Fast Dispersion of Mobile Robots on Arbitrary Graphs

Ajay D. Kshemkalyani[1], Anisur Rahaman Molla[2], and Gokarna Sharma[3(✉)]

[1] University of Illinois at Chicago, Chicago, USA
ajay@uic.edu
[2] Indian Statistical Institute, Kolkata, India
molla@isical.ac.in
[3] Kent State University, Kent, USA
sharma@cs.kent.edu

Abstract. The dispersion problem on graphs asks $k \leq n$ robots placed initially arbitrarily on the nodes of an n-node anonymous graph to reposition autonomously to reach a configuration in which each robot is on a distinct node of the graph. This problem is of significant interest due to its relationship to other fundamental robot coordination problems, such as exploration, scattering, load balancing, and relocation of self-driven electric cars (robots) to recharge stations (nodes). In this paper, we provide a novel deterministic algorithm for dispersion in arbitrary graphs in a synchronous setting where all robots perform their actions in every time step. Our algorithm has $O(\min(m, k\Delta) \cdot \log k)$ steps runtime using $O(\log n)$ bits of memory at each robot, where m is the number of edges and Δ is the maximum degree of the graph. This is a significant improvement over the $O(mk)$ steps best previously known algorithm that uses logarithmic memory at each robot. In particular, the runtime of our algorithm is optimal (up to a $O(\log k)$ factor) in constant-degree arbitrary graphs.

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 [14,15]. 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 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 [1,16]. 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 [1,16]. One

A. R. Molla was supported in part by DST Inspire Faculty research grant DST/INSPIRE/04/2015/002801.

F. Dressler and C. Scheideler (Eds.): ALGOSENSORS 2019, LNCS 11931, pp. 23–40, 2019.
https://doi.org/10.1007/978-3-030-34405-4_2

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 [10,11]. In this paper, we study trade-off between memory requirement and time to solve DISPERSION.

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, where D is the diameter of the 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, where m is the number of edges in the graph. Recently, Kshemkalyani and Ali [16] 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 $O(\log(\max(k, \Delta)))$ bits at each robot with $O(mk)$ time, where Δ is the maximum degree of the graph. Randomized algorithms are presented in [18] to solve DISPERSION where the random bits are mainly used to reduce the memory requirement at each robot.

In this paper, we provide a new deterministic algorithm for solving DISPERSION in arbitrary graphs. Our algorithm improves significantly on the runtime of the best previously known algorithm with logarithmic memory at each robot; see Table 1.

Overview of the Model and Results. We consider the same model as in Augustine and Moses Jr. [1] and Kshemkalyani and Ali [16] where a system of $k \leq n$ robots are operating on an n-node anonymous graph G. The robots are *distinguishable*, i.e., they have unique IDs in the range $[1, k]$. The robots have no visibility; but they can communicate with each other only when they are at the same node of G. The graph G is assumed to be connected and undirected. The nodes of G are indistinguishable (G is anonymous) but the ports (leading to incident edges) at each node have unique labels from $[1, \delta]$, where δ is the degree of that node. It is assumed that the robots know m, n, Δ, k[1]. Similar assumptions are made in the previous work in DISPERSION [1]. The nodes of G do not have memory and the robots have memory. *Synchronous* setting is considered as in [1] where all robots are activated in a round and they perform their operations simultaneously in synchronized rounds. Runtime is measured in rounds (or steps). We establish the following theorem in an arbitrary graph.

Theorem 1. *Given any initial configuration of $k \leq n$ mobile robots in an arbitrary, anonymous n-node graph G having m edges and maximum degree Δ,* DISPERSION *can be solved in $O(\min(m, k\Delta) \cdot \log k)$ time with $O(\log n)$ bits at each robot.*

Theorem 1 improves significantly over the $O(mk)$ time algorithm of [16] with logarithmic memory (Table 1). Notice that, when $\Delta \leq k$, the runtime depends only on k,

[1] In fact, it is enough to know only m, Δ and k to accomplish the results. Without robots knowing m, Theorem 1 achieves DISPERSION in $O(k\Delta \cdot \log k)$ time with $O(\log(\max(k, \Delta)))$ bits memory at each robot, which is better in terms of memory of $O(\log n)$ bits in Theorem 1 but not the time $O(\min(m, k\Delta) \cdot \log k)$ when $m < k\Delta$.

i.e., $O(k^2 \log k)$. For constant-degree arbitrary graphs (i.e., when $\Delta = O(1)$), the time becomes near-optimal – only a $O(\log k)$ factor away from the time lower bound $\Omega(k)$.

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 per robot (in bits)	Time (in rounds)
Lower bound	$\Omega(\log(\max(k, \Delta)))$	$\Omega(k)$
First algorithm of [1][a]	$O(\log n)$	$O(mn)$
Second algorithm of [1]	$O(n \log n)$	$O(m)$
First algorithm of [16]	$O(k \log \Delta)$	$O(m)$
Second algorithm of [16]	$O(D \log \Delta)$	$O(\Delta^D)$
Third algorithm of [16]	$O(\log(\max(k, \Delta)))$	$O(mk)$
Theorem 1	$O(\log n)$	$O(\min(m, k\Delta) \cdot \log k)$

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

Challenges and Techniques. The well-known *Depth First Search* (DFS) traversal approach [5] was used in the previous papers to solve DISPERSION [1,16]. If all k robots are positioned initially on a single node of G, then the DFS traversal finishes in $\min(4m - 2n + 2, k\Delta)$ rounds solving DISPERSION. If k robots are initially on k different nodes of G, then DISPERSION is solved by doing nothing. However, if not all of them are on a single node initially, then the robots on nodes with multiple robots need to reposition (except one) 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. Care is needed here otherwise they may re-traverse sequentially and the total time for the DFS traversal increases by a factor of k to $\min(4m - 2n + 2, k\Delta) \cdot k$ rounds, in the worst-case. This is in fact the case in the previous algorithms of [1,16]. We design a smarter way to synchronize the parallel DFS traversals so that the total time increases only by a factor of $\log k$ to $\min(4m - 2n + 2, k\Delta) \cdot \log k$ rounds, in the worst-case. This approach is a non-trivial extension and requires overcoming many challenges on synchronizing the parallel DFS traversals efficiently.

Related Work. One problem closely related to DISPERSION is the graph exploration. The exploration problem has been heavily studied in the literature for specific as well as arbitrary graphs, e.g., [2,4,8,13,17]. It was shown that a robot can explore an anonymous graph using $\Theta(D \log \Delta)$-bits memory; the runtime of the algorithm is $O(\Delta^{D+1})$ [13]. 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 [17]. The collective exploration by a team of robots is studied in [12] for trees. Another problem related to DISPERSION

is the scattering of k robots in graphs. This problem has been studied for rings [9,20] and grids [3]. Recently, Poudel and Sharma [19] 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., [6,21]. We refer readers to [10,11] for other recent developments in these topics.

Paper Organization. We discuss details of the model and some preliminaries in Sect. 2. We discuss the DFS traversal of a graph in Sect. 3. We present an algorithm for arbitrary graphs in Sect. 4. Finally, we conclude in Sect. 5 with a short discussion.

2 Model Details and Preliminaries

Graph. We consider the same graph model as in [1,16]. Let $G = (V, E)$ be an n-node m-edge graph, i.e., $|V| = n$ and $|E| = m$. G is assumed to be connected, unweighted, and undirected. G is *anonymous*, i.e., nodes do not have identifiers but, at any node, its incident edges are uniquely identified by a *label* (aka port number) in the range $[1, \delta]$, where δ is the *degree* of that node. The *maximum degree* of G is Δ, which is the maximum among the degree δ of the nodes in G. We assume that there is no correlation between two port numbers of an edge. Any number of robots are allowed to move along an edge at any time. The graph nodes do not have memory.

Robots. We also consider the same robot model as in [1,16]. 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 sometime use i to denote robot r_i. No robot can reside on the edges of G, but one or more robots can occupy the same node of G. Each robot has a unique $\lceil \log k \rceil$-bit ID taken from $[1, k]$. Robot has no visibility and hence a robot can only communicate with other robots present on the same node. Following [1,16], it is assumed that 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 on the same node. Each robot is assumed to know parameters m, n, Δ, k. Such assumptions are also made in the previous work on DISPERSION [1].

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

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

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

Definition 1 (DISPERSION). *Given any n-node anonymous graph $G = (V, E)$ having $k \leq n$ 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, and (ii) **Memory** – the number of bits stored at each robot.

Table 2. Description of the variables used in Sects. 3 and 4. These variables are maintained by each robot and may be read/updated by other robots (at the same node).

Symbol	Description
$round$	The counter that indicates the current round. Initially, $round \leftarrow 0$
$pass$	The counter that indicates the current pass. Initially, $pass \leftarrow 0$
$parent$	The port from which robot entered a node in forward phase. Initially, $parent \leftarrow 0$
$child$	The smallest port (except $parent$ port) that was not taken yet. Initially, $child \leftarrow 0$
$treelabel$	The label of a DFS tree. Initially, $treelabel \leftarrow \top$
$settled$	A boolean flag that stores either 0 (false) or 1 (true). Initially, $settled \leftarrow 0$
$mult$	The number of robots at a node at the start of Stage 2. Initially, $mult \leftarrow 1$
$home$	The lowest ID unsettled robot at a node at the start of Stage 2 sets this to the ID of the settled robot at that node. Initially, $home \leftarrow \top$

3 DFS Traversal of a Graph

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

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

We are now ready to describe $DFS(k)$. In round 1, the maximum ID robot r_k writes $r_k.treelabel \leftarrow 1$ (the ID of the smallest robot in $N(v)$, which is 1), $r_k.child \leftarrow 1$ (the smallest port at v among $P(v)$), and $r_k.settled \leftarrow 1$. The robots $N(v) \backslash \{r_k\}$ exit v

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

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

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

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

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

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

Lemma 1. *Algorithm $DFS(k)$ correctly solves* DISPERSION *for $k \leq n$ robots initially positioned on a single node of a n-node arbitrary graph G in $\min(4m - 2n + 2, k\Delta)$ rounds using $O(\log(\max(k, \Delta)))$ bits at each robot.*

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. In $k\Delta$ rounds, $DFS(k)$ visits at least k different nodes of G. If $4m - 2n + 2 < k\Delta$, $DFS(k)$ visits all n nodes of G. Therefore, it is clear that the runtime of $DFS(k)$ is $\min(4m - 2n + 2, k\Delta)$ rounds. Regarding memory, variable $treelabel$ takes $O(\log k)$ bits, $settled$ 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 $O(\log(\max(k, \Delta)))$ bits. □

4 Algorithm

We present and analyze an algorithm, *Graph_Disperse(k)*, that solves DISPERSION of $k \leq n$ robots on an arbitrary n-node graph in $O(\min(m, k\Delta)\cdot\log k)$ time with $O(\log n)$ bits of memory at each robot. This algorithm significantly improves the $O(mk)$ time of the best previously known algorithm [16] for arbitrary graphs (Table 1).

4.1 High Level Overview of the Algorithm

Algorithm *Graph_Disperse(k)* runs in passes and each pass is divided into two stages. Each pass runs for $O(\min(m, k\Delta))$ rounds and there will be total $O(\log k)$ passes until DISPERSION is solved. The algorithm uses $O(\log n)$ bits memory at each robot. To be able to run passes and stages in the algorithm, we assume following [1] that robots know n, m, k, and Δ. At their core, each of the two stages uses a modified version of the DFS traversal by robots (Algorithm $DFS(k)$) described in Sect. 3.

At the start of stage 1, there may be multiple nodes, each with more than one robot (top left of Fig. 1). The (unsettled) robots at each such node begin a DFS in parallel, each such DFS instance akin to $DFS(k)$ described in Sect. 3. Each such concurrently initiated DFS induces a DFS tree where the $treelabel$ of the robots that settle is common, and the same as the ID of the robot with the smallest ID in the group.

Unlike $DFS(k)$, here a DFS traversal may reach a node where there is a settled robot belonging to another (concurrently initiated) DFS instance. As the settled robot cannot track variables ($treelabel$, $parent$, $child$) for the multiple DFS trees owing to its limited memory, it tracks only one DFS tree instance and the other DFS instance(s) is/are stopped. Thus, some DFS instances may not run to completion and some of their robots may not be settled by the end of stage 1. Thus, groups of stopped robots exist at different nodes at the end of stage 1 (top right of Fig. 1).

In stage 2, all the groups of stopped robots at different nodes in the same connected component of nodes with settled robots are gathered together into one group at a single node in that connected component (bottom left of Fig. 1). Since stopped robots in a group do not know whether there are other groups of stopped robots, and if so, how many and where, one robot from each such group initiates a DFS traversal of its connected component of nodes with settled robots, to gather all the stopped robots at its starting node. The challenge is that due to such parallel initiations of DFS traversals, robots may be in the process of movement and gathering in different parts of the connected component of settled nodes. The algorithm ensures that despite the unknown number of concurrent initiations of the DFS traversals for gathering, all stopped robots in a connected component of settled robots get collected at a single node in that component at the end of stage 2. Our algorithm has the property that the number of nodes with such gathered (unsettled) robots in the entire graph at the end of stage 2 is at most half the number of nodes with more than one robot at the start of stage 1 (of the same pass). This implies the sufficiency of $\log k$ passes, each comprised of these two stages, to collect all graph-wide unsettled robots at one node. In the first stage of the last pass, DISPERSION is achieved (bottom right of Fig. 1).

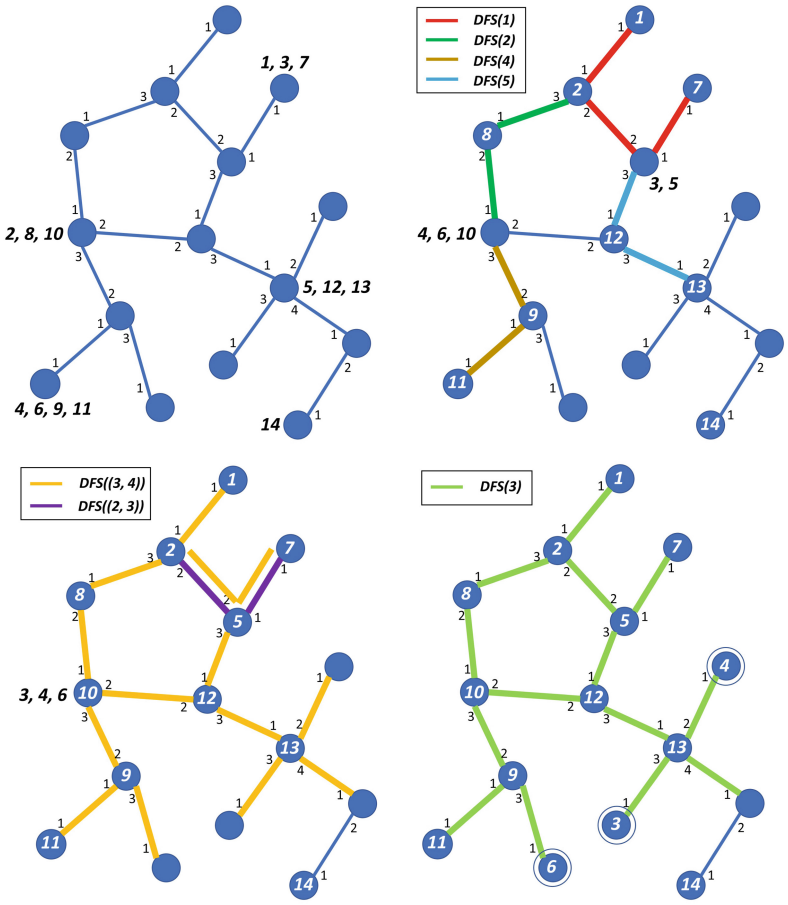

Fig. 1. An illustration of the two stages in a pass of Algorithm 1 for $k = 14$ robots in an 15-node graph G. (**top left**) shows C_{init} with one or more robots at 5 nodes of G; the rest of the nodes of G are empty. (**top right**) shows the configuration after Stage 1 finishes for $DFS(.)$ started by 4 nodes with multiple robots on them; the respective DFS trees formed are shown through colored edges (the same colored edges belong to the same DFS tree). A single robot (14) at a node settles there. (**bottom left**) shows the configuration after Stage 2 finishes for $DFS((.,.))$ started by two nodes with more than one robot (see top right) on them when Stage 1 finishes. The robots 3,4,6 are collected at the node of G where robot 10 is settled since $DFS((3,4))$ started from there has higher lexico-priority than $DFS((2,3))$ started from the node of G where 5 is settled. (**bottom right**) shows the configuration after Stage 1 of the next pass in which all k robot settle on k different nodes of G. There is only one DFS tree $DFS(3)$ started from the node of G (where 10 is settled and all robots are collected in Stage 2) that traverses G until all $3, 4, 6$ are settled reaching the empty nodes of G. The nodes where they are settled are shown inside a circle. (Color figure online)

4.2 Detailed Description of the Algorithm

The pseudocode of the algorithm is given in Algorithm 1. The variables used by each robot are described in Table 2. We now describe the two stages of the algorithm; Fig. 1 illustrates the working principle of the stages.

Stage 1. We first introduce some terminology. A settled/unsettled robot i is one for which $i.settled = 1/0$. For brevity, we say a node is settled if it has a settled robot. At the start of stage 1, there may be multiple (≥ 1) unsettled robots at some of the nodes. Let U^{s1}/U^{e1}/U^{e2} be the set of unsettled robots at a node at the start of stage 1/end of stage 1/end of stage 2. In general, we define a U-set to be the (non-empty) set of unsettled robots at a node. Let the lowest robot ID among U^{s1} at a node be U^{s1}_{min}. We use r to denote a settled robot.

In stage 1, the unsettled robots at a node begin $DFS(|U^{s1}|)$, following the lowest ID ($= U^{s1}_{min}$) robot among them. Each instance of the DFS algorithm, begun concurrently by different U^{s1}-sets from different nodes, induces a DFS tree in which the settled nodes have robots with the same $treelabel$, which is equal to the corresponding U^{s1}_{min}. During this DFS traversal, the robots visit nodes, at each of which there are four possibilities.

The node may be free, or may have a settled robot r, where $r.treelabel$ is less than, equals, or is greater than $x.ID$, where x is the visiting robot with the lowest ID. The second and fourth possibilities indicate that two DFS trees, corresponding to different $treelabel$s meet. As each robot is allowed only $O(\log n)$ bits memory, it can track the variables for only one DFS tree. We deal with these possibilities as described below.

1. If the node is free (line 6), the logic of $DFS(k)$ described in Sect. 3 is followed. Specifically, the highest ID robot from the visiting robots (call it r) settles, and sets $r.settled$ to 1 and $r.treelabel$ to $x.ID$. Robot x continues its DFS, after setting $r.parent$, $r.child$ and $r.phase$ for its own DFS as per the logic of $DFS(k)$ described in Sect. 3; and other visiting robots follow x.
2. If $r.treelabel < x.ID$ (line 11), all visiting robots stop at this node and discontinue growing their DFS tree.
3. If $r.treelabel = x.ID$ (line 13), robot x's traversal is part of the same DFS tree as that of robot r. Robot x continues its DFS traversal and takes along with it all unsettled (including stopped) robots from this node, after updating $r.child$ if needed as per the logic of $DFS(k)$ described in Sect. 3.
4. If $r.treelabel > x.ID$ (line 16), robot x continues growing its DFS tree and takes along all unsettled robots from this node with it. To continue growing its DFS tree, x overwrites robot r's variables set for r's old DFS tree by including this node and r in its own DFS tree. Specifically, $r.treelabel \leftarrow x.ID$, $r.parent$ is set to the port from which x entered this node, and $r.child$ is set as per the logic described for $DFS(k)$ in Sect. 3.

Note that if the robots stop at a node where $r.treelabel < x.ID$, they will start moving again if a robot x' arrives such that $x'.ID \leq r.treelabel$. At the end of stage 1, either all the robots from any U^{s1} are settled or some subset of them are stopped at some node where $r.treelabel < U^{s1}_{min}$.

Algorithm 1: Algorithm *Graph_Disperse(k)* to solve DISPERSION.

if i *is alone at node* **then**
  $i.settled \leftarrow 1$; do not set $i.treelabel$
for $pass = 1, \log\ k$ **do**
  Stage 1 (Graph_DFS: for group dispersion of unsettled robots)
  for $round = 0, \min(4m - 2n + 2, k\Delta)$ **do**
    if *visited node is free* **then**
      highest ID robot r settles; $r.treelabel \leftarrow x$.ID, where x is robot with lowest ID
      x continues its DFS after r sets its $parent$, $child$ for DFS of x
      other visitors follow x
    else if *visited node has a settled robot* r **then**
      if $r.treelabel < x.ID$ *for visitors* x **then**
        all visiting robots: stop until ordered to move
      else if $r.treelabel \leq y.ID$ *for visitors* y *and* $r.treelabel = x.ID$ *for some visitor* x **then**
        x continues its DFS after r updates $child$ if needed
        all other unsettled robots follow x
      else if *visitor* $x(x \neq r)$ *has lowest ID and lower than* $r.treelabel$ **then**
        $r.treelabel \leftarrow x.ID$
        x continues its DFS after r sets its $parent$, $child$ for DFS of x
        all other unsettled robots follow x
  All settled robots: reset $parent$, $child$
  Stage 2 (Connected_Component_DFS_Traversal: for gathering unsettled robots)
  All robots: $mult \leftarrow$ count of local robots
  if i *has the lowest ID among unsettled robots at its node* **then**
    $i.home \leftarrow r.ID$, $r.treelabel \leftarrow i.ID$, where r is the settled robot at that node
    i initiates DFS traversal of connected component of nodes with settled robots
  for $round = 0, \min(4m - 2n + 2, 2k\Delta)$ **do**
    if *visited node is free* **then**
      ignore the node; all visitors backtrack, i.e., retrace their step
    else if *visited node has a settled robot* r **then**
      if *lexico-priority of* r *is highest and greater than that of all visitors* **then**
        all visiting robots: stop until ordered to move
      else if *lexico-priority of* r *is highest but equal to that of some visitor* x **then**
        x continues its DFS traversal after r updates $child$ if needed (until $x.home = r.ID$ and all ports at the node where r is settled are explored)
        all other unsettled robots: follow x if $x.home \neq r.ID$
      else if *visitor* $x(x \neq r)$ *has highest lexico-priority and higher than that of* r **then**
        $r.treelabel \leftarrow x.ID$, $r.mult \leftarrow x.mult$
        x continues its DFS traversal after r sets $parent$, $child$ for DFS of x
        all other unsettled robots follow x
  reset $parent$, $child$, $treelabel$, $mult$, $home$

Lemma 2. *For any U^{s1}-set, at the end of stage 1, either (i) all the robots in U^{s1} are settled or (ii) the unsettled robots among U^{s1} are present all together along with robot with ID U^{s1}_{min} (and possibly along with other robots outside of U^{s1}) at a* single node *with a settled robot r having $r.treelabel < U^{s1}_{min}$.*

Proof. The DFS traversal of the graph can complete in $4m - 2n + 2$ steps as each tree edge gets traversed twice, and each back edge, i.e., non-tree edge of the DFS tree, gets traversed 4 times (twice in the forward direction and twice in the backward direction) if the conditions in lines (6), (13), or (16) hold. The DFS traversal of the graph required to settle k robots and hence discover k new nodes, can also complete in $k\Delta$ steps as a node may be visited multiple times (at most its degree which is at most Δ times). As $k \geq |U^{s1}|$, possibility (i) is evident.

In the DFS traversal, if condition in line (11) holds, the unsettled robots remaining in U^{s1}, including that with ID U^{s1}_{min}, stop together at a node with a settled robot r' such that $r'.treelabel < U^{s1}_{min}$. They may move again together (lines (15) or (19)) if visited by a robot with ID U'_{min} equal to or lower than $r'.treelabel$ (lines (13) or (16)), and may either get settled (possibility (i)), or stop (the unsettled ones together) at another node with a settled robot r'' such that $r''.treelabel < U'_{min}$. This may happen up to $k - 1$ times. However, the remaining unsettled robots from U^{s1} never get separated from each other. If the robot with ID U^{s1}_{min} is settled at the end of stage 1, so are all the others in U^{s1}. If U^{s1}_{min} robot is not settled at the end of stage 1, the remaining unsettled robots from U^{s1} have always moved and stopped along with U^{s1}_{min} robot. This is because, if the robot with ID U^{s1}_{min} stops at a node with settled robot r''' (line 12), $r'''.treelabel < U^{s1}_{min}$ and hence $r'''.treelabel$ is also less than the IDs of the remaining unsettled robots from U^{s1}. If the stopped robot with ID U^{s1}_{min} begins to move (line 15 or 19), so do the other stopped (unsettled) robots from U^{s1} because they are at the same node as the robot with ID U^{s1}_{min}. Hence, (ii) follows. □

Let us introduce some more terminology. Let $\mathcal{U}^{s1}$ be the set of all U^{s1}. Let $\mathcal{U}^{s1}_{min}$ be $\min_{U^{s1} \in \mathcal{U}^{s1}}(U^{s1}_{min})$. The set of robots in that U^{s1} having $U^{s1}_{min} = \mathcal{U}^{s1}_{min}$ are dispersed at the end of stage 1 because the DFS traversal of the robots in that U^{s1} is not stopped at any node by a settled robot having a lower $treelabel$ than that U^{s1}_{min}. Let u^{s1}_p, $u^{e1}_p = u^{s2}_p$, and u^{e2}_p denote the number of nodes with unsettled robots at the start of stage 1, at the end of stage 1(or at the start of stage 2), and at the end of stage 2 respectively, all for a pass p of the algorithm. Thus, u^{s1}_p (= $|\mathcal{U}^{s1}_p|$) is the number of U-sets at the start of stage 1 of pass p. Analogously, for $u^{e1}_p = u^{s2}_p$, and u^{e2}_p.

We now have the following corollary to Lemma 2.

Corollary 1. $u^{e1}_p \leq u^{s1}_p - 1$.

In stage 1, each set of unsettled robots U^{s1} induces a partial DFS tree, where the $treelabel$ of settled robots is U^{s1}_{min}. This identifies a sub-component $SC_{U^{s1}_{min}}$. Note that some subset of U^{s1} may be stopped at a node outside $SC_{U^{s1}_{min}}$, where the $treelabel < U^{s1}_{min}$.

Definition 2. *A sub-component SC_α is the set of all settled nodes having $treelabel = \alpha$. $\mathcal{SC}$ is used to denote the set of all SCs at the end of stage 1.*

Theorem 2. *There is a one-to-one mapping from the set of sub-components $\mathcal{SC}$ to the set of unsettled robots $\mathcal{U}^{s1}$. The mapping is given by: $SC_\alpha \mapsto U^{s1}$, where $\alpha = U^{s1}_{min}$.*

Proof. From Definition 2, each SC_α corresponds to a $treelabel = \alpha$. The $treelabel$ is set to the lowest ID among visiting robots, and this corresponds to a unique set of unsettled robots U^{s1} whose minimum ID robot has ID α, i.e., $U^{s1}_{min} = \alpha$. □

Lemma 3. *Sub-component SC_α is a connected sub-component of settled nodes, i.e., for any $a, b \in SC_\alpha$, there exists a path (a, b) in G such that each node on the path has a settled robot.*

Proof. For any nodes a and b in SC_α, the robot with ID U_{min} $(= \alpha)$ has visited a and b. Thus there is some path from a to b in G that it has traversed. On that path, if there was a free node, a remaining unsettled robot from U (there is at least the robot with ID U_{min} that is unsettled) would have settled there. Thus there cannot exist a free node on that path and the lemma follows. □

Within a sub-component, there may be stopped robots belonging to one or more different sets U^{s1} (having a higher U^{s1}_{min} than the $treelabel$ at the node where they stop). There may be multiple sub-components that are adjacent in the sense that they are separated by a common edge. Together, these sub-components form a connected component of settled nodes.

Definition 3. *A connected component of settled nodes (CCSN) is a set of settled nodes such that for any $a, b \in CCSN$, there exists a path (a, b) in G with each node on the path having a settled robot.*

Lemma 4. *If not all the robots of U^{s1} are settled by the end of stage 1, then $SC_{U^{s1}_{min}}$ is part of a CCSN containing nodes from at least two sub-components.*

Proof. Let the unsettled robots in U^{s1} begin from node a. The unsettled robots of U^{s1} stopped (line 12), and possibly moved again (line 15 or 19) only to be stopped again (line 12), c times, where $|\mathcal{U}^{s1}| > c \geq 1$.

Consider the first time the robots arriving along edge (u, v) were stopped at some node v. $U^{s1}_{min} > r.treelabel$, where robot r is settled at v. Henceforth till the end of stage 1, $r.treelabel$ is monotonically non-increasing, i.e., it may only decrease if a visitor arrives with a lower ID (line 16). The path traced from a to u must have all settled nodes, each belonging to possibly more than one sub-component, i.e., possibly in addition to $SC_{U^{s1}_{min}}$, at the end of stage 1, which together form one or more adjacent sub-components. In any case, these sub-components are necessarily adjacent to the sub-component SC_α, where $\alpha = r.treelabel$. Thus, at least two sub-components including $SC_{U^{s1}_{min}}$ and SC_α are (possibly transitively) adjacent and form part of a CCSN.

Extending this reasoning to each of the c times the robots stopped, it follows that there are at least $c + 1$ sub-components in the resulting CCSN.(Additionally, (1) unsettled robots from the sub-component that stopped the unsettled robots of U^{s1} for the c-th time may be (transitively) stopped by robots in yet other sub-components, (2) other groups of unsettled robots may (transitively or independently) be stopped at

nodes in the above identified sub-components, (3) other sub-components corresponding to even lower $treelabel$s may join the already identified sub-components, (4) other sub-components may have a node which is adjacent to one of the nodes in an above-identified sub-component. This only results in more sub-components, each having distinct $treelabel$s (Definition 2) and corresponding to as many distinct U-sets (Theorem 2), being adjacent in the resulting CCSN.) □

Theorem 3. *For any U^{s1} at a, its unsettled robots (if any) belong to a single U^{e1} at b, where a and b belong to the same connected component of settled nodes (CCSN).*

Proof. From Lemma 2, it follows that the unsettled robots from U^{s1} (at a) end up at a single node b in the set U^{e1}. It follows that there must exist a path from a to b that these unsettled robots traversed. On this path, if there was a free node, a robot that belongs to U^{s1} and U^{e1} would have settled. Thus, there cannot exist such a free node. It follows that a and b belong to the same CCSN. □

Using the reasoning of Lemma 2 and Corollary 1, if there are s sub-components within a CCSN, there may be stopped (unsettled) robots at at most $s-1$ nodes. In stage 2, all such unsettled robots within a CCSN are collected at a single node within that component.

Stage 2. Stage 2 begins with each robot setting variable $mult$ to the count of robots at its node. The lowest ID unsettled robot x at each node (having $mult > 1$) concurrently initiates a DFS traversal of the CCSN after setting $x.home$ to the ID of the settled robot r and setting the $r.treelabel$ of the settled robot to its ID, $x.ID$. The DFS traversal is initiated by a single unsettled robot at a node rather than all unsettled robots at a node.

In the DFS traversal of the CCSN, there are four possibilities, akin to those in stage 1. If a visited node is free (line 27), the robot ignores that node and backtracks. This is because neither the free node nor any paths via the free node need to be explored to complete a DFS traversal of the CCSN.

If a visited node has a settled robot, the visiting robots may need to stop for two reasons. (i) Only the highest "priority" unsettled robot should be allowed to complete its DFS traversal while collecting all other unsettled robots. Other concurrently initiated DFS traversals for gathering unsettled robots should be stopped so that only one traversal for gathering succeeds. (ii) With the limited memory of $O(\log n)$ at each robot, only one DFS traversal can be enabled at each settled robot r in its $r.treelabel$, $r.parent$, and $r.child$. That is, the settled robot can record in its data structures, only the details for one DFS tree that is induced by one DFS traversal. The decision to continue the DFS or stop is based, not by comparing $treelabel$ of the settled robot with the visiting robot ID, but by using a lexico-priority, defined next.

Definition 4. *The lexico-priority is defined by a tuple, $(mult,\ treelabel/ID)$. A higher value of $mult$ is a higher priority; if $mult$ is the same, a lower value of $treelabel$ or ID has the higher priority.*

The lexico-priority of a settled robot r that is visited, $(r.mult,\ r.treelabel)$, is compared with $(x.mult, x.ID)$ of the visiting robots x. The lexico-priority is a total order. There are three possibilities, as shown in lines (30), (32), and (35).

- (line 30): Lexico-priority of $r >$ lexico-priority of all visitors: All visiting robots stop (until ordered later to move) because they have a lower lexico-priority than r. The DFS traversal of the unsettled robot x' corresponding to $x'.ID = r.treelabel$ kills the DFS traversal of the visitors.
- (line 32): The visiting robot x having the highest lexico-priority among the visiting robots, and having the same lexico-priority as r continues the DFS traversal because it is part of the same DFS tree as r. r updates $r.child$ if needed as per the logic of $DFS(k)$ described in Sect. 3. This DFS search of x continues unless x is back at its home node from where it began its search and all ports at the home node have been explored. As x continues its DFS traversal, it takes along with it all unsettled robots at r.
- (line 35): The visiting robot x having the highest lexico-priority that is also higher than that of r overrides the $treelabel$ and $mult$ of r. It kills the DFS traversal and corresponding DFS tree that r is currently storing the data structures for. Robot x includes r in its own DFS traversal by setting $r.treelabel \leftarrow x.ID$, $r.mult \leftarrow x.mult$, and $r.parent$ to the port from which x entered this node; $r.child$ is set as per the logic of $DFS(k)$ described in Sect. 3. Robot x continues its DFS traversal and all other unsettled robots follow it.

The reason we use the lexico-priority defined on the tuple rather than on just the $treelabel/ID$ is that the sub-component with the lowest $treelabel$ may have no unsettled robots, but yet some node(s) in it are adjacent to those in other sub-components, thus being part of the same CCSN. The nodes in the sub-component with the lowest $treelabel$ would then stop other traversing robots originating from other sub-components, but no robot from that sub-component would initiate the DFS traversal.

Lemma 5. *Within a connected component of settled nodes (CCSN), let x be the unsettled robot with the highest lexico-priority at the start of Stage 2.*

1. *x returns to its home node from where it begins the DFS traversal of the component, at the end of Stage 2.*
2. *All settled nodes in the connected component have the same lexico-priority as x at the end of Stage 2.*

Proof. (Part 1): Robot x encounters case in line (35) for the first visit to each node in its CCSN and includes that node in its own DFS traversal, and on subsequent visits to that node, encounters the case in line (32) and continues its DFS traversal. Within $\min(4m - 2n + 2, 2k\Delta)$ steps, it can complete its DFS traversal of the CCSN and return to its home node. This is because it can visit all the nodes of the graph within $4m - 2n + 2$ steps. The robot can also visit the at most k settled nodes in $2k\Delta$ steps; $k\Delta$ steps may be required in the worst case to visit the k settled nodes in its CCSN and another at most $k\Delta$ steps to backtrack from adjacent visited nodes that are free.

(Part 2): When x visits a node with a settled robot r for the first time (line 35), the lexico-priority of r is changed to that of x (line 36). Henceforth, if other unsettled robots y visit r, r will not change its lexico-priority (line 30) because its lexico-priority is now highest. □

Analogous to stage 1, unsettled robots beginning from different nodes may move and then stop (on reaching a higher lexico-priority lp node), and then resume movement again (when visited by a robot with lexico-priority lp or higher). This may happen up to $s-1$ times, where s is the number of sub-components in the CCSN. We show that, despite the concurrently initiated DFS traversals and these concurrent movements of unsettled robots, they all gather at the end of stage 2, at the home node of the unsettled robot having the highest lexico-priority (in the CCSN) at the start of stage 2.

Lemma 6. *Within a connected component of settled nodes (CCSN), let x be the unsettled robot with the highest lexico-priority at the start of Stage 2. All the unsettled robots in the component gather at the home node of x at the end of Stage 2.*

Proof. Let y be any unsettled robot at the start of the stage. At time step t, let y be at a node denoted by $v(t)$. Let τ be the earliest time step at which y is at a node with the highest lexico-priority that it encounters in Stage 2. We have the following cases.

1. lexico-priority(settled robot at $v(\tau)$) $<$ lexico-priority(x): We have a contradiction because at $t = \min(4m - 2n + 2, 2k\Delta)$, settled robots at all nodes have lexico-priority that of x, which is highest.
2. lexico-priority(settled robot at $v(\tau)$) $>$ lexico-priority(x): This contradicts the definition of x.
3. lexico-priority(settled robot at $v(\tau)$) $=$ lexico-priority(x).
   (a) $v(\tau) = x.home$: Robot y will not move from $x.home$ (line 32) and the lemma stands proved.
   (b) $v(\tau) \neq x.home$: y ends up at another node with lexico-priority that of x at time step τ. It will not move from node $v(\tau)$ unless robot x visits $v(\tau)$ at or after τ, in which case y will accompany x to $x.home$ and the lemma stands proved.

   We need to analyze the possibility that x does not visit $v(\tau)$ at or after τ. That is, the last visit by x to $v(\tau)$ was before τ. By definition of τ, lexico-priority(settled robot at $v(\tau-1)$) $<$ lexico-priority(settled robot at $v(\tau)$) (= lexico-priority of x in this case). By Lemma 5, x is yet to visit $v(\tau-1)$, so the first visit of x to $v(\tau-1)$ is after $\tau-1$. As $v(\tau-1)$ and $v(\tau)$ are neighbors and x is doing a DFS, x will visit $v(\tau)$ at or after $\tau+1$. This contradicts that the last visit by x to $v(\tau)$ was before τ and therefore rules out the possibility that x does not visit $v(\tau)$ at or after τ.

□

4.3 Correctness of the Algorithm

Having proved the properties of stage 1 and stage 2, we now prove the correctness of the algorithm.

Lemma 7. $u_p^{e2} = u_{p+1}^{s1} \leq \frac{1}{2} \cdot u_p^{s1}$

Proof. From Lemma 2, for any U^{s1} at the end of stage 1, (i) a set of unsettled robots U^{s1} is fully dispersed, or (ii) a subset of U^{s1} of unsettled robots is stopped and present together at at most one node with a settled robot r such that $r.treelabel < U_{min}^{s1}$.

In case (i), there are two possibilities. (i.a) There is no group of unsettled robots stopped at nodes in the CCSN where the robots of U have settled. In this case, this U^{s1}-set does not have its robots in any U^{e1}-set. (i.b) $z(\geq 1)$ groups of unsettled robots are stopped at nodes in the CCSN where the robots of U have settled. These groups correspond to at least $z+1$ unique U-sets and at least $z+1$ sub-components that form a CCSN (by using reasoning similar to that in the proof of Lemma 4). In case (ii), at least two sub-components, each having distinct $treelabel$s and corresponding to as many distinct U-sets (Theorem 2), are adjacent in the CCSN (Lemma 4).

From Lemma 2, we also have that any U^{s1}-set cannot have unsettled robots in more than one U^{e1}. Each robot in each U^{s1}-set in the CCSN, that remains unsettled at the end of stage 1, belongs to some U^{e1}-set that also belongs to the *same* CCSN (Theorem 3). From Lemma 6 for stage 2, all the unsettled robots in these U^{e1}-sets in the CCSN, are gathered at one node in that CCSN. Thus, each unsettled robot from each U^{s1}-set in the same CCSN is collected at a single node as a U^{e2}-set in the same CCSN. Thus, in cases (i-b) and (ii) above, *two or more* sub-components, each corresponding to a distinct $treelabel$ and a distinct U^{s1}-set (Theorem 2), combine into a single CCSN (Lemma 4) and in stage 2, there is a single node with unsettled robots from all the U^{s1}-sets belonging to the same CCSN, i.e., a single U^{e2}-set, or a single U^{s1}-set for the next round. Note that each sub-component SC_α is a connected sub-component (Lemma 3) and hence belongs to the same CCSN; thus when sub-components merge, i.e., their corresponding U^{s1}-sets merge, and we have a single U^{e2}-set in the CCSN, there is no double-counting of the same SC_α and of its corresponding U^{s1}-set in different CCSNs. Thus, u_p^{e2} (= u_{p+1}^{s1}), the number of U-sets after stage 2, is $\leq \frac{1}{2} \cdot u_p^{s1}$, where u_p^{s1} is the number of U-sets before stage 1. □

Theorem 4. DISPERSION *is solved in* $\log k$ *passes in Algorithm 1.*

Proof. $u_1^{s1} \leq k/2$. From Lemma 7, it will take at most $\log k - 1$ passes for there to be a single U-set. In the first stage of the $\log k$-th pass, there will be a single U-set. By Lemma 2, case (i) holds and all robots in the U-set get settled. (Case (ii) will not hold because there is no node with a $treelabel < U_{min}$ as all $treelabel$s of settled nodes are reset to $\top$ (the highest value) at the end of stage 2 of the previous pass and all singleton robots before the first pass settle with $treelabel = \top$ (line 2)). Thus, DISPERSION will be achieved by the end of stage 1 of pass $\log k$. □

Note that the DFS traversal of stage 2 is independent of the DFS traversal of stage 1 within a pass (but the $treelabel$s are not erased), and the DFS traversal of stage 1 of the next pass is independent of the DFS traversal of stage 2 of the current pass.

Proof of Theorem 1 : Theorem 4 proved that DISPERSION is achieved by Algorithm 1. The time complexity is evident due to the two loops of $O(\min(4m-2n+2, 2\Delta k))$ for the two stages nested within the outer loop of $O(\log k)$ passes. The space complexity is evident from the size of the variables: $treelabel$ ($\log k$ bits), $parent$ ($\log \Delta$ bits), $child$ ($\log \Delta$ bits), $settled$ (1 bit), $mult$ ($\log k$ bits), $home$ ($\log k$ bits), $pass$ ($\log \log k$ bits), $round$ ($O(\log n)$ bits to maintain the value $O(\min(m, k\Delta)$ for each pass). □

We have the following corollary to Theorem 1 when $\Delta \leq k$.

Corollary 2. *Given $k \leq n$ robots in an n-node arbitrary graph G with maximum degree $\Delta \leq k$, Algorithm $Graph\_Disperse(k)$ solves* DISPERSION *in $O(\min(m, k^2) \cdot \log k)$ rounds with $O(\log n)$ bits at each robot.*

We also have the following corollary to Theorem 1 when $\Delta = O(1)$.

Corollary 3. *Given $k \leq n$ robots in an n-node arbitrary graph G with maximum degree $\Delta = O(1)$, algorithm $Graph\_Disperse(k)$ solves* DISPERSION *in $O(\min(m, k) \cdot \log k)$ rounds with $O(\log n)$ bits at each robot.*

5 Concluding Remarks

We have presented a deterministic algorithm for solving DISPERSION of $k \leq n$ robots on n-node arbitrary graphs. Our result significantly improves the $O(mk)$ runtime of the best previously known algorithm with logarithmic memory at each robot [16] to $O(\min(m, k\Delta) \cdot \log k)$ with logarithmic memory at each robot. For future work, it will be interesting to solve DISPERSION on arbitrary graphs with time $O(k)$ or improve the existing time lower bound of $\Omega(k)$ to $\Omega(\min(m, k\Delta))$. Another interesting direction is to remove the $\log k$ factor from the time bound in Theorem 1. Furthermore, it will be interesting to achieve Theorem 1 without each robot knowing parameters m, Δ, and k. Finally, another interesting direction will be to extend our algorithms to solve DISPERSION in semi-synchronous and asynchronous settings.

References

1. Augustine, J., Moses Jr., W.K.: Dispersion of mobile robots: a study of memory-time trade-offs. CoRR, abs/1707.05629, [v4] (2018). (A preliminary version in ICDCN 2018)
2. Bampas, E., Gąsieniec, L., Hanusse, N., Ilcinkas, D., Klasing, R., Kosowski, A.: Euler tour lock-in problem in the rotor-router model. In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 423–435. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04355-0_44
3. Barriere, L., Flocchini, P., Mesa-Barrameda, E., Santoro, N.: Uniform scattering of autonomous mobile robots in a grid. In: IPDPS, pp. 1–8 (2009)
4. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. ACM Trans. Algorithms **4**(4), 42:1–42:18 (2008)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
6. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. **7**(2), 279–301 (1989)
7. Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: Autonomous mobile robots with lights. Theor. Comput. Sci. **609**, 171–184 (2016)
8. Dereniowski, D., Disser, Y., Kosowski, A., Pajak, D., Uznański, P.: Fast collaborative graph exploration. Inf. Comput. **243**(C), 37–49 (2015)
9. Elor, Y., Bruckstein, A.M.: Uniform multi-agent deployment on a ring. Theor. Comput. Sci. **412**(8–10), 783–795 (2011)
10. Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by oblivious mobile robots. Synth. Lect. Distrib. Comput. Theory **3**(2), 1–185 (2012)

11. Flocchini, P., Prencipe, G., Santoro, N.: Distributed Computing by Mobile Entities. Theoretical Computer Science and General Issues, vol. 1. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11072-7
12. Fraigniaud, P., Gasieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks **48**(3), 166–177 (2006)
13. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. **345**(2–3), 331–344 (2005)
14. Hsiang, T.-R., Arkin, E.M., Bender, M.A., Fekete, S., Mitchell, J.S.B.: Online dispersion algorithms for swarms of robots. In: SoCG, pp. 382–383 (2003)
15. Hsiang, T.-R., Arkin, E.M., Bender, M.A., Fekete, S.P., Mitchell, J.S.B.: Algorithms for rapidly dispersing robot swarms in unknown environments. In: Boissonnat, J.-D., Burdick, J., Goldberg, K., Hutchinson, S. (eds.) Algorithmic Foundations of Robotics V. STAR, vol. 7, pp. 77–93. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-45058-0_6
16. Kshemkalyani, A.D., Ali, F.: Efficient dispersion of mobile robots on graphs. In: ICDCN, pp., 218–227 (2019)
17. Menc, A., Pajak, D., Uznanski, P.: Time and space optimality of rotor-router graph exploration. Inf. Process. Lett. **127**, 17–20 (2017)
18. Molla, A.R., Moses, W.K.: Dispersion of mobile robots: the power of randomness. In: Gopal, T.V., Watada, J. (eds.) TAMC 2019. LNCS, vol. 11436, pp. 481–500. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14812-6_30
19. Poudel, P., Sharma, G.: Time-optimal uniform scattering in a grid. In: ICDCN, pp. 228–237 (2019)
20. Shibata, M., Mega, T., Ooshita, F., Kakugawa, H., Masuzawa, T.: Uniform deployment of mobile agents in asynchronous rings. In: PODC, pp. 415–424 (2016)
21. Subramanian, R., Scherson, I.D.: An analysis of diffusive load-balancing. In: SPAA, pp. 220–225 (1994)