees that are subsumed disappear and join the subsuming tree which will have maximal depth. (All subsuming trees will have maximal depth because the others with lower depth are not active.) In *Subsume_Processing*, the robots of the other (subsumed) tree(s) are collected at the root of the subsuming tree. The subsuming tree then continues the BFS algorithm at the same depth by executing *Populate_Next_Level* again if $nrobots = 0$ and $lvlfull = false$ (which tracks if the level has not been filled completely), i.e., level $i+1$ may not be fully populated yet, where i is the current maximum level of settled robots. Collisions may occur again in *Populate_Next_Level* and hence *Collision_Processing* and *Subsume_Processing* may have to be re-executed. This may repeat but each time, at least one BFS tree gets subsumed and hence the total number of serial executions of the three procedures is bounded by $k-1$. In the execution of this algorithm, there are synchronize() statements to ensure all tree growths occur in lock-step. Such barrier synchronizations can be implemented in the global communication model by a $\log k$-sized counter.

A collision is detected in tree with root *root* when a robot visits a node at level $i+1$, where i is the current maximum level of settled robots, from node u along outgoing port out_u. A 4-tuple collision record $T = \langle root, i+1, u, out_u\rangle$ is associated with this direction of the exploration collision. In the code, we refer to the third and fourth parameters as *bordernode* and *borderport*, respectively. A message $Collide(\langle root, i+1, u, out_u\rangle, \langle root', lvl', v, out_u\rangle)$ is broadcast to indicate that the exploration denoted by the first parameter has collided with the second parameter exploration/tree.

A unique robot executes *Collision_Processing* to process the *Collide* messages. It creates an undirected *Collision* graph $G_C = (V_C, E_C)$, where

$$V_C = \{T.root \mid Collide(T,*) \text{ or } Collide(*,T) \text{ message is received }\}$$

$$E_C = \{(T.root, T'.root) \mid \text{ at least one } Collide(T,T') \text{ message is received }\}$$

It then creates a Maximal Independent Set (MIS) M from among those nodes $T.root$ of V_C having $T.depth = i+1$, where i is the current maximum level of settled robots. For each tree identified by $x \in M$, it includes adjacent trees identified by $T''.root \in V_C$ with which there has been a collision, in set $P(x)$. Such trees $T''.root$ may have $T''.depth \leq i+1$. (If $T''.depth < i+1$ then T'' is not active, (which can happen in line 28), and if $T''.depth = i+1$ then T'' is active (first instance of broadcast in line 37) or inactive (second instance of broadcast in line 37).) For each $x \in M$, trees in $P(x) \setminus \{x\}$ are asked to collapse and subsume into tree identified by x, by broadcasting $Subsume(T, T'')$; when they get subsumed by the tree $x \in M$, this unblocks the x tree. Note that if tree $T''.root \notin M$ is adjacent to multiple trees $x, x' \in M$, the tree $T''.root$ will be asked to collapse and be subsumed in to only one of those trees in M. However, $T''.root$ no longer blocks both x and x'. The MIS construction guarantees that if there are two adjacent trees of depth $i+1$ (hence active) in V_C, either one will be subsumed into the other or at least one of them will get subsumed by yet other(s). Hence, unblocking occurs. For trees $y \notin M$ such that $y.depth = i+1$, they must have a neighbor z such that $z \in M$ and tree z subsumes tree y. Thus, all active trees (which are those having level $i+1$), whether in or outside M, unblock or get subsumed.

Algorithm 4: Algorithm *Graph_Disperse_BFS* to solve DISPERSION in global model for multi-rooted case. r denotes a settled robot, if any, at that node.

1 Initialize: $nrobots \leftarrow$ number of robots; $level$, i, $demand[1 \dots \Delta]$, $child\_id[1 \dots \Delta]$, $parent\_id$, $parent$, $winner$, $lvlfull$
2 robot with lowest ID settles at root, $nrobots \leftarrow nrobots - 1$
3 **while** $nrobots > 0$ **do**
4 *Determine_Leaf_Demand*(i)
5 *Populate_Next_Level*(i)
6 *Collision_Processing*, *Subsume_Processing*, synchronize()
7 **while** $nrobots > 0 \wedge lvlfull = 0$ **do**
8 *Populate_Next_Level*(i)
9 *Collision_Processing*, *Subsume_Processing*, synchronize()
10 synchronize() and while waiting for synchronization execute *Subsume_Processing* when any *Subsume* messages received, $i \leftarrow i + 1$
11 Execute *Subsume_Processing* when any *Subsume* messages received
12 *Collision_Processing*: executed by robot r_{min} lowest ID robot
13 Using all *Collide* messages broadcast since the last execution of *Collision_Processing*,r_{min} creates an undirected Collision graph $G_C = (V_C, E_C)$.
14 $X \leftarrow V_C$
15 **while** $\exists x \in X$, *growing level* $i + 1$ **do**
16 $M \leftarrow M \cup \{x\}$, $P(x) \leftarrow \{x\} \cup \{$ 1-hop neighbors of $x\}$, $X \leftarrow X \setminus P(X)$
17 $\forall x \in X, \forall y \in P(x) \setminus \{x\}$ do: among *Collide*(T, T') received such that $(T.root = x \wedge T'.root = y)$, select one and broadcast *Subsume*(T, T'); if no such selected then among *Collide*(T, T') received such that $(T.root = y \wedge T'.root = x \wedge T.lvl = T'.lvl = i + 1)$, select one and broadcast *Subsume*(T', T)
18 *Subsume_Processing*: process received *Subsume(T,T')* messages as follows.
19 **if** *robot j is waiting (without settling) at node v at level* $i + 1$ *and* $j.root = T.root$ **then**
20 j sets $level \leftarrow i + 1$, $parent\_id \leftarrow u$ (its parent's ID), $parent \leftarrow in$ (port through which it entered), j settles at v.
21 **else if** *robot j is waiting (without settling) at node v at level* $i + 1$ *and* $j.root = T'.root$ **then**
22 j retraces its step back to u (its parent), (then moves on to the root of the tree that subsumes its tree as described next.)
23 **else if** *robot j is waiting (without settling) at node v at level* $i + 1$ *and no* Subsume *message has been received such that* $j.root = T.root$ *or* $j.root = T'.root$ **then**
24 j sets $level \leftarrow i + 1$, $parent\_id \leftarrow u$ (its parent's ID), $parent \leftarrow in$ (port through which it entered), j settles at v.
25 For message *Subsume*(T, T'), $T'.bordernode$ identifies a path H from $T'.root$ to $T'.bordernode$ by nodes along H serially broadcasting B2(my_id, $parent\_id$) progressively up the path from $T'.bordernode$ to $T'.root$ ($i + 1$ serial broadcasts suffice).
26 In parallel, all robots in T', except those robots along H, beginning from the leaf node robots move up tree T' to $T'.root$ using the *parent* pointers ($i + 1$ rounds suffice). The robots in T' then move down path H from $T'.root$ to $T'.bordernode$ ($i + 1$ rounds suffice); then to $T.bordernode$ via $T'.borderport$ and then up to $T.root$ using the *parent* pointers ($i + 1$ rounds suffice). $nrobots$ at $T.root$ is then incremented with the count of the newly arrived robots.
27 *Determine_Leaf_Demand*(i)
28 Each settled robot r at a leaf node u at level i does a 2-bounded DFS to count the number of neighbors v at level $i + 1$. On exploring (u, v) via out_u, (i) if v has a settled robot r' of another tree with root $root'$ and level lvl' ($lvl' < i + 1$), r broadcasts *Collide*$(\langle root, i + 1, u, out_u \rangle, \langle root', lvl', v, in_v \rangle)$ – then discount and backtrack, (ii) else if v is level $i - 1$ in my tree, then discount and backtrack, (iii) else if v has a level $i - 1$ neighbor in my tree, then v is level i node - discount and backtrack, (iv) else v is a level $i + 1$ node, hence robot r sets $demand_u[out_u] \leftarrow U$.
29 Wait until Δ^2 rounds are elapsed or synchronize().
30 *Populate_Next_Level*(i): changes to Algorithm 3 are given
31 **Lines 18-23: are to be executed only with reference to robots belonging to my own tree (having same root).**
32 **Line 32, replace by the following block:**
33 The robot x reaches node v at level $i + 1$ via incoming port in_v
34 **if** *no other robot from any tree has arrived at or is settled at* v **then**
35 $level \leftarrow i + 1$, $parent\_id \leftarrow u$, $parent \leftarrow in_v$, x settles at the node
36 **else**
37 For each other robot r' of tree with root $root'$ and level $i + 1$ that arrived from u' via $out_{u'}$, broadcast *Collide*$(\langle root, i + 1, u, out_u \rangle, \langle root', i + 1, u', out_{u'} \rangle)$. If robot r' of another tree with root $root'$ and level lvl' (= $i + 1$) is already settled at v, broadcast *Collide*$(\langle root, i + 1, u, out_u \rangle, \langle root', i + 1, v, in_v \rangle)$. Wait until *Subsume_Processing* is executed.

In *Subsume_Processing*, for each received message *Subsume*(T, T'), where $T = \langle root, i + 1, u, out_u \rangle$ and $T' = \langle root', lvl', v, out_u \rangle$, the path H from $root'$ to v is identified in a backwards manner, via parents, as follows. Nodes along H beginning from *bordernode* leaf node v serially broadcast $B2(my\_id, parent\_id)$ progressively up the path from v to $root'$ ($i + 1$ serial broadcasts suffice). On receiving $B2(x, my\_id)$ at robot my_id, the next node on the path H is x, reachable via outgoing port $child\_id^{-1}[x]$, and my_id also broadcasts B2(my_id, $parent\_id$) to continue identifying the previous nodes along the path. (The path H can alternately be identified by each robot beginning with v serially moving up to its parent (to let it identify which port to take towards v) and back to its original position.)

All robots in T' except those along H move upwards and collect to $root'$. They move along H down to v, then to u, and upwards in T to $root$. These movements take $3(i + 1)$ steps.

Theorem 6.1. *Algorithm 4 (*Graph_Disperse_BFS*) solves* DISPERSION *in multi-rooted graphs in* $O((D + k)\Delta(\Delta + D))$ *rounds and requires* $O(\log D + \Delta \log k))$ *memory using the global communication model.*

Proof. Each concurrently-initiated BFS tree grows until it collides with another (or runs out of robots). When two (or more) trees collide, then in the processing of the *Collision* graph we have the following. (1) For each $x \in M$, trees in $P(x) \setminus \{x\}$ are asked to collapse and subsume into tree identified by x, by broadcasting a *Subsume* message; when they get subsumed by the tree $x \in M$, this unblocks the x tree. (2) For trees $y \notin M$ such that $y.depth = i + 1$, they must have a neighbor z such that $z \in M$ and tree z subsumes tree y. Thus, all active trees (which are those having level $i + 1$) unblock or get subsumed. On unblocking, the tree continues to grow. It can collide at most $k - 1$ times, hence at most $k - 1$ invocations of *Collision_Processing* and *Subsume_Processing*. It continues to grow until termination (exhausting all the unsettled robots at its root). On termination, one robot is settled at each distinct node of the BFS tree(s). Hence Dispersion is achieved.

Building on the proof of Theorem 5.3 for Algorithm 3, in addition to $O(\sum_{i=1}^{D} \Delta^2 + i\Delta) = O(\Delta D(\Delta + D))$ rounds, we have at most $k-1$ executions of *Populate_Next_Level*, *Collision_Processing*, and *Subsume_Processing*. Let there be x such iterations corresponding to x serial subsumptions by the tree in question. Let the depth (level) of the tree for the jth subsumption be d_j. Then the following additional time cost is incurred:

$$\sum_{j=1}^{x} (\Delta + 1)(2\Delta + 2d_j) + 2d_j \text{ (for } \textit{Populate\_Next\_Level}\text{, see proof of Theorem 5.3)}$$
$$+ \sum_{j=1}^{x} 3d_j \text{ (for } \textit{Subsume\_Processing}\text{)},$$

$d_j < D$ and $x < k$. Thus the additional time is $O(\Delta^2 k + 2kD\Delta + 3Dk)$. The total time complexity follows.

The logic introduced in the multi-rooted algorithm adds variables ($\log k$ sized counters for synchronize()) that do not increase the bit complexity.

The global communication model needs to be used for issuing and processing the broadcasts. The theorem follows. □

Proof of Theorem 1.2: Follows from Theorems 5.3 and 6.1.

7. Concluding remarks

We have presented two results for solving Dispersion of $k \leq n$ robots on n-node arbitrary graphs. The first result is based on a DFS traversal and the algorithm performs better than the best previously known algorithm using the local communication model by a $O(\log k)$ factor, with an additional $O(\Delta)$ bits in memory. The second algorithm is based on a BFS traversal and runs significantly faster than the $O(\Delta^D)$ time of the previously known algorithm using 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 $\Omega(\min(m, k\Delta))$. Furthermore, it will be interesting to achieve $O(\min(m, k\Delta))$ time bound in Theorem 1.1(b) with only $O(\log(k + \Delta))$ bits per robot, removing the additional $O(\Delta)$ bits factor. For BFS-based algorithms, it will be interesting to improve the $(D + k)$ factor to $O(D)$ for arbitrary graphs. The fourth interesting direction will be to consider faulty robots. The fifth interesting direction will be to extend our algorithms to semi-synchronous and asynchronous settings.

CRediT authorship contribution statement

Ajay D. Kshemkalyani: Formal analysis, Investigation, Methodology, Writing – original draft, Writing – review & editing. **Anisur Rahaman Molla:** Formal analysis, Investigation, Validation. **Gokarna Sharma:** Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Supervision, Writing – original draft.

Declaration of competing interest

On behalf of all the authors, I, Gokarna Sharma, declare that we have no competing or conflict of interest with any of the referees suggested.

References

[1] J. Augustine, W.K.M. Jr., Dispersion of mobile robots: a study of memory-time trade-offs, in: ICDCN, 2018, 1.
[2] E. Bampas, L. Gasieniec, N. Hanusse, D. Ilcinkas, R. Klasing, A. Kosowski, Euler tour lock-in problem in the rotor-router model: I choose pointers and you choose port numbers, in: DISC, 2009, pp. 423–435.
[3] L. Barriere, P. Flocchini, E. Mesa-Barrameda, N. Santoro, Uniform scattering of autonomous mobile robots in a grid, in: IPDPS, 2009, pp. 1–8.
[4] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, D. Peleg, Label-guided graph exploration by a finite automaton, ACM Trans. Algorithms 4 (4) (2008) 42.
[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009.
[6] G. Cybenko, Dynamic load balancing for distributed memory multiprocessors, J. Parallel Distrib. Comput. 7 (2) (1989) 279–301.
[7] S. Das, P. Flocchini, G. Prencipe, N. Santoro, M. Yamashita, Autonomous mobile robots with lights, Theor. Comput. Sci. 609 (2016) 171–184.
[8] S. Das, D. Dereniowski, C. Karousatou, Collaborative exploration of trees by energy-constrained mobile robots, Theory Comput. Syst. 62 (5) (2018) 1223–1240.
[9] D. Dereniowski, Y. Disser, A. Kosowski, D. Pajak, P. Uznański, Fast collaborative graph exploration, Inf. Comput. 243 (C) (2015) 37–49.
[10] Y. Disser, F. Mousset, A. Noever, N. Skoric, A. Steger, A general lower bound for collaborative tree exploration, CoRR, arXiv:1610.01753 [abs].
[11] Y. Elor, A.M. Bruckstein, Uniform multi-agent deployment on a ring, Theor. Comput. Sci. 412 (8–10) (2011) 783–795.
[12] P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Oblivious Mobile Robots, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2012.
[13] P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Mobile Entities, Theoretical Computer Science and General Issues, vol. 1, Springer International Publishing, 2019.
[14] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, D. Peleg, Graph exploration by a finite automaton, Theor. Comput. Sci. 345 (2–3) (2005) 331–344.

[15] P. Fraigniaud, L. Gasieniec, D.R. Kowalski, A. Pelc, Collective tree exploration, Networks 48 (3) (2006) 166–177.
[16] T. Hsiang, E.M. Arkin, M.A. Bender, S.P. Fekete, J.S.B. Mitchell, Algorithms for rapidly dispersing robot swarms in unknown environments, in: WAFR, 2002, pp. 77–94.
[17] T.-R. Hsiang, E.M. Arkin, M.A. Bender, S. Fekete, J.S.B. Mitchell, Online dispersion algorithms for swarms of robots, in: SoCG, 2003, pp. 382–383.
[18] A.D. Kshemkalyani, F. Ali, Fast graph exploration by a mobile robot, in: First IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2018, 2018, pp. 115–118.
[19] A.D. Kshemkalyani, F. Ali, Efficient dispersion of mobile robots on graphs, in: ICDCN, 2019, pp. 218–227.
[20] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Fast dispersion of mobile robots on arbitrary graphs, in: ALGOSENSORS, 2019, pp. 23–40.
[21] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Dispersion of mobile robots in the global communication model, in: ICDCN, 2020, 12.
[22] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Dispersion of mobile robots on grids, in: WALCOM, 2020, pp. 183–197.
[23] A. Menc, D. Pajak, P. Uznanski, Time and space optimality of rotor-router graph exploration, Inf. Process. Lett. 127 (2017) 17–20.
[24] A.R. Molla, W.K.M. Jr., Dispersion of mobile robots: the power of randomness, in: TAMC, 2019, pp. 481–500.
[25] C. Ortolf, C. Schindelhauer, Online multi-robot exploration of grid graphs with rectangular obstacles, in: SPAA, 2012, pp. 27–36.
[26] P. Poudel, G. Sharma, Time-optimal uniform scattering in a grid, in: ICDCN, 2019, pp. 228–237.
[27] M. Shibata, T. Mega, F. Ooshita, H. Kakugawa, T. Masuzawa, Uniform deployment of mobile agents in asynchronous rings, in: PODC, 2016, pp. 415–424.

Ajay D. Kshemkalyani (Senior Member, IEEE) received the BTech degree in computer science and engineering from the Indian Institute of Technology, Bombay, India, in 1987, and the MS and PhD degrees in computer and information science from The Ohio State University, Columbus, Ohio, in 1988 and 1991, respectively. He spent six years at IBM Research Triangle Park working on various aspects of computer networks, before joining academia. He is currently a professor with the Department of Computer Science, University of Illinois at Chicago, Chicago, Illinois. His research interests are in distributed computing, distributed algorithms, computer networks, and concurrent systems. In 1999, he received the National Science Foundation Career Award. He served on the editorial board of the Elsevier Journal Computer Networks and the IEEE Transactions on Parallel and Distributed Systems. He has co-authored a book entitled Distributed Computing: Principles, Algorithms, and Systems (Cambridge University Press, 2011). He is a distinguished scientist of ACM.

Anisur Rahaman Molla is an Assistant Professor in the Cryptology and Security Research Unit at the Indian Statistical Institute (ISI), Kolkata, India. He received his Ph.D. from Nanyang Technological University, Singapore. He has held a faculty position at the National Institute of Science Education and Research (NISER) Bhubaneswar. He did Postdoc from the University of Freiburg, Germany and Singapore University of Technology and Design (SUTD). He is a member of the ACM. He serves as a program committee member for several international conferences. He also serves as an external reviewer for numerous international journals and conferences and has given several invited talks and tutorials. His research interests are in distributed network algorithms, security in distributed computing and distributed mobile robots. His work is supported by DST, Govt. of India.

Gokarna Sharma is currently an Associate Professor in the Department of Computer Science, Kent State University, Kent, OH, USA. He received the bachelor's degree in computer engineering from Tribhuvan University, Kirtipur, Nepal, in 2005, the dual European Master of Science degree in computer science from the Free University of Bolzano, Bolzano, Italy and Vienna University of Technology, Vienna, Austria, in 2008, and the Ph.D. degree in computer science from Louisiana State University, Baton Rouge, LA, USA, in 2014. He interned as a Summer Consultant with the Bell Labs in summer 2008. His current research interests include distributed computing theory, systems, algorithms, and data structures, emerging technologies such as the Internet of Things and Blockchain, and robotics. In 2021, he received the US National Science Foundation CAREER Award. He is a senior member of the IEEE and a member of the ACM.