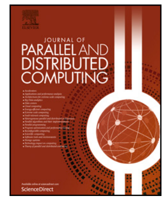
Journal of Parallel and Distributed Computing 204 (2025) 105139

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc



Dispersion of mobile robots on directed anonymous graphs ☆

Giuseppe F. Italiano [a], Debasish Pattanayak [b,*], Gokarna Sharma [c]

[a] LUISS University, Rome, Italy
[b] University of Ottawa, Ottawa, Canada
[c] Kent State University, Kent, OH, USA

ARTICLE INFO

Keywords:
Distributed algorithms
Multi-agent systems
Autonomous mobile robots
Local and 1-hop communication
Directed graphs
Deficiency
Dispersion
Time and memory complexity

ABSTRACT

Given any arbitrary initial configuration of $k \leq n$ robots positioned on the nodes of an n-node anonymous graph, the problem of dispersion is to autonomously reposition the robots such that each node will contain at most one robot. This problem gained significant interest due to its resemblance with several fundamental problems such as exploration, scattering, load balancing, relocation of electric cars to charging stations, etc. The objective is to solve dispersion simultaneously minimizing (or providing trade-off between) time and memory requirement at each robot. The literature mainly dealt with dispersion on undirected anonymous graphs. In this paper, we initiate the study of dispersion on directed anonymous graphs. We first show that it may not always be possible to solve dispersion when the directed graph is not strongly connected. We then establish some lower bounds on both time and memory requirement at each robot for solving dispersion on a strongly connected directed graph. Finally, we provide three deterministic algorithms solving dispersion on any strongly connected directed graph. Let D be the graph diameter, Δ_{out} be its maximum out-degree, and d be the deficiency (the minimum number of edges needed to add to the graph to make it Eulerian). The first algorithm solves dispersion in $O(d \cdot k^2)$ time with $O(k \cdot \log(k + \Delta_{out}))$ bits at each robot. The second algorithm solves dispersion in $O(k^2 \cdot \Delta_{out})$ time with $O(\log(k + \Delta_{out}))$ bits at each robot. The third algorithm solves dispersion in $O(k \cdot D)$ time with $O(k \cdot \log(k + \Delta_{out}))$ bits at each robot, provided that robots in the 1-hop neighborhood can communicate. All three algorithms extend to handle crash faults.

1. Introduction

The dispersion of autonomous mobile robots in a region is a problem of significant interest in distributed robotics [2,3]. Recently, this problem has been formulated by Augustine and Moses Jr. [4] in the context of graphs 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, e.g., in relocating self-driving electric cars (robots) to recharging stations (nodes), assuming that the cars have smart devices to communicate with each other to find a free/empty charging station. This problem is also important due to its relationship to many other multi-robot coordination problems including exploration, scattering, load balancing, and self-deployment [4–7].

The objective in DISPERSION is to simultaneously minimize (or provide trade-off between) two fundamental performance metrics, (i) *time* and (ii) *memory* at each robot, to successfully solve the problem. The literature studied DISPERSION mainly on undirected graphs [4–10]. The following question naturally arises: *Is it possible to solve DISPERSION on directed anonymous graphs?* The main motivation behind posing this problem is that the approaches for undirected graphs may not extend to solve dispersion on directed graphs. The existing approaches for undirected graphs rely substantially on visiting a graph edge in both directions. The main challenge for directed graphs is the direction on edges that restricts movement in only one direction. So any technique for directed graphs should take a critical care on this aspect. In this paper, we study DISPERSION for the first time on directed graphs.

Contributions. We consider an *anonymous* port-labeled directed graph $G = (V, E)$, $|V| = n$, $|E| = m$, where nodes have no IDs and hence are indistinguishable from each other, but the outgoing ports (leading to outgoing edges and neighbors) at each node are distinguishable. The (outgoing) ports of any node $v \in G$ with out-degree δ_{out}^v have unique

☆ A preliminary version of this article appears in the Proceedings of SIROCCO'22 [1].

* Corresponding author.
E-mail addresses: gitaliano@luiss.it (G.F. Italiano), dpattana@uottawa.ca (D. Pattanayak), gsharma2@kent.edu (G. Sharma).

https://doi.org/10.1016/j.jpdc.2025.105139
Received 3 August 2023; Received in revised form 13 June 2025; Accepted 15 June 2025

Available online 20 June 2025
0743-7315/© 2025 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Table 1
Summary of Contributions.

Result	Crash	Memory (bits)	Time (rounds)	Remark
Lower Bound	-	$\Omega(\log k)$	$\Omega(k)$ for $k \le n$	Section 3
Lower Bound	-	$\Omega(\log k)$	$\Omega(D^2)$ for $k = n$	Section 3
Lower Bound (DFS-based)	-	$\Omega(\log k)$	$\Omega(k \cdot \min\{k, d\})$	Section 3
Algorithm (Local Model)	**Crash**	**Memory (bits)**	**Time (rounds)**	**Remark**
DFS-based	No	$O(k \log(k + \Delta_{out}))$	$O(d \cdot k^2)$	Section 4
DFS-based	Yes	$O(k \log(k + \Delta_{out}))$	$O(f \cdot d \cdot k^2)$	Section 7
DFS-based	No	$\Theta(\log(k + \Delta_{out}))$	$O(k^2 \cdot \Delta_{out})$	Section 5
DFS-based	Yes	$\Theta(\log(k + \Delta_{out}))$	$O(f \cdot k^2 \cdot \Delta_{out})$	Section 7
Algorithm (1-hop Model)	**Crash**	**Memory (bits)**	**Time (rounds)**	**Remark**
BFS-based	No	$O(k \log(k + \Delta_{out}))$	$O(k \cdot D)$	Section 6
BFS-based	Yes	$O(k \log(k + \Delta_{out}))$	$O(k \cdot D)$	Section 7

labels in $[1, \delta^v_{out}]$. The graph G is assumed to have deficiency d, which denotes the minimum number of edges needed to add to G to make it Eulerian. A directed graph is Eulerian if each vertex has the equal number of incoming and outgoing edges. We consider $k \le n$ robots on graph nodes, initially positioned arbitrarily (at least one node with multiple robots positioned on it; otherwise, the problem is trivially solved). Each robot has a unique $O(\log k)$-bit ID from $[1, k^{O(1)}]$. The robots have memory, but the graph nodes do not. The setting is *synchronous* – all robots become active and perform their operations simultaneously in synchronized rounds – and time is measured in rounds. Following the literature [4–6,8–10], we consider *local* and *1-hop* communication models. In the local model, only robots co-located at a graph node can communicate. In the 1-hop model, a robot may also communicate with robots positioned on its 1-hop neighbors (in addition to the co-located robots). The robots move respecting the direction of the edges. We model robot *crashes* such that a robot may stop communicating with other robots at any round and if it does so then it will continue to not communicating with other robots in each round thereafter as if it is vanished from the system.

First, we establish one impossibility and some lower bound results (**Section 3**). We show that it may not always possible to solve Dispersion on a directed graph that is not strongly connected; a directed graph is *strongly connected* if there is a path in each direction between each pair of vertices. For strongly connected directed graphs, (i) we observe a time lower bound of $\Omega(k)$ for any $k \le n$, (ii) we prove a time lower bound of $\Omega(D^2)$ for $k = n$, (iii) we prove a time lower bound of $\Omega(k \cdot \min\{k, d\})$ for any DFS traversal based algorithm, and (iv) we prove a memory lower bound of $\Omega(\log k)$ bits per robot. All these lower bounds are deterministic.

Second, as our main contribution, we provide three deterministic algorithms solving Dispersion on any strongly connected directed graph. Specifically, we prove the following theorem.

Theorem 1. *Consider $k \le n$ robots positioned initially arbitrarily on an n-node strongly connected directed anonymous graph with diameter D, outdegree Δ_{out}, and deficiency d, Dispersion can be solved in*

a. *$O(d \cdot k^2)$ time with $O(k \cdot \log(k + \Delta_{out}))$ bits at each robot under the local model with a DFS based algorithm. For $f \le k$ robot crashes, the time bound becomes $O(f \cdot d \cdot k^2)$.* ***(Sections 4 and 7)***
b. *$O(k^2 \cdot \Delta_{out})$ time with $\Theta(\log(k + \Delta_{out}))$ bits at each robot under the local model with a DFS based algorithm. For $f \le k$ robot crashes, the time bound becomes $O(f \cdot k^2 \cdot \Delta_{out})$.* ***(Sections 5 and 7)***
c. *$O(k \cdot D)$ time with $O(k \log(k + \Delta_{out}))$ bits at each robot under the 1-hop model. The time bound also applies for the case of $f \le k$ robot crashes.* ***(Sections 6 and 7)***

To the best of our knowledge, these are the first results for Dispersion on directed graphs, see also Table 1. The results in Theorem 1 are close to the $\Omega(k \cdot \min\{k, d\})$ time lower bound for any $d > 0$; it remains open to close the gaps between the lower and upper bounds. Additionally, Theorem 1(b) is the best among the algorithms as it uses the asymptotically optimal memory per robot.

Although Theorem 1 guarantees robots correctly reach Dispersion configuration (at most a fault-free robot per node), the robots individually do not know whether Dispersion has been finished for all the robots and hence they cannot terminate. We note here that Theorem 1 can be made terminating if the respective problem parameters on the individual time bounds, k, Δ_{out}, D, d, f, are known to robots beforehand. The idea is for each robot to maintain a *round counter* initiated to 0 and incremented by 1 in every round. Each robot can then terminate as soon as its counter reaches the reported runtime of each result in Theorem 1. The only change in the respective algorithm is the maintenance of a counter variable. Since $f \le k$, for the algorithms with time complexity based only on parameters f, k, Δ_{out} (Theorem 1.b), the memory remains asymptotically the same. For the algorithms with parameters d or D involved in their time complexity (Theorems 1.a and 1.c), their memory bounds have an additional factor of $O(\log(k + d))$ or $O(\log(k + D))$, respectively. We summarize the results in the following corollary.

Corollary 1. *Dispersion can be solved with termination with the runtime guarantees reported in Theorem 1*

a. *If in Theorem 1(a), memory is $O(\log(k + d) + k \cdot \log(k + \Delta_{out}))$ bits at each robot and the parameters d, k, Δ_{out}, and f are known to robots a priori.*
b. *If in Theorem 1(b), no change in memory asymptotically but the parameters k, Δ_{out}, and f are known to robots a priori.*
c. *If in Theorem 1(c), memory is $O(\log(k + D) + k \cdot \log(k + \Delta_{out}))$ bits at each robot and the parameters D, k, Δ_{out}, and f are known to robots a priori.*

Techniques. The impossibility result is established considering an initial configuration on a not-strongly-connected directed graph so that there is always a node with two robots positioned on it, irrespective of the technique used. The time lower bound $\Omega(k)$ is obtained considering a (strongly connected) directed cycle. The time lower bound $\Omega(D^2)$ is established by constructing a (strongly connected) directed graph modified from an undirected tree G with $k = n$ nodes and diameter (height) $D = \omega(1)$ such that any deterministic algorithm needs $\Omega(D^2)$ rounds to achieve Dispersion. The time lower bound $\Omega(k \cdot \min\{k, d\})$ for a DFS traversal based algorithm is proved considering a strongly connected directed graph where dispersing $k/2$ robots on $k/2$ different nodes needs exactly $\min\{k/2, d\}$ rounds each, requiring in total at least $\Omega(k \cdot \min\{k, d\})$ rounds to disperse all k robots successfully. The memory lower bound $\Omega(\log k)$ bits per robot is established considering the minimum number of bits needed to distinguish IDs of two different robots as per the robot model used.

As our main contributions, the time upper bound $O(k^2 \cdot d)$ is achieved through simulating Stephen Kwek's exploration algorithm [11] for a strongly connected directed graph of deficiency d appropriately to solve DISPERSION, settling a robot on each new node visited while under exploration. Note that Kwek's algorithm uses a *depth first search* (DFS) traversal. The main drawback of Kwek's algorithm for DISPERSION is that the memory requirement per robot is $O(k \cdot \log(k + \Delta_{out})$ bits, which is factor $O(k)$ away from the lower bound of $\Omega(\log(k + \Delta_{out}))$ bits per robot. Our second algorithm achieves the time upper bound $O(k^2 \cdot \Delta_{out})$ running the DFS algorithm and improving the memory requirement so that only $O(\log(k + \Delta_{out}))$ bits per robot is enough, which is an improvement of $O(k)$ factor compared to the adaption of Kwek's algorithm. The main difficulty in improving memory is to backtrack over a directed edge $u \to v$, i.e., to find a directed path from v to u. The challenge is, if not done carefully, the traversal gets stuck failing to reach u from v (which we call the *local maximum* problem). We avoid the local maximum problem by ranking the nodes (with settled robots) based on the order of the first DFS arrival; for any two robots r_1, r_2 belonging to a DFS with ID j, r_1 has a rank lower than r_2 if r_1 settles before r_2 (no two robots belonging to a DFS settle at the same time). During the traversal, if it is found that there is a backtracking path to a lower ranked node, then the backtracking path is updated to point to that lower ranked node. We also devise a technique that bounds the length of each backtrack path to $\leq k-1$ edges, even when DFSs start in parallel from multiple nodes. Furthermore, when a DFS meets another, we subsume one DFS by another based on the unique DFS IDs derived from the robot IDs. Interestingly, we show that there is only additional overhead doing this subsumption. Putting these ideas together, we show that visiting $k \cdot \Delta_{out}$ directed edges takes the DFS traversal to at least k different nodes, allowing to settle k robots, one on each node, with total time $k \cdot \Delta_{out} \cdot k = O(k^2 \cdot \Delta_{out})$ rounds.

As our another main contribution, the time upper bound $O(k \cdot D)$ is established exploiting the information that can be broadcasted and gathered from the 1-hop communication so that in every D rounds, at least a robot can be settled on a previously empty node. Having 1-hop information is of no use in the DFS based algorithm as it cannot exploit the information collected through 1-hop communication. To bookkeep the information to successfully run broadcast and gather through 1-hop communication, the memory is increased to $O(k \cdot \log(k + \Delta_{out}))$ bits per robot.

For crash faults, we analyze the working principles of the fault-free cases of the above algorithms to see how much extra work (time and memory) is required for each crash. We show that the extra time is proportional to $O(f \cdot k^2 \cdot d)$ and $O(f \cdot k^2 \cdot \Delta_{out})$, respectively, for the first two DFS-based algorithms and zero for the third algorithm; the memory bound stays the same as in the fault-free cases in all the algorithms.

Related Work. The literature studied DISPERSION mostly in undirected graphs. Two communication models were considered, local and global.

A significant amount of work in the literature is in the local model which we discuss first. Augustine and Moses Jr. [4] were the first to study 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. 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. Kshemkalyani and Ali [5] provided an $\Omega(k)$ time lower bound for arbitrary graphs for $k \leq n$. They then provided three deterministic algorithms on 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, where Δ and D, respectively, are the maximum degree and diameter of the graph. Kshemkalyani et al. [6] provided an algorithm that runs in $O(\min(m, k\Delta) \cdot \log k)$ time using $O(\log n)$ bits of memory at each robot on arbitrary graphs, given that parameters m, n, k are known to robots. Shintaku et al. [12] established the same time bound without robots knowing m, n, k. Recently, Kshemkalyani and Sharma [9] improved the time bound to $O(\min(m, k\Delta))$ keeping memory $O(\log(k + \Delta))$ bits at each robot. For grid graphs, Kshemkalyani et al. [8] provided an algorithm that runs in $O(\min(k, \sqrt{n}))$ time using $O(\log k)$ bits at each robot. Randomized algorithms are presented in [10] to solve DISPERSION from rooted initial configurations where the random bits are mainly used to reduce the memory requirement at each robot.

Recently, there is some work in the *global* model which we discuss now. In the global model, the robots on the graph can communicate with each other despite their locations on the graph. Kshemkalyani et al. [7] provided two deterministic algorithms on arbitrary graphs: (i) the first algorithm using $O(\log(k + \Delta))$ bits at each robot with $O(\min(m, k\Delta))$ time, and (ii) the second algorithm using $O(\log D + \Delta \log k)$ bits at each robot with $O((D + k)\Delta(D + \Delta))$ time. For grid graphs, Kshemkalyani et al. [8] provided a $O(\sqrt{k})$ time algorithm with $O(\log k)$ bits at each robot.

DISPERSION in dynamic (undirected) graphs was considered in [13]. Dispersion under crash faults was considered in [14] and under byzantine faults was considered in [15,16]. In this paper, we initiate study of DISPERSION on directed graphs and present results using the local and 1-hop communication models.

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., [17–22]. The vast majority of works considered undirected graphs. It was shown that a robot can explore an anonymous graph using $\Theta(D \log \Delta)$-bits of memory and the runtime of the algorithm is $O(\Delta^{D+1})$ [20]. In the model where graph nodes also have memory, Cohen et al. [18] 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 [22]. The collective exploration by a team of robots is studied in [23] for trees. In directed graphs, two cooperating robots were considered to explore and learn about the anonymous strongly connected directed graph in [24]. The exploration and mapping of directed graphs was also done by using a pebble in [25]. Exploration in directed graphs is also studied for a searcher that tries to minimize the tour required to visit all the nodes in a graph [26] using both randomized and online algorithms [27]. Exploration of directed graph has been parameterized by deficiency of a directed graph. Deficiency as a measure of difficulty was introduced by Shay Kutten [28]. Deng and Papadimitriou [29] studied the competitive ratio of an online exploration algorithm vs offline ones with respect to the number of edge traversals. They showed that an Eulerian graph has a competitive ratio to be 2 and proposed an optimal exploration algorithm for Eulerian graphs. They proposed an exploration algorithm for a graph with deficiency d that has a ratio $d^{O(d)}$. Albers and Henzinger [26] improved the dependency on deficiency to $d^{O(\log d)}$. Fleischer and Trippen [30] improved the bound further to show a polynomial dependence with an algorithm having competitive ratio $O(d^8)$.

Another problem related to DISPERSION is the scattering of robots in graphs. Scattering has been studied for rings [31,32] and grids [33,34]. 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 (undirected) graphs, e.g., see [35].

We refer readers to [36,37] for recent developments in the above mentioned research topics.

2. Model and preliminaries

Graph. Let $G = (V, E)$ be an n-node arbitrary, connected, unweighted, directed, port-labeled, anonymous graph with m edges, i.e., $|V| = n$ and

$|E| = m$, such that nodes do not have identifiers but, at any node $v_i \in V$, its incident (outgoing) edges are uniquely identified by a *label* (aka port number) in $[1, \delta_{out}^{v_i}]$, where $\delta_{out}^{v_i}$ is the *out-degree* of v_i. The *out-degree* of graph G is $\Delta_{out} = \max_{1 \le i \le n} \delta_{out}^{v_i}$, i.e., the maximum $\delta_{out}^{v_i}$ among the nodes in G. There is no bandwidth limitation on the edges, i.e., any number of robots are allowed to traverse an edge at any time following the direction of the arrow. The graph nodes do not have memory.

For any two nodes $u, v \in G$, we denote by path $p(u, v)$ the sequence of consecutive directed edges starting from u and ending at v. The diameter D is the longest shortest directed path $p(u, v)$ between any two nodes $u, v \in G$. A directed graph G is said to be *strongly connected* if for each pair of nodes u, v, there is a directed path $p(u, v)$ reaching to v from u as well as a directed path $p(v, u)$ reaching to u from v.

Kutten [28] introduced deficiency as the number of edges needed to add to the graph to make it Eulerian. A directed graph is *Eulerian* if each vertex has the equal number of incoming and outgoing edges. *Deficiency*, denoted as d, is defined as the sum of differences between the number of incoming edges and outgoing edges at a sink in the directed graph. Formally,

$$d = \sum_{v \in V} \begin{cases} \delta_{in}^{v} - \delta_{out}^{v}, & \text{if } \delta_{in}^{v} > \delta_{out}^{v} \\ 0, & \text{otherwise} \end{cases}$$

Robots. Let $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$ be a set of $k \le n$ robots residing on the nodes of G. No robot can reside on the edges of G, but one or more robots can occupy the same node. Each robot has a unique $O(\log k)$-bit ID taken from $[1, k^{O(1)}]$. The ID of a robot r_i is denoted by $r_i.ID$ with $r_i.ID = i$. Furthermore, it is assumed that each robot is equipped with memory to store information.

Communication Model. There are two communication models: local and global [7,8,13]. In the local model, a robot can only communicate with other robots co-located on the same node. In the global model, a robot can communicate with any other robot, irrespective of their positions on graph. We define an intermediate model of 1-hop communication in which a robot can communicate to another robot, if they are located in neighboring nodes. This paper considers the local and 1-hop communication models.

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 $r_i \in \mathcal{R}$ that is at some node v_i, another robot r_j at v_i (and, in the 1-hop model, robots positioned at v_i's neighbor nodes) can observe the memory of r_i. Robot r_i can also observe its own memory;
(ii) *Compute:* Robot r_i may perform an arbitrary computation using the information observed during the "communicate" portion of the cycle. This includes determining a (possibly) port to use to exit v_i and the information that is communicated to the robot r_j that is at v_i,
(iii) *Move:* At the end of the cycle, r_i communicates the information to r_j at v_i (so that r_j will store/update in its memory), and exits v_i using the computed port to reach a neighbor of v_i.

Rooted and General Initial Configurations. An initial configuration is *rooted* if all $k \le n$ robots are positioned on a single node of G. In a *general* initial configuration, there are robots on $1 < k' < k$ different nodes with at least one node among k' has multiple robots. DISPERSION is trivially solved when $k' = k$.

Crash Faults. A robot is susceptible to crash anytime and once it crashes at time $t \ge 0$, it stops communicating with other robots, i.e., at any time $t' > t$, it appears like it has vanished from the system.

Dispersion. DISPERSION in directed graphs is formally defined as follows. We denote by $f \le k$ the number of faults. The case of $f = 0$ is a fault-free DISPERSION and the case of $f > 0$ is a faulty DISPERSION.

Definition 1 *(DISPERSION).* Given $k \le n$ robots positioned initially arbitrarily on the nodes of an n-node anonymous directed graph $G = (V, E)$ with $0 \le f \le k$ robots that may crash at any time, the robots reposition autonomously such that each non-faulty robot is on a distinct node of G and stays stationary thereafter.

Activation, Time, and Memory. Following previous works [4–6,8–10,13], we consider the synchronous setting where every robot is active in every CCM cycle, performing the cycle in synchrony. Time is measured in rounds. Another parameter is memory which we measure as the number of bits.

3. Impossibility and lower bounds

We discuss here one impossibility result, three time lower bounds, and one memory lower bound for DISPERSION on directed anonymous graphs. These results collectively show the difficulty in obtaining a solution as well as obtaining fast runtime and low memory when a solution exists.

We first discuss the impossibility result which is established looking at the graph type, strongly connected or not strongly connected. Consider the case of a directed graph G that is not strongly connected and all the robots are located on a single node $u \in G$. We present the following impossibility result.

Theorem 2. *It may not always be possible to solve DISPERSION on a directed graph that is not strongly connected.*

Proof. By definition, a graph $G = (V, E)$ is strongly connected if and only if, for each pair of vertices $u, v \in V$, there is a directed path $p(u, v)$ from u to v and directed path $p(v, u)$ from v to u. If G is not strongly connected then there exists at least a pair of vertices $u, v \in V$ such that at least $p(u, v)$ or $p(v, u)$ is not available. Assume that, in graph G, v is not reachable from u, i.e., directed path $p(u, v)$ is not available, but the path $p(v, u)$ is available. Let $U \subset V$ be the set of vertices that are reachable from u. As v is not reachable from u, v must also not be reachable from every vertex of the set U, otherwise there would be a directed path $p(u, v)$ from u to v violating our assumption. Since v is not reachable from u, it cannot be in the set U and it must be the case that $|U| \le n - 1$. Consider now the rooted initial configuration of n robots located at node u. Since v is not reachable from u (and all the nodes in the set U), at any time there must be at least two robots located at some node in U. That is, DISPERSION is not achieved. □

Remark. Theorem 2 considers a rooted initial configuration of n robots on a node u and a node v that is not reachable from u. If v is reachable from u, the problem has a solution. However, since G is not known a priori, an algorithm may not be designed that is able to achieve such a solution configuration.

We now present three time lower bounds for solving DISPERSION on strongly connected directed graphs.

Theorem 3. *Any deterministic algorithm for DISPERSION on directed graphs requires $\Omega(k)$ rounds.*

Proof. Consider a directed cycle G. Note that G is strongly connected since for any two nodes $u, v \in G$, there is a directed path $p(u, v)$ and $p(v, u)$. Consider a rooted initial configuration of $k \le n$ 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. To reach a node to settle, some robot must travel $k - 1$ directed edges of G, taking $k - 1$ rounds. □

For $k = n$, we present the following time lower bound for solving DISPERSION on directed graphs.

Theorem 4. *For $k = n$, there exists a directed graph G with n nodes and diameter D with $D = \omega(1)$ such that any deterministic algorithm for* DISPERSION *requires $\Omega(D^2)$ rounds.*

Proof. We use the lower bound proof for exploration due to Disser et al. [38] to prove this lower bound. The goal in exploration is to visit all nodes of the graph. For $k \le n$, the goal in DISPERSION is to visit k nodes of the graph. Therefore, for the case of $k = n$, all n nodes need to be visited to place one robot per node and hence any solution to exploration would solve DISPERSION for $k = n$ robots. It was shown in [4] that a lower bound for exploration applies to DISPERSION for the case of $k = n$. Therefore, we borrow the lower bound of [38] for DISPERSION for the case of undirected graphs and prove a lower bound for DISPERSION in directed graphs. Disser et al. [38] proved a lower bound for exploration assuming a rooted initial configuration in which $k = n$ robots are on a single node v_{root} of an undirected tree graph G. Moreover, they assumed that the nodes of G have unique identifiers and the robots have global communication. Specifically, they showed that: Using $k = n$ robots, there exists a tree G on n vertices with diameter (height) $D = \omega(1)$ such that any deterministic exploration strategy requires at least $D^2/3 = \Omega(D^2)$ steps to explore G. We convert G to a directed graph G' as follows: For each undirected edge (u, v), replace it with two directed edges $u \rightarrow v$ and $v \rightarrow u$. The resulting directed graph G' is strongly connected. Moreover, in G', exploration is done essentially similar as in G in Disser et al. [38].

In our model, nodes are indistinguishable which is in contrast and weaker than the model used by Disser et al. [38] in their proof. Therefore, the $\Omega(D^2)$ lower bound applies to DISPERSION on directed graphs, combined with result of [4] that shows an exploration lower bound applies to DISPERSION for $k = n$. □

We prove the following time lower bound for DISPERSION on directed graphs using a DFS traversal based algorithm, which improves on $\Omega(D^2)$ when $k > O(D)$.

Theorem 5. *For deficiency $d > 0$, any DFS traversal based algorithm for* DISPERSION *on directed graphs requires $\Omega(k \cdot \min\{k, d\})$ rounds.*

Proof. The proof construction is given in Fig. 1. The graph G in Fig. 1 is strongly connected and $k \le n$ robots start from node u. The proof uses the fact that after $k/2$ robots settle at the top arc in $k/2$ rounds, for the rest $k/2$ robots, it needs $k/2$ rounds each for a robot to settle. This gives in overall $\Omega(k^2)$ rounds lower bound. In the first $k/2 - 1$ rounds, $k/2$ robots settle at $k/2$ nodes on the top arc and a robot settles on a column node in the round $k/2$. After that, each new node in the column is visited in every next $k/2 + 1$ rounds, after visiting completely the nodes on the top arc.

Notice that the deficiency of the graph G in Fig. 1 is $n - k/2 - 1$. We can use deficiency as a parameter for the lower bound and get the following result. Any DFS traversal based algorithm needs at least $\Omega(k \cdot d)$ rounds to achieve DISPERSION. The theorem follows combining the above two results. □

For an arbitrary directed graph, the lower bound corresponds to the size of the directed cycles that contains a deficient node. Intuitively, any DFS-based exploration algorithm can only visits one of the cycles for each visit to the deficient node. Consequently, a deficient node is also visited at least the deficiency number of times. We finally prove a lower bound of $\Omega(\log k)$ bits at each robot for any deterministic algorithm.

Theorem 6. *Any deterministic algorithm for* DISPERSION *on n-node directed anonymous graphs requires $\Omega(\log k)$ bits at each robot, where $k \le n$ is the number of robots.*

Proof. Suppose initially all $k \le n$ robots are on a rooted configuration. To be able to distinguish two robots their IDs need to be uniquely differentiated. To emphasize, in the absence of identifiers, all anonymous robots would behave the same under the same algorithm. For example, if two robots start at the same node: if one decides to move, the other also moves; and if one decides to stay, the other also stays. Thus dispersion cannot be achieved as it requires at most one robot at a node. Therefore, to distinguish one robot from another, k robots need k distinct IDs and at least $\log k$ bits are necessary to have such distinct identities. □

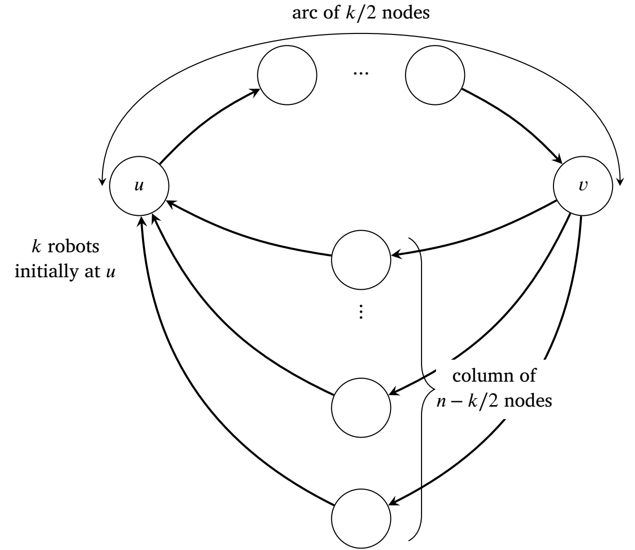


Fig. 1. Lower bound construction for a DFS traversal based DISPERSION algorithm.

4. DISPERSION using Kwek's exploration algorithm

In this section, we present an algorithm that achieves DISPERSION by following an online exploration strategy by Stephen Kwek [11]. On a high level, Kwek's algorithm performs DFS on a directed graph by keeping the map of the graph in the robot's memory by having an edge list and a vertex list of a graph. Using that, it can compute the shortest path in the explored part of the graph from a node without any unseen outgoing edges to a node previously visited. In the following, we present Algorithm `DFS_kwek` that executes Kwek's algorithm in the dispersion model and achieves DISPERSION. We can have identifiers at a node by a robot that is settled there. If there is no settled robot at a node u, the robot with highest ID among the visiting robot settles at u, and other robots use the ID of the settled robot as an identifier for the node. Each edge at a node is marked by a port number and after traversing an edge (u, v), a robot can store it in its memory with corresponding port numbers as a quadruple $[u, p_{out}, v, p_{in}]$. Thus it is possible to identify each edge and its direction at each node. Kwek's algorithm completes exploration of the graph with $O(dn^2 + m)$ edge traversals, where d is the *deficiency* of the directed graph [11]. We have the following corollary.

Corollary 2. *`DFS_kwek` achieves* DISPERSION *in $O(dk^2 + m)$ rounds with $O(k \log(k + \Delta))$ memory, where Δ is the maximum degree of a node, d is the deficiency, m is the number of edges, and k is the number of robots.*

Proof. Since we have at most k robots, with $k \le n$, the last robot that settles would visit at most $k - 1$ nodes before it settles. Visiting $k - 1$ nodes under the Kwek's algorithm would cost $O(dk^2 + m)$. To store the graph in the memory of the robot, it needs to store at most k^2 edges. Since each edge of the graph needs at most $\log(k + \Delta)$ bits of memory, the robot needs at most $2k \log(k + \Delta)$ bits of memory. Hence, `DFS_kwek`

Table 2
Summary of variables used by each robot.

Symbol	Description	Initialization
$settled$	Indicates a settled robot at a node	0
$nodeRank$	Indicates the DFS order in which the robot is settled	0
$backtrackTargetNode$	Indicates the ID of the last settled robot at a node with unvisited ports	$null$
$cycleClosingNode$	Indicates whether the node closes the current traversal cycle	$null$
$cycleClosingNodeRank$	Indicates value in the $nodeRank$ variable of the cycle closing node	$null$
$DFSID$	ID of the DFS traversal starting from a node (Used only in general case)	$null$
$recentPort$	Points to the last port via which the unsettled robots exited	0
$phase$	Indicates $explore$, $retrace1$, or $retrace2$ phase	$explore$
$status$	Indicates the $visited$ or $fully\text{-}visited$ status of a node	$null$
$rootPort = (port, rank)$	Indicates the port and associated rank at a node to reach DFS root	$(null, null)$

Fig. 2. A state transition diagram depicting each phase of the DFS algorithm and what prompts a transition from one phase to another. Initially, the algorithm starts from the explore phase.

achieves DISPERSION in $O(dk^2 + m)$ rounds with $O(k \log(k + \Delta))$ bits memory per robot. □

5. DFS dispersion algorithm

We present here a deterministic DFS traversal based (or simply DFS) algorithm. We first discuss the variables used, then an overview of the approach, then a detailed discussion, and finally the analysis.

Variables. Each robot maintains the ten variables to keep track of the DFS traversal as described in Table 2. Variable $r_i.DFSID$ is used only in the general case. The $r_i.settled$ variable indicates the robot r_i that is settled at a node v_i and r_i remains at v_i forever. For a settled robot r_i, $r_i.status \in \{visited, fully\text{-}visited\}$; $r_i.status$ becomes $visited$ when v_i is visited by DFS for the first time and becomes $fully\text{-}visited$ after all the outgoing ports at v_i were already visited.

Overview of the Algorithm. The algorithm simulates a DFS traversal in three phases: $explore$, $retrace1$, and $retrace2$. Fig. 2 depicts how the DFS transitions from one phase to another along with what happens during each phase and what prompts a phase transition. Initially, the DFS is in $explore$ phase. At any time, when the DFS visits an unvisited node, then it settles a robot on it and marks $visited$ (Line 5 of Algorithm 1). When the DFS reaches to a node with a settled robot, say v_x, then backtracking is required. The traversal executes $retrace1$ phase (Line 12) to find a *backtrack target node*, (if any) and $retrace2$ phase (Line 20) setting backtrack path, one after another, and then it continues $explore$ phase from the backtrack target node. If v_x has *visited* status, then $retrace1$ and $retrace2$ phases start and end at v_x (i.e., v_x becomes the *cycle closing node*, Line 7). Otherwise, these phases start and end at some other node $v_y \neq v_x$ that has *visited* status (i.e., v_y becomes the cycle closing node, Line 9). The phases $retrace1$ and $retrace2$ compute a backtrack path setting variable $rootPort$ so that backtracking can be done traversing the $rootPort$ set. After finishing $retrace1$ and $retrace2$ and returning to the cycle closing node, the traversal either (i) moves to the backtrack target node (if not null) to execute the $explore$ phase (Line 21) or (ii) takes $rootPort$ set (if backtrack target node is null) to reach a $visited$ node to continue again $retrace1$ and $retrace2$ phases (Line 17). This process stops as soon as the robots are settled solving DISPERSION.

We first discuss the rooted case (the pseudocode is in Algorithm 1). After that, we will discuss the general case (the pseudocode is in Algorithms 1 and 2).

5.1. Rooted algorithm

Let $\mathcal{R}(v)$ be the set of $k \leq n$ robots $\{r_1, r_2, \ldots, r_k\}$ positioned initially on node $v \in G$ with $r_i.ID = i$, forming a rooted configuration. At round 1, the highest ID robot $r_k \in \mathcal{R}(v)$ settles on v setting $r_k.settled \leftarrow 1$, $r_k.status \leftarrow visited$, $r_k.recentPort \leftarrow 1$, and $r_k.nodeRank \leftarrow 1$. The robots $\mathcal{R}(v) \backslash \{r_k\}$ exit v via $r_k.recentPort$. The exiting robots carry $r_k.nodeRank$, the rank of the robot settled at v.

At round 2, suppose the robots $\mathcal{R}(v) \backslash \{r_k\}$ arrive at node w. The robot r_{k-1} settles at w setting $r_{k-1}.settled \leftarrow 1$, $r_{k-1}.status \leftarrow visited$, $r_{k-1}.recentPort \leftarrow 1$, and $r_{k-1}.nodeRank \leftarrow r_k.nodeRank + 1$ (note that $r_k.nodeRank$ was carried from v). The robots in $\mathcal{R}(v) \backslash \{r_k, r_{k-1}\}$ exit w via $r_{k-1}.recentPort$. The traversal process continues this way. If k different nodes are visited before reaching to a node with an already settled robot, DISPERSION is achieved. However, if the DFS reaches to a node with a settled robot, backtracking is required. In backtracking, the DFS needs to (in sequence)

- find a cycle closing node;
- starting from the cycle closing node, find a backtrack target node b from which the DFS can again continue its $explore$ phase settling the robots at new empty nodes;
- create/update backtrack path setting $rootPort$ information appropriately;

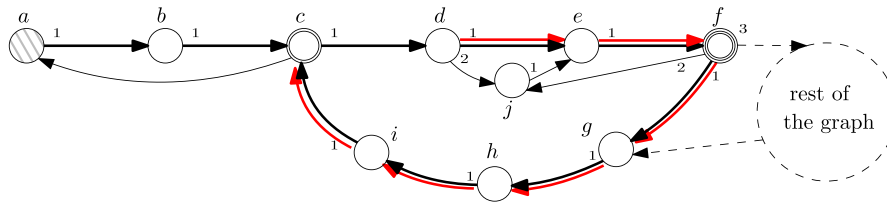


Fig. 3. An illustration of the DFS traversal, starting from node a, after it reaches node c traversing the sequence of edges $a \to b \to c \to d \to \ldots \to i \to c$ (shown in bold). Node c is a *visited* node where $ActivePath(a)$ ends and hence c can serve as a cycle closing node. The node f that is farthest from c in the cycle $c \to \ldots \to f \to \ldots \to c$ with unvisited out-going ports becomes the backtrack target node during $retrace1$. During $retrace2$, (i) nodes g, h, i have no unvisited out-going ports and marked *fully-visited* and (ii) the current port from d to i are stored in $rootPort$ (the corresponding edges are shown in red).

Algorithm 1: Algorithm DFS l.

1 initially, node rank is set to 0 for all robots;
2 let r_i be a robot that belongs to DFS l at node v;
3 let r_X be the settled robot at node v (if one exists; otherwise DFS l settles r_X);
4 **if** $r_i.phase == explore$ **then**
5   **if** *v is an unvisited node* **then**
6   settle robot r_X, write $r_X.recentPort \leftarrow r_X.recentPort + 1$, $r_X.status \leftarrow visited$, $r_X.nodeRank \leftarrow r_i.nodeRank + 1$, and exit v via $r_X.recentPort$;
7   **if** *v is a visited node* **then**
8   set v (a.k.a., r_X) as the cycle closing node and set $r_i.phase \leftarrow retrace1$;
9   **if** *v is a fully-visited node* **then**
10   traverse $rootPort$ pointers starting from v until finding a $visited$ node, set that node as the cycle closing node, and set $r_i.phase \leftarrow retrace1$;
11 **if** $r_i.phase == retrace1$ **then**
12   **if** *r_i is the smallest ID robot on v* **then**
13   r_i revisits the cycle following $recentPort$ pointers and while doing so computes the backtrack target node b that is farthest from v; `// v can be the backtrack target node`
14   after returning to v, set $r_i.phase \leftarrow retrace2$;
15 **if** $r_i.phase == retrace2$ **then**
16   **if** *r_i is the smallest ID robot on v* **then**
17   **if** $b == null$ **then**
18   mark all nodes in the cycle as fully-visited; `// root ports are already set`
19   follow $rootPort$ at v until finding a $visited$ node, set v as the cycle closing node, and set $r_i.phase \leftarrow retrace1$;
20   **else**
21   r_i revisits the cycle following $recentPort$ pointers and while doing so set the $r_y.rootPort \leftarrow (r_y.recentPort, r_1.cycleClosingNodeRank)$ as well as mark all the nodes after the backtrack target node b and before the cycle closing node $fully\text{-}visited$;
22   after returning to v, follow again $recentPort$ pointers to reach b then set $r_i.phase \leftarrow explore$, $r_b.recentPort \leftarrow r_b.recentPort + 1$, and exit via $b.recentPort$ (r_b is the robot settled at b);

- go to backtrack target b from the cycle closing node to continue $explore$ phase.

Finding cycle closing and backtrack target nodes happens in $retrace1$ whereas setting backtrack path and going to the cycle closing node happen in $retrace2$.

We describe separately below how each of these steps is done by the DFS.

Finding cycle closing node. Suppose, at some round $t > 1$, the DFS reaches a node x with an already settled robot r_x. Since r_x was settled, $r_x.status \in \{visited, fully\text{-}visited\}$ and $r_x.settled = 1$.

- If $r_x.status = visited$, then x becomes the cycle closing node and robot r_x sets $r_x.cycleClosingNode \leftarrow 1$, indicating that the cycle ends at it. Fig. 3 provides an illustration where the cycle ends at node c and c becomes the cycle closing node.
- $r_x.status = fully\text{-}visited$, then the DFS takes the $rootPorts$ (details later on how the $rootPorts$ are set) until reaching the first node z with status $r_z.status = visited$ on the settled robot r_z. z becomes the cycle closing node and robot r_z sets $r_z.cycleClosingNode \leftarrow 1$, indicating that the cycle ends at it. In Fig. 4(iii), when node g ($fully\text{-}visited$) is visited from f, node d is picked as a cycle closing node following the root port from g.

In the first case, traversing the $recentPort$ pointers from x brings the DFS back to x, which we denote as the cycle C. In the second case, traversing the $recentPort$ pointers from z upto x and the $rootPort$ pointers from x brings the DFS back to z, which we denote as the cycle C'.

Finding backtrack target node. The DFS backtracks to a node such that it has at least an unvisited port left. Backtrack target node b is computed as follows. Let the DFS be at the cycle closing node x. The lowest ID robot in the traversal, r_1, sets $r_1.phase \leftarrow retrace1$. Other unsettled robots stay at x waiting for r_1 to finish $retrace1$. If $r_x.recentPort < \delta^x_{out}$, r_1 sets $r_1.backtrackTargetNode \leftarrow r_x$ (initially $null$). The robot r_1 then leaves r_x following $r_x.recentPort$ which makes r_1 traverse the cycle C again and return to x. While in $retrace1$ phase, for every node y it visits in C, r_1 updates $r_1.backtrackTargetNode$ to the ID of the robot settled at y if $r_y.recentPort < \delta^y_{out}$. This gives the backtrack target node b that is farthest from x in C. In Fig. 3, node f in the cycle becomes the backtrack target node b since nodes g, h, i do not have any unvisited port left and f is the farthest node from c in the cycle with unvisited ports.

There may be situations in which this procedure returns b null. If no node with unvisited ports left in C, then no node in the cycle C is eligible to become the backtrack target node b. In this situation, backtrack target node b is computed as follows. When there is no node in C that is eligible to be a backtrack target node, it can be identified at the end of $retrace1$, since it returns $r_1.backtrackTargetNode$ null. In that case, after $retrace2$ finishes ($retrace2$ marks all node in the cycle $fully\text{-}visited$), the $rootPorts$ are taken from the cycle closing node until the DFS reaches a $visited$ node z. The node z becomes a cycle closing node and the search for the backtrack target node happens in the cycle that is closed by the cycle closing node z. In Fig. 4, when the cycle $d \to e \to f \to g \to d$ is closed at d (Fig. 4(iii)), there is no node in the cycle that is eligible to become a backtrack target node, and hence $rootPort$ set is taken to reach c (Fig. 4(iv)) which becomes a cycle closing node. In Fig. 4(iv), when c becomes a cycle closing node, the cycle $c \to d \to e \to f \to c$ is visited in $retrace1$ and finds c as a backtrack target node. There may also be the case that no node is eligible to become a backtrack target node in this new cycle closed by z. In this case, the process again repeats visiting the $rootPorts$ set (after finishing $retrace1$ and $retrace2$ phases) starting from z to reach a $visited$ node.

Creating/updating backtrack path (i.e., setting $rootPorts$). Finding backtrack target b ends at the cycle closing node x. The phase is now switched to $retrace2$. r_1 is responsible again to execute this phase. r_1 sets $r_1.phase \leftarrow retrace2$, $r_1.cycleCloseNodeRank \leftarrow r_x.nodeRank$, and re-traverses the cycle again doing two things. First it sets $rootPort$ on the nodes on the cycle except node x (the red directed edges in Figs. 3, 4, and 5 depict the root ports set). This information will be

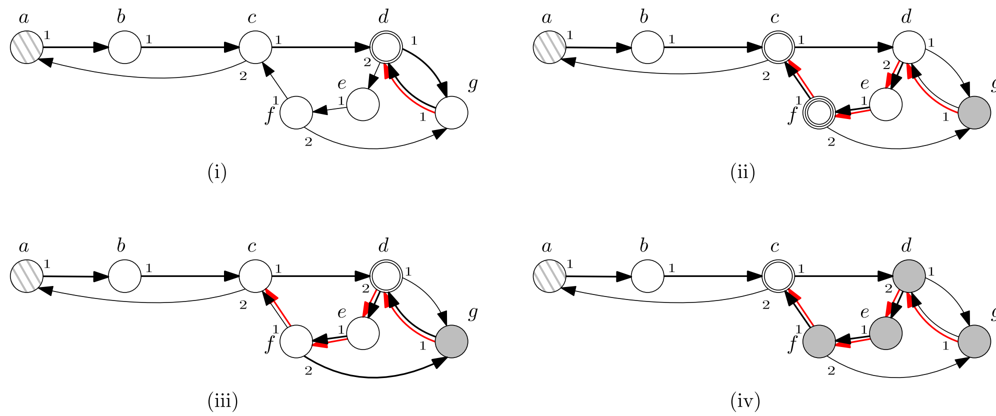


Fig. 4. An illustration of DFS traversal where a fully-visited node is visited on the explore phase. In the figure, we have four instances of the graph. (i) The node d becomes the cycleClosingNode for the cycle $d \to g \to d$ and also a backtrack target node; (ii) DFS continues from d visiting $e \to f \to c$; c becomes the cycleClosingNode and f becomes the backtrack target node; (iii) DFS continues from f visiting g; since g is fully visited, the DFS takes $RootPath()$ to go to d which becomes the cycleClosingNode; and (iv) DFS traverses $d \to e \to f \to g \to d$ cycle; since no node in the cycle has unvisited ports, d, e, f are marked $fully\text{-}visited$, and c becomes the cycleClosingNode after taking $RootPath()$ from d.

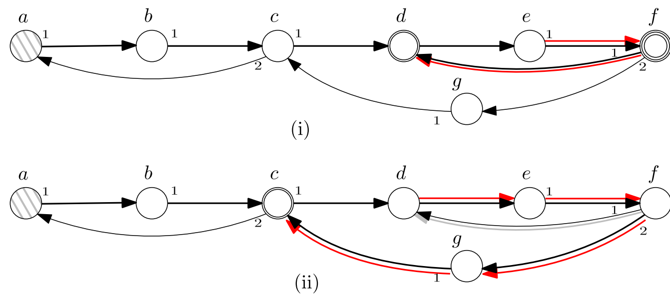


Fig. 5. An illustration of root port information update during DFS traversal. (i) Node d becomes the cycle closing node after DFS traverses $a \to b \to c \to d \to e \to f \to d$ and thus root ports are set in e and f pointing to f and d, respectively; f becomes the backtrack target node. (ii) Node c becomes the cycle closing node after DFS continues from f visiting g and then c. Since c has rank smaller than d, the root port at f is updated to point g and g points to c.

useful to reach a $visited$ node to perform $retrace1$ and $retrace2$ phases after a node becomes $fully\text{-}visited$ (note that $fully\text{-}visited$ node cannot act as a cycle closing node). Let $y \neq x$ be a node in the cycle. It sets $y.rootPort \leftarrow (y.recentPort, r_1.cycleClosingNodeRank)$. Keeping $cycleClosingNodeRank$ in the memory of y helps to update $y.rootPort$ information later to reach a cycle closing node that may have rank smaller than the information stored at $y.rootPort$, avoiding the local maximum problem. The second task is to update $status$ of each node y (if any) in the cycle between $b = r_1.backtrackTargetNode$ and x with $r_y.status \leftarrow fully\text{-}visited$ (in Fig. 3, nodes g, h, i are marked $fully\text{-}visited$ during $retrace2$).

We now discuss the situation that results in updating the backtrack path. Consider a node y which has $rootPort$ information set. Suppose it again becomes the part of a cycle such that r_1 is carrying $cycleClosingNodeRank$ that is smaller than $y.cycleClosingNodeRank$. y updates its $rootPort$ information such that $y.rootPort$ has this new smaller cycle closing node rank and the $recentPort$ associated with this new cycle. Fig. 5 provides an example of the $rootPort$ information update. In the example of Fig. 5, the first cycle detected was the $d \to e \to f \to d$ cycle and the rootpath was set from e to d. Subsequently, another cycle $c \to d \to e \to f \to g \to c$ was detected. The rootport pointers at e and f are updated to point to c, and new rootport pointers are set at d and g.

Going to the backtrack target b to continue $explore$ phase. After r_1 reaches x finishing $retrace2$, the traversal has sufficient information to resume the $explore$ phase, if the backtrack target node b is in the cycle (i.e., $b \neq null$). Robot r_x (on cycle closing node x) sets $r_x.cycleClosingNode \leftarrow 0$, robot r_1 sets $r_1.cycleClosingNodeRank \leftarrow null$, and all the waiting robots at x including r_1 leave x to reach backtrack target node b, re-traversing C. While at the backtrack target node b, r_1 then updates $r_1.phase \leftarrow explore$, $r_b.recentPort \leftarrow r_b.recentPort + 1$ and all unsettled robots leave b through port $r_b.recentPort$, where r_b is the robot settled at the backtrack target node b.

Algorithm 2: DFS algorithm.

1 **Input:** $k \leq n$ robots on $1 \leq k' \leq k$ nodes of G;
2 **if** r_i *is alone at node* $v \in G$ **then**
3   r_i settles at v writing $r_i.settled \leftarrow 1$ and setting DFS ID i;
4 **else**
5   each robot on v sets their $DFSID$ i the largest ID among the robots on v and perform DFS i (Algorithm 1);
6   **if** *DFS i meets DFS j* **then**
7   **if** $i < j$ **then**
8   all unsettled robots of DFS i traverse DFS j to reach its head node, $head(j)$, to continue the traversal of DFS j;
9   **else**
10   DFS i continues its traversal, erasing the values set in all the variables by DFS j and writing all the variables based on DFS i;

5.2. General algorithm

We now discuss the general case. Suppose initially robots are positioned on $1 < k' < k$ nodes of G. At round 1, a single robot on a node, if any, settles at that node and the nodes with multiple robots initiate parallel DFSs as in the rooted case (Lines 2–5 of Algorithm 2). The node from which a DFS i starts is called the root of the DFS i and denoted as $root(i)$. If a parallel DFS does not meet another until all robots disperse, we are done. The challenge is how to deal with a traversal meeting another.

A DFS i *meets* DFS j if the robots with DFS ID i arrive at a node x where a robot from DFS ID j is settled. (If robots from DFS ID i and DFS ID j arrive at a node where there is no settled robot, the robot from the DFS with the higher ID settles in that round and the lower ID DFS is said to meet higher ID DFS.) If $i > j$, then we call DFS i *subsuming* otherwise *subsumed*. The *head* of DFS i, denoted as $head(i)$, is the node where the unsettled robots (if any) of that DFS are currently located at (except the robot that is responsible of executing the $retrace1$ and $retrace2$ phases), or else it is the node where the last robot of that DFS settled. That is, while under the $explore$ phase, $head(i)$ has all unsettled

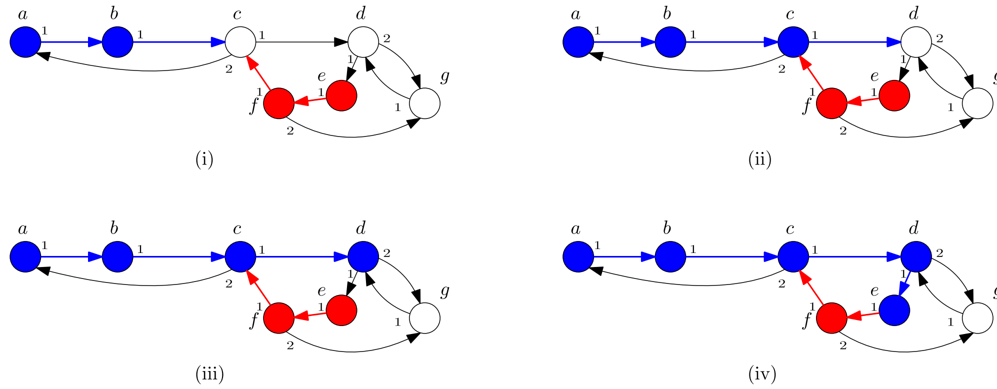


Fig. 6. Meeting of two DFSs colored, blue and red. The blue DFS has higher ID than the red DFS. In the figure, (i) both DFS meet at node c; (ii) blue DFS subsumes red DFS; (iii) blue DFS continues exploration as usual, and the robots from red DFS follow blue DFS; (iv) when blue DFS reaches node e with already settled robot from red DFS, it overwrites the DFS ID to make it part of blue DFS.

robots, however while under $retrace1$ and $retrace2$ phases, the cycle closing node c acts as $head(i)$ and all unsettled robots except the robot performing $retrace1$ and $retrace2$ phases are on c. A DFS is assigned ID i as follows. Suppose a DFS starts initially from a node v_i with a set of $\mathcal{R}(v_i)$ robots. The highest ID robot $r_h \in \mathcal{R}(v_i)$ serves as ID i. DFS i meeting DFS j at a junction node x is handled as follows (Lines 6–10 of Algorithm 2):

DFS i is subsuming, i.e., $(i > j)$: DFS i continues its traversal. For each settled robot w belonging to DFS j it visits, it erases the values set in all the variables, except in $w.settled$. It then sets all the variables based on DFS i and continues the traversal.

DFS i is subsumed, i.e., $(i < j)$: All unsettled robots of DFS i traverse DFS j to reach node $head(j)$. DFS j then continues the traversal from $head(j)$.

In Fig. 6, we show the meeting of two DFSs. The blue DFS has higher ID and subsumes the red DFS and continues its exploration while overwriting the settled nodes of red DFS if it encounters them.

5.3. Analysis of the algorithm

We begin with analyzing the rooted case. We will then analyze the general case building upon the ideas developed for the rooted case. We need Definitions 2 and 3.

Definition 2 *(Rooted spanning tree).* Consider a node v on a strongly connected directed graph $G = (V, E)$. A spanning tree $\mathcal{T} = (V', E')$ rooted at v is a directed subgraph of G such that $|V| = |V'|$, each node has exactly one outgoing edge except v which has no outgoing edge (i.e., $E' \subseteq E$ with $|E'| = |V| - 1$), and all edges are directed toward v.

Definition 3 *(Root path).* Consider $\mathcal{T}$ in Definition 2. A *root path* $RootPath(v', v)$ is a directed path from node $v' \neq v$ to the root v. For each node $w \neq v$, $\mathcal{T}$ contains such a path.

Next, we describe the time complexity when $\mathcal{T}$ is known, and later show how it can be built on the fly. Note that having a known $\mathcal{T}$, no robot gets lost or stuck in a directed graph when a robot reaches a node with all outgoing ports already traversed. In that case, the robot can always simply go back to the root, following $\mathcal{T}$. Since in the undirected case a robot can always move in the reverse direction of the explored edges, but in a directed graph, that is impossible. Having $\mathcal{T}$, it is easier to analyze the time complexity since each root path is at most as long as k nodes, which bounds the cost of backtracking (as we will see in the following section). We split the analysis into the known and unknown case for simplicity.

Time Complexity, Rooted Case, Known $\mathcal{T}$. Let DFS starts from node v which becomes the root. Suppose DFS arrives at node v' for the first time at some round $t > 1$. At that time, v' is marked *visited*. Let v' be the k'-th in the order of the nodes visited by DFS for the very first time. Node v' is assigned $rank(v') := k'$. Initially, v' was marked $null$ (meaning unvisited); once DFS reaches v', it is marked $visited$; and v' will be marked *fully-visited* when DFS visits all the outgoing ports of v'. Therefore, after marked visited, v' will never be marked $null$ (unvisited), and after marked fully-visited, it will never be marked visited or $null$ (unvisited).

Definition 4 *(Active edge).* The edge associated with port $1 \leq port^{v'}_{out} \leq \delta^{v'}_{out}$ of node v' is called *active* if v' is marked *visited* and DFS has exited v' recently through $port^{v'}_{out}$.

Definition 5 *(Active path).* An *active path* $ActivePath(v)$ is a directed path that connects nodes marked *visited* through active edges starting from the root v of DFS.

Lemma 1. *The $RootPath(v', v)$ in $\mathcal{T}$ from any node $v' \neq v$ to the root v always intersects $ActivePath(v)$.*

Proof. $ActivePath(v)$ starts at root v. $RootPath(v', v)$ starts at $v' \neq v$ and ends at v. Therefore, if a node $v'' \neq v$ that is in $ActivePath(v)$ is also in $RootPath(v', v)$, then $RootPath(v', v)$ intersects $ActivePath(v)$ at v''. Otherwise, v is in both $RootPath(v', v)$ and $ActivePath(v)$ and the intersection must happen at v. □

Lemma 2. *If $ActivePath(v)$ ends at a visited node, then the last edge in it closes a cycle of active edges.*

Proof. $ActivePath(v)$ is a collection of active edges. Therefore, if $ActivePath(v)$ ends at a visited node v', then all the edges from v up to v' must be active edges. □

Lemma 3. *Let $e \to f$ be the last edge in Lemma 2. We have that $rank(f) < rank(e)$.*

Proof. Since the directed edge $e \to f$ closes a cycle of active edges, $DFS(v)$ must have visited f before e and hence f must have appeared in $ActivePath(v)$ before e. According to the algorithm, for node α visited before β, $DFS(v)$ assigns rank smaller than the rank of β. Therefore, $rank(f) < rank(e)$. □

Definition 6 *(Backtrack target node).* Backtrack target is always on $ActivePath(v)$. Furthermore, let $V(ActivePath(v))$ be the set of nodes in $ActivePath(v)$ such that, for each node $v' \in V(ActivePath(v))$, the

port number leading to its active edge is smaller than its degree $\delta_{out}^{v'}$. Node $v' \in V(ActivePath(v))$ is called *backtrack target node* if it is the highest ranked node in $V(ActivePath(v))$.

Lemma 4. *Backtracking is required when $ActivePath(v)$ points to a visited or fully-visited node.*

Proof. Suppose the last edge in $ActivePath(v)$ is $e \rightarrow f$. If f is not a $visited$ node or a $fully\text{-}visited$ node (meaning that it is an unvisited node), $DFS(v)$ can continue with the explore phase settling a robot on f and adding a new active edge on $ActivePath(v)$. Therefore, for $DFS(v)$ not being able to continue the explore phase from f, f must be either a $visited$ node or a $fully\text{-}visited$ node. Therefore, backtracking is required to reach to a node from which the explore phase can be continued. □

Definition 7 *(Backtrack path).* For an active edge $e \rightarrow f$, the backtrack path $BacktrackPath(f)$ from node f is defined as follows:

- If f is a fully-visited node: $BacktrackPath(f)$ consists of two segments:
  - $RootPath(f, v)$ from f up to the first intersection node c with $ActivePath(v)$.
  - The portion of $ActivePath(v)$ from c to the backtrack target node b (as defined in Definition 6).
- If f is a visited node: $BacktrackPath(f)$ is simply the segment of $ActivePath(v)$ from f to the backtrack target node b.

Lemma 5 *(Cycle closing node). If $ActivePath(v)$ ends at a visited node v', v' becomes a cycle closing node. If $ActivePath(v)$ ends at a fully-visited node v'', let c be the intersection node of $ActivePath(v)$ and $RootPath(v'', v)$. Node c becomes a cycle closing node.*

Proof. If $ActivePath(v)$ ends at a $visited$ node v', v' must appear in two edges $v' \rightarrow f$ and $f' \rightarrow v'$. Additionally, by Lemma 2, $v' \rightarrow f$ must have appeared in $ActivePath(v)$ before $f' \rightarrow v'$ and edge $f' \rightarrow v$ closes a cycle of active edges. Therefore, since v' is in $ActivePath(v)$, v' can be a cycle closing node.

If $ActivePath(v)$ ends at a $fully\text{-}visited$ node v', v' appears only in edge $f' \rightarrow v'$. In this case, take the $RootPath(f', v)$ until the first node c' is found with status $visited$. By Lemma 1, node c' must be on $ActivePath(v)$. Therefore, c' can be a cycle closing node. □

Lemma 6. *Algorithm 1 solves* Dispersion *for any rooted initial configuration of $k \le n$ robots in $O(k^2 \cdot \Delta_{out})$ rounds, given directed rooted spanning tree $\mathcal{T}$.*

Proof. We first prove that Algorithm 1 solves Dispersion correctly. Notice that, at any time, the cycle closing node as well as the backtrack target node are on $ActivePath(v)$. As established in Lemma 5, the cycle closing node is determined based on whether $ActivePath(v)$ ends at a visited or fully-visited node and by Lemma 4, backtracking is required to continue exploration. Moreover, the backtrack target node b is the farthest node from root v in $ActivePath(v)$ with at least one unvisited port left. This property resembles DFS backtrack in undirected graph case. Additionally, the backtrack target node can always be found and reached. Let c be a cycle closing node and f' be a backtrack target node. It is the case that, if f' is in the cycle and $f' \neq c$, $rank(f') > rank(c)$, otherwise c is both the cycle closing node as well as a backtrack target node (ref. Lemma 3). If f' is not in the cycle, then $rootPort$ set is taken from c which will take the traversal to a cycle closing node $c' \neq c$ with $rank(c') < rank(c)$ and the process of finding a backtrack target node f' starts from there. Furthermore, for any edge $e \rightarrow f$ on $ActivePath(v)$, e cannot become $fully\text{-}visited$ before f. These properties are sufficient to show that the DFS executes correctly, solving Dispersion.

We now prove the time bound. Since there are $k \le n$ robots, any $Backtrack(.)$ path is of length at most k as it visits only the nodes with a robot already settled on each of them. DFS needs to traverse at most $k \cdot \Delta_{out}$ edges before all k robots settle at k different nodes. For each outgoing edge, backtracking is needed at most once. Since backtracking is executed in two phases $retrace1$ and $retrace2$, the total number of rounds to finish these phases is $\le 2 \cdot k$. In $retrace1$, if DFS visits a $fully\text{-}visited$ node, traversing the $RootPath(.,.)$ is required to find a $visited$ node to make it a cycle closing node. This length is $< k$. In $retrace2$, traversing to the backtrack target node takes additional $< k$ rounds. Therefore, each backtracking finishes in $< 4 \cdot k$ rounds. Therefore, Algorithm 1 needs $< (4 \cdot k) \cdot (k \cdot \Delta_{out}) = O(k^2 \cdot \Delta_{out})$ rounds. □

Time Complexity, Rooted Case, Unknown $\mathcal{T}$. We now remove the assumption of known $\mathcal{T}$ from Lemma 6 and construct it on-the-fly. We start with the following lemmas.

Lemma 7. *For any two $visited$ nodes v', v'' that are eligible to become a backtrack target node, if $rank(v'') < rank(v')$, then v' becomes $fully\text{-}visited$ before v''.*

Proof. Marking nodes with status $fully\text{-}visited$ happens in the $retrace2$ phase. Phase $retrace2$ starts from and ends at a cycle closing node c which is on $ActivePath(v)$. The cycle closing node c has the lowest rank among the nodes in the cycle and the other nodes in the cycle have ranks in the increasing order (not necessarily consecutive). The backtracking either takes to the node in the cycle that is farthest from c from which the $explore$ phase can be continued or if no such node exists in the cycle, the backtracking marks all the nodes in the cycle including c $fully\text{-}visited$. Therefore, the nodes in the cycle, if eligible to be a backtrack target node, become $fully\text{-}visited$ starting from the highest ranked in the cycle to the lowest ranked. □

Lemma 8. *When node v' becomes $fully\text{-}visited$, $RootPath(v', v)$ ends at the smallest ranked node in $ActivePath(v)$ reachable from v'.*

Proof. We have that the nodes in $ActivePath(v)$ are assigned ranks in an increasing order. Furthermore, by Lemma 7, the nodes start to become $fully\text{-}visited$ in $ActivePath(v)$ starting from the highest ranked node. Let c be a cycle closing node in $ActivePath(v)$. If any node after c in $ActivePath(v)$ closes a cycle with a node c' such that $rank(c') < rank(c)$ then in the $retrace1$ and $retrace2$ phases executed from c', the $rootPort$ information at each node in the cycle is updated to point to c'. An example illustration is given in Fig. 5, where the root path is updated to point to node c that has smaller rank then node d. If there is no node c' with $rank(c') < rank(c)$, the root path to c will not be modified. □

During the execution of DFS, $\mathcal{T}$ may be a forest of many disjoint directed rooted trees. Let $\mathcal{F}$ be the *directed rooted forest* defined as a collection of the disjoint directed rooted trees $\mathcal{T}_1, \ldots, \mathcal{T}_i$.

Lemma 9. *The collection of $RootPath(v', v)$ of the fully-visited nodes v' form a directed rooted forest $\mathcal{F}$.*

Proof. We prove this lemma using induction. This lemma holds before any node becomes $fully\text{-}visited$. This lemma also holds when a node v' becomes $fully\text{-}visited$ because there is only one outgoing edge in the root path leading either to a $visited$ node or to the root. Given that this lemma holds from the beginning of the algorithm until v' becomes $fully\text{-}visited$, we prove that it also holds after v' becomes $fully\text{-}visited$. If $v' = v$, then there is no edge in its root path, i.e., it does not select any edge in the root path edge. Assume that $v' \neq v$. When v' becomes $fully\text{-}visited$, there is at least one $visited$ node (the root v).

Assume for the contradiction that when v' becomes $fully\text{-}visited$, there is no $visited$ node. Let $V_{prev,v'}$ be the set of nodes that were visited before v' was $visited$ the first time. Let $V_{aft,v'}$ be the set of nodes,

including v', that were $visited$ for the first time after v and before v' becomes $fully\text{-}visited$. Let $E(aft, prev)$ be the set of directed edges from a node in the set $V_{aft,v'}$ to a node in the set $V_{prev,v'}$. Since G is strongly connected, we have that $E(aft, prev) \neq \emptyset$. Furthermore, before v' becomes $fully\text{-}visited$, there is no edge from a node in the set $V_{prev,v'}$ to a node in the set $V_{after,v'}$ except the active edge. Therefore, all the rooted trees $\mathcal{T}_i$ in the set $V_{prev,v'}$ are rooted at the $visited$ nodes in $V_{prev,v'}$. Consider each root path $RootPath(x, v)$ going through each edge $e \in E(aft, prev)$. Let $a \to b$ the edge in $E(aft, prev)$ whose associated root path $RootPath(x, v)$ is the last among the root paths to pass over v' before it becomes $fully\text{-}visited$.

The edge $a \to b$ was traversed when a was still active. All the directed edges from v' to b must have been included in $RootPath(x, v)$. Since no node in $V_{prev,v'}$ becomes $fully\text{-}visited$ after v' is $visited$ and before v' becomes $fully\text{-}visited$, no $rootPort$ information on a node in $V_{prev,v'}$ is changed at that period. Since $RootPath(x, v)$ is the last among the root paths, the $rootPort$ information in $RootPath(x, v)$ does not change before v' becomes $fully\text{-}visited$. Therefore, the $rootPort$ selected by v' when it becomes $fully\text{-}visited$ cannot close a cycle of $fully\text{-}visited$ nodes. Hence, a contradiction. □

Lemma 10. *For any fully-visited node v', $RootPath(v', v)$ of length $\leq k-1$ can be constructed that leads either to a visited node v'' or to root v with $rank(v) < rank(v'') < rank(v')$.*

Proof. Since G is strongly connected, a path as required exists. Consider a backtracking path $BacktrackPath(v')$ for node $v' \neq v$ in Algorithm 2. Each such path consists of zero or more edges in $RootPath(v', v)$ and zero or more edges in $ActivePath(v)$. Suppose $BacktrackPath(v')$ goes over an active edge $e \to f$ for the very first time, then, with respect to f, this path leads to a node, say d, that was visited by DFS before f, i.e., $rank(d) < rank(f)$. Therefore, the active edges along the cycle from f up to the cycle closing node should be marked as root edges.

The root v has the smallest rank and if there exists a visited node v'' it must be on $ActivePath(v)$ and hence $rank(v) < rank(v'') < rank(v')$, where v' is a $fully\text{-}visited$ node (ref. Lemma 8). From Lemma 9, since the collection of $RootPath(v', v)$ from any node $v' \neq v$ form a directed root forest, $RootPath(v', v)$ does not visit any node twice. This immediately proves that the length of any $RootPath(v', v) < k$ since there are only at most k nodes (robots) in the forest. □

Lemma 11. *Algorithm 1 solves* Dispersion *for any rooted initial configuration of $k \leq n$ robots in $O(k^2 \cdot \Delta_{out})$ rounds, constructing $\mathcal{T}$ on-the-fly.*

Proof. We have from Lemma 10 that the length of $RootPath(v', v)$ from any node $v' \neq v$ to root v is $< k$. Moreover, $RootPath(v', v)$ ends at a $visited$ node. Therefore, a backtrack finishes in $O(k)$ rounds. Furthermore, an edge needs at most one backtrack and there are $k \cdot \Delta_{out}$ edges. Hence the proof. □

Time Complexity, General Case, Unknown $\mathcal{T}$. Since robots are on $1 \leq k' < k$ nodes in any general initial configuration, there will be k' parallel DFSs initiated at round $t = 1$. DFS i does not meet any other DFS j at $t = 1$. If no two DFSs meet until all robots settle, the analysis for the rooted case applies. We analyze here DFS i meeting DFS j and show that $O(k^2 \cdot \Delta_{out})$ time bound can be established. Consider DFS i at some round $t > 1$. Consider the nodes of G that are occupied with robots belonging to DFS i.

Lemma 12. *Let $x \neq head(i)$ be a node of G belonging to DFS i. $head(i)$ is always reachable from x.*

Proof. $head(i)$ is always a $visited$ node (but not the $fully\text{-}visited$ or unvisited node). We first show that $head(i)$ is reachable from $root(i)$. Consider first the situation of $ActivePath(root(i))$ ending at a $visited$ node y. In this case, $head(i)$ is the node y and y is reachable from $root(i)$ traversing $ActivePath(root(i))$. Consider the situation of $ActivePath(root(i))$ ending at a $fully\text{-}visited$ node y. In this case, $head(i)$ is the first $visited$ node z in $RootPath(y, root(i))$. Since z is a $visited$ node, it must be in $ActivePath(root(i))$. Therefore, z can be visited from $root(i)$ traversing $ActivePath(root(i))$.

We now show that $head(i)$ is reachable from any node $w \neq root(i)$ that belongs to DFS i. Node w is either a $visited$ node or a $fully\text{-}visited$ node. If w is a $visited$ node, it must be in $ActivePath(root(i))$. If w is between $root(i)$ and $head(i)$ in $ActivePath(root(i))$, it reaches $head(i)$ following the $ActivePath(root(i))$. If w is after $head(i)$ in $ActivePath(root(i))$, it can follow its $RootPath(w, root(i))$ until reaching $head(i)$. If w is a $fully\text{-}visited$ node, it can follow its $RootPath(w, root(i))$ until it finds the first $visited$ node (i.e., $RootPath(w, root(i))$ intersects $ActivePath(root(i))$) and then follow $ActivePath(root(i))$ to reach $head(i)$. Thus, $head(i)$ is always reachable from any node $x \neq head(i)$ belonging to DFS i. □

Lemma 13. *If k_i robots belong to DFS i, then $head(i)$ is at distance $\leq k_i - 1$ from any node x in DFS i.*

Proof. The sum of the lengths of $ActivePath(root(i))$ and $RootPath(w, root(i)), w \neq root(i)$, is $\leq k_i - 1$, since they do not visit the same node of DFS i twice. Therefore, in all the cases of Lemma 12, the path length cannot be more than $k_i - 1$ to reach $head(i)$ of DFS i from any node $w \neq head(i)$ in DFS i. □

Since robots have unique IDs, $i \neq j$ for any two DFSs i and j. Suppose DFS i meets DFS j. DFS i is either subsuming ($i > j$) or subsumed ($i < j$). Due to the k' DFSs initiated in parallel, a DFS j may be met by different other DFSs, and DFS j may in turn meet another DFS concurrently. Further, transitive chains of such meetings can occur concurrently. This leads us to formalize a notion of a *meeting graph*.

Definition 8 *(Meeting graph)*. The meeting graph $G_M = (V_M, E_M)$, where V_M is the DFS IDs and there is a directed edge in E_M from i and j if DFS i meets DFS j.

Nodes in V_M have an arbitrary in-degree but out-degree 1. Moreover, G_M may be composed of multiple connected components. Furthermore, there may be cycles in connected components. We focus on a single connected component C_M in G_M; other components of G_M can be dealt analogously. At round 1, G_M has k' nodes and $E_M = \emptyset$, which are components by themselves. There are multiplicity nodes in at least one component. At round 2 or after, the number of components monotonically decrease and when Dispersion is achieved, $E_M = \emptyset$ and no multiplicity node in each component. We have the following observation.

Observation 1. *A connected component C_M in graph G_M never splits into multiple sub-components.*

Suppose there are γ nodes (i.e., DFSs) in C_M. We slightly abuse the notation to represent the DFS IDs in the increasing order as DFS 1, DFS 2, …, DFS γ. Note that the ID of DFS γ would be the highest ID DFS in C_M. This DFS γ subsumes all other $\gamma - 1$ DFSs in that component. During this process, the number of settled robots monotonically increase. Therefore, C_M never disconnects to form multiple sub-components. Furthermore, C_M may meet another component and they become a single component. When components merge, the total number of settled robots may increase or remain the same, but the number of settled robots in the resulting component increases. This leads us to formalize a notion of a meeting tree for the meeting graph G_M.

Definition 9 *(Meeting tree)*. The meeting tree T_M is constructed as follows:

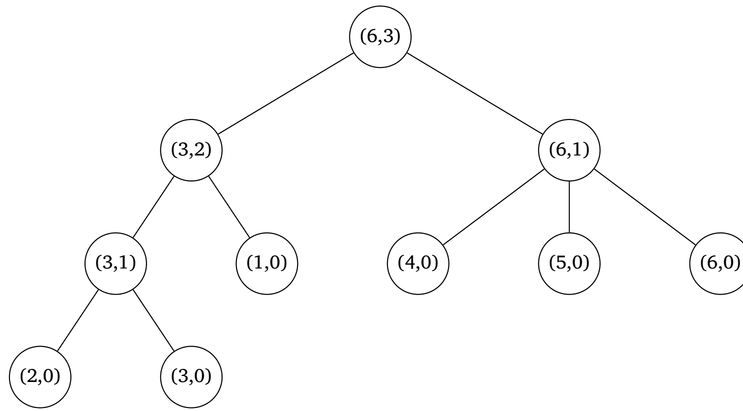


Fig. 7. An example of meeting tree T_M starting with 6 DFSs.

- Initial state: The k' initial DFSs form the leaf nodes of T_M at level 0, denoted as $(i,0)$ for each DFS i.
- Node creation: When α DFSs meet in a component of the meeting graph G_M:
  - Let the meeting DFSs be represented by nodes (a_i, h_i) for $i \in [1,\alpha]$
  - A new node (γ, h) is created in T_M as the parent of these α nodes, where:
    - γ is the highest DFS ID among the α meeting DFSs
    - $h = 1 + \max_{i\in[1,\alpha]} h_i$
- This process continues recursively as more DFSs meet.

In Fig. 7, we show an example illustrating mergers of six different DFSs and their corresponding representation in the meeting tree. When two or more DFSs (and components) meet, one of the DFSs in the formed component subsumes all other DFSs. Therefore, there is a DFS which appears in each level of the meeting tree T_M starting from level 0 upto the highest level in that component. Furthermore, the height h of T_M is $0 \le h \le k'-1$. The maximum height $k'-1$ represents the sequential meeting of DFS IDs. The height $h < k'-1$ represents meetings and subsumptions that happen in parallel across different components. Therefore, it will sufficient to bound termination time for sequential cases since it immediately subsumes the time bounds for all other (parallel) cases. The cases we consider are as follows:

(I) DFS l meets DFS $l-1$, $2 \le l \le k'$ (meeting in decreasing order of the DFS IDs),
(II) DFS l meets DFS $l+1$, $1 \le l \le k'-1$ (meeting in increasing order of the DFS IDs), and
(III) any combination of cases (I) and (II) (meeting sometime in increasing order and sometime in decreasing order of the DFS IDs).

We now discuss the time bound to achieve Dispersion. Consider node $head(i)$ of DFS i. Consider the situation of $head(i)$ not having any unsettled robot, i.e., all robots belonging to DFS i have been settled. Let the smallest round on which $head(i)$ not having any unsettled robot be t_i. Let DFS j meets DFS i and get subsumed by DFS i ($j < i$). The unsettled robots of DFS j need to move to $head(i)$ for DFS i to continue its traversal. Let t_j be the round at which the unsettled robots of DFS j reach $head(i)$. The difference $t_j - t_i$ is the wait time for $head(i)$. The total wait time $totalWait(head(i))$ for a DFS i in a component C_M is the sum of the wait times of the $head(i)$ before either it is subsumed or Dispersion is solved.

Lemma 14. *Consider the highest ID DFS γ in component C_M. $totalWait(head(M)) \le (k_M - 1)\cdot(\gamma - 1)$ for either DFS γ to be subsumed or Dispersion is solved, where k_M is the number of robots in C_M and γ is the number of DFSs in C_M.*

Proof. We prove this lemma considering Cases (I) – (III). Consider Case (I) in which the highest ID DFS γ in C_M first meets DFS $\gamma-1$, after subsuming DFS $\gamma-1$, its meets DFS $\gamma-2$, and so on. This decreasing DFS ID sequence ends when DFS γ meets DFS 1 after subsuming DFS 2. Since DFS γ always meets lower ID DFS $j<\gamma$, it continues its traversal without any wait on $head(\gamma)$. Therefore, for Case (I), $totalWait(head(\gamma)) = 0$.

Consider Case (II) in which DFS 1 first meets DFS 2, after subsumed to DFS 2, DFS 2 meets DFS 3, and so on. This increasing DFS ID sequence ends when DFS $\gamma-1$ is subsumed to the highest ID DFS γ in C_M and DFS γ finishes its traversal. After DFS 1 meets DFS 2, from Lemma 13, the unsettled robots of DFS 1 can reach $head(2)$ in next $\le k_M - 1$ rounds, since there are k_M robots in C_M and only at most k_M robots belong to a DFS ID. After subsuming DFS 1 and meeting DFS 3, the unsettled robots of DFS 2 can reach $head(3)$ in next $\le k_M - 1$ rounds. Arguing this way, the unsettled robots of DFS $\gamma-1$ reach $head(\gamma)$ in next $\le k_M - 1$ rounds after DFS $\gamma-1$ meets DFS γ. Therefore, $totalWait(head(\gamma)) \le (k_M - 1)\cdot \gamma - 1$, adding the times to move the unsettled robots from $\gamma - 1$ pairs of $head(j)$, $head(j+1)$.

Consider now Case (III) of sometime DFS i meeting DFS $j<i$ and sometime meeting DFS $j>i$. When DFS i meets DFS $j<i$, there is no wait time and when DFS i meets DFS $j>i$, then the time to move unsettled robots of DFS i to $head(j)$ is $k_M - 1$ rounds. Therefore, $totalWait(head(M)) \le (k_M - 1)\cdot(\gamma-1)$. The lemma follows. □

Lemma 15. *Consider the highest ID DFS γ in component C_M. Either DFS γ is subsumed or robots of C_M disperse to k_M nodes in $O((k_M)^2 \cdot \Delta_{out} + k_M \cdot \gamma)$ rounds, where k_M is the number of robots in C_M and γ is the number of DFSs in C_M.*

Proof. Consider the rooted case of all k_M robots on a single node. The robots disperse to k_M nodes in $O((k_M)^2 \cdot \Delta_{out})$ rounds. Now consider a component C_M of the meeting graph G_M. Suppose C_M is not subsumed by any other DFS (component). We have that from Lemma 14 that the total wait time for $head(M)$ of the highest ID DFS M in C_M is $totalWait(head(\gamma)) \le (k_M - 1)\cdot(\gamma-1)$ rounds. Except the total wait time, the DFS traversal of DFS γ is as in the rooted case (of no waiting at head nodes) which finishes in $O((k_M)^2 \cdot \Delta_{out})$ rounds. Therefore, the total number of rounds until k_M robots disperse to k_M different nodes is $O((k_M)^2\cdot\Delta_{out}) + totalWait(head(M)) = O((k_M)^2\cdot\Delta_{out} + k_M\cdot\gamma)$ rounds. □

Lemma 16. *Algorithm 2 solves Dispersion deterministically for any general initial configuration of $k \le n$ robots in $O(k^2 \cdot \Delta_{out})$ rounds.*

Proof. We first argue that for any DFS j in a component C_M, $totalWait(head(j)) \le totalWait(head(\gamma))$. Since DFS γ is the highest ID DFS in C_M of the meeting graph G_M, it never gets subsumed by

any other DFS j in C_M. Therefore, consider Case (I) of DFS γ always meeting DFS $l, l < M$. In this case, DFS γ has to subsume each DFS met. Since DFS γ continues its traversal even after meeting j, $totalWait(head(\gamma)) \geq totalWait(head(j))$. Consider now the Case (II) of DFS l meeting DFS $l+1, 1 \leq l \leq \gamma - 1$. In this case, DFS γ has to continue its traversal even after DFS $\gamma - 1$ reaches $head(\gamma)$, i.e., $totalWait(head(\gamma)) > totalWait(head(j))$. Finally, for Case (III), since DFS M never gets subsumed, $totalWait(head(\gamma)) \geq totalWait(head(j))$ for any other DFS $j \neq \gamma$.

We now prove time bound. Consider the meeting tree T_M of the meeting graph G_M. Each node in T_M is a component (at level 0, a DFS itself is a component) and at the end either there is a single root node in each component tree formed. Therefore, the height of the meeting tree T_M can be at most $\leq k' - 1$ since there are only k' DFS traversals. The total wait time $totalWait(head(\gamma_{\max}))$ of the largest ID DFS $\gamma_{\max}$ is bounded by $k \cdot k' < k^2$ (ref. Lemma 15). Except the wait time, for k robots, DFS finishes in $O(k^2 \cdot \Delta_{out})$ rounds. Therefore, the total time is $O(k^2 \cdot \Delta_{out}) + totalWait(head(M_{\max})) = O(k^2 \cdot \Delta_{out} + k^2) = O(k^2 \cdot \Delta_{out})$ rounds. □

6. BFS dispersion algorithm

We now design a deterministic BFS traversal based algorithm that exploits 1-hop communication and achieves $O(k \cdot D)$ time bound. The pseudocode of the algorithm is given in Algorithm 3. We note that any deterministic DFS algorithm needs $\Omega(k^2)$ time for DISPERSION even in the 1-hop model, i.e., 1-hop model is of no benefit for DFS traversal based algorithms.

Suppose initially k robots are on $1 \leq k' < k$ nodes. Denote the node set by $V_{k'}$. Let $k'' \leq k'$ be the multiplicity nodes, i.e., two or more robots positioned on them. Denote the node set by $V_{k''} \subseteq V_{k'}$. Each node $v \in V_{k''}$ is called a *source node*. Let $w \in V_{k'}$ be a node such that it has at least one empty out-going neighbor. Node w is called a *request node*. Let V_R be the set of request nodes. A source node may also be a request node if it has empty out-going neighbor(s).

The goal in the algorithm is to find a matching between a source node and a request node so that the empty (out-going) neighbors of the request node can be occupied by robots. We exploit 1-hop communication to find such matching in $O(D)$ rounds. Since there are $k \leq n$ robots in total, the algorithm can finish settling all the robots at k different nodes in $O(k \cdot D)$ rounds. Consider a source node $v \in V_{k''}$. The highest ID robot among the robots $R(v)$ on v settles at v. We have two cases.

Case 1 – v is a source as well as a request node: Let $d(v) \leq \delta^v_{out}$ be the number of empty out-going neighbors of v. Node v sends $\lfloor (|\mathcal{R}(v)| - 1)/d(v) \rfloor$ robots to $d(v) - 1$ empty out-going neighbors and $(|\mathcal{R}(v)|-1)/d(v) + (|\mathcal{R}(v)|-1) \mod d(v)$ robots to one empty out-going neighbor.

Case 2 – v is a source node but not a request node: The matching is done in three stages, Stages 1–3. Case 2 starts in a round, when v becomes a source node but not a request node. In Stage 1, v sends a *broadcast message* to all its (non-empty) out-going neighbors and the (non-empty) out-going neighbors forward the message to their (non-empty) out-going neighbors, and so on. This message broadcast forms a directed BFS tree $BT(v)$ with v becomes the root and the request nodes become the leaves. As soon as a request node w receives a broadcast message Stage 2 starts in which node w sends a *request message* following the path in $BT(v)$ to reach the source node v. Each internal node u in $BT(v)$ stores a neighbor node from which it received a broadcast message as the parent of u in Stage 1 and this information helps in forwarding the request message to the root. In Stage 3, the source node v sends $|\mathcal{R}(v)| - 1$ robots in the set $\mathcal{R}(v)$ to the request node from which it receives the first request message. At the end of Stage 3, a request node becomes a source as well as a request node and Case 1 applies.

We describe below separately some more details on Stages 1–3 of Case 2 of the BFS algorithm.

Algorithm 3: BFS algorithm for robot r_i at node $v_i \in G$ at some round $t \geq 1$.

```
1  if r_i is alone at node v_i then
2  |  r_i settles at v_i writing r_i.settled ← 1;
3  if r_i is not alone at v_i and v_i is a source node (no empty outgoing neighbors) then
4  |  if v_i has no settled robot and r_i has the highest ID then
5  |  |  r_i settles at v_i writing r_i.settled ← 1;
6  |  send broadcast(t, r_i.ID) message to all its outgoing neighbors;
7  if r_i is not alone at v_i and v_i is a request node (empty outgoing neighbor(s)) then
8  |  if v_i has no settled robot and r_i has the highest ID then
9  |  |  r_i settles at v_i writing r_i.settled ← 1;
10 |  divide almost equally and send all unsettled robots at v_i to its empty
   |    neighbors;
11 if r_i is alone at v_i and receives a Broadast(.,) message then
12 |  if r_i not a request node then
13 |  |  r_i forwards it to all its outgoing neighbors if its not duplicate;
14 |  else
15 |  |  r_i sends Request(t, r_i.ID) to neighbor from which it received
   |  |    Broadcast(.);
16 if r_i receives a Request(.,) message then
17 |  if r_i not a source node then
18 |  |  r_i sends to neighbor from which it received Broadcast(.) message;
19 |  else
20 |  |  r_i sends all the unsettled robots to the request node;
```

- **Stage 1:** Source node v sends its broadcast message as a tuple $(roundNo, robotID)$, where $roundNo$ is the round at which the message was created and $robotID$ is the highest ID robot in the set $\mathcal{R}(v)$ of robots positioned at node v. Internal nodes (robots serving as nodes) in the BFS tree $BT(v)$ keep this tuple in their memory so that this information will be useful in Stage 2 to send request messages back to v. During Stage 1, if there are α source nodes satisfying Case 2, then α BFS trees will be built with α source nodes serving as their roots.
- **Stage 2:** As soon as a request node w receives a broadcast message, it will send a request message in response to that broadcast message. The request message is sent to the source node from which the broadcast message was received. If multiple broadcast messages were received, then the highest ranked message in the lexicographical order is picked. The request message is also a tuple $(roundNo, robotID)$ such that $roundNo$ denotes the round at which the message was generated and $robotID$ is the ID of the robot present at the requesting node.
- **Stage 3:** The source node v may receive two or more request messages at the same round. In this case, it will again pick the highest ranked request message and the robots are sent to that request node following the path in the BFS tree $BT(v)$ from which the request message came to the source.

Fig. 8 illustrates the directed BFS tree $BT(S)$ formed during Stage 1 by broadcast messages. It also shows how a request message from a request node R reaches S in Stage 2, and finally the robots from S follow the directed path $p(S, R)$ to reach R from S.

Theorem 7. *To solve DISPERSION of $k \leq n$ robots on a n-node strongly-connected directed graph, any deterministic DFS algorithm needs $\Omega(k^2)$ time under the 1-hop communication model.*

Proof. Consider the same graph G as in Fig. 1 and the rooted initial configuration of all k robots initially at node u. Starting from u, traversing the upper arc up to v cannot exploit information given by the 1-hop model. From v, even when 1-hop communication tells how many nodes are unvisited, a DFS traversal can only visit one node on the column at a time. After visiting one node, it takes additional $\Omega(k)$ rounds to visit another. Therefore, a DFS algorithm needs $\Omega(k^2)$ rounds. □

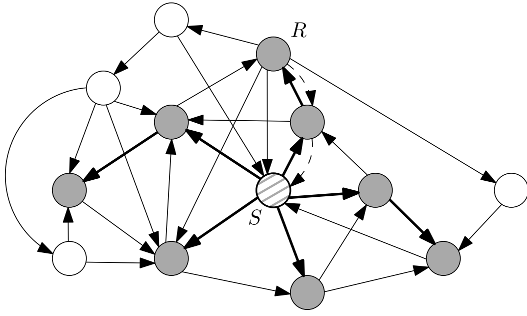


Fig. 8. An illustration of (i) a directed BFS tree $BT(S)$ (bold edges) rooted at a source node S build by the broadcast message sent by S and (ii) a directed path (dashed edges) the request message from request node R traverses in $BT(S)$ to reach S. After the request message reaches S, the robots at S (except settled) follow the directed path $P(S,R)$ to reach R. R then sends those robots to its empty neighbors to settle.

The algorithm terminates when there is no new source node (i.e., no multiplicity node).

Lemma 17. *A request node can be matched with a source node in $O(D)$ rounds.*

Proof. If a source node v is also a request node w as in Case 1, then the matching happens immediately. Therefore, we focus on proving that the matching between a source node v and a request node $w \neq v$ as in Case 2 can be done in $O(D)$ rounds. Particularly, we prove that the length of the (directed) path $P(v,w)$ in the BFS tree $BT(v)$ between a source node v and a request node w in Stage 1 is $\leq D$. Since the request message from w in Stage 2 follows the path $P(v,w)$ in the reverse order, it can then be guaranteed that Stage 2 finishes in $\leq D$ rounds. Stage 3 then moves the robots from source v to request node w following $P(v,w)$, the robots from v move to w in next $\leq D$ rounds. After Stage 3 finishes, the empty neighbors of w will be occupied by robots in a round. Therefore, in total $3D+1 = O(D)$ rounds are sufficient for a request node to be matched with a source node and (at least) one new empty node is occupied.

We now prove that how Stage 1 is done in $\leq D$ rounds. Assume that all n nodes of G are occupied with robots. Since G is a strongly connected directed graph, there is a directed path from a node in G to every other node in G. Suppose a node v' constructs a directed BFS tree $BT(v')$ of outgoing edges using broadcast messages. There will be exactly n nodes in $BT(v')$ and the length of the (directed) path $P(v',w')$ from v' to any other node w' in $BT(v')$ is $\leq D$. Furthermore, path $P(v',w')$ in $BT(v')$ is the shortest directed path from v' to w'.

Suppose now that not all nodes of G are occupied. In fact, since Dispersion is solved when $k \leq n$ nodes are occupied by robots, assume that only $\kappa < k$ nodes are occupied. This implies that there is at least a source node. Let a source node v'' constructs a BFS tree $BT(v'')$ through broadcast messages. Since there are only κ nodes of G occupied, $BT(v'')$ will have κ nodes. It also may be the case that for a node in $w'' \neq v''$ in $BT(v'')$, the length of the (directed) path $P(v'',w'') > D$. Let $P'(v'',w'')$ be the shortest (directed) path between v'' and w'' in $BT(v'')$ when all n nodes were occupied. This means that the broadcast message from v'' could not follow $P'(v'',w'')$ to reach w'' due to the fact that at least a node in the path $P'(v'',w'')$ was empty. Let w''' be the first node closest to v'' in path $P'(v'',w'')$ such that its neighbor y in path $P'(v'',w'')$ is empty. Node w''' must be a request node, since otherwise y must be in $BT(v'')$. Therefore, $P(v'',w''') \leq D$ even when only $\kappa < k$ nodes are occupied and w''' can send its request message following path $P(v'',w''')$ in reverse order to reach v''. When the request from w''' reaches v'' there are two cases: (i) v'' does not win competition as it has to compete with other request messages or (ii) v'' was not the source node anymore. In both the cases, there was another request node z with $P(v'',z) \leq P'(v'',w''') \leq D$ and the matching happens between v'' and z. □

Theorem 8. *Algorithm 3 solves Dispersion deterministically for any general initial configuration of $k \leq n$ robots in $O(k \cdot D)$ rounds under the 1-hop communication model.*

Proof. We have from Lemma 17 that a request node can be matched with a source node in $O(D)$ rounds so that the request node will have at least two or more robots. In the next round, at least one of its empty out-going neighbors is occupied with a robot. The matching process is initiated as soon as a node becomes a new source. Therefore, in every $O(D)$ rounds, the matching between a source node and a request node is completed and at least one empty out-going neighbor is occupied. When there is a single robot on a node, it settles there and never moves in future rounds. If there are multiple robots, then exactly one settles there and never moves after that. Therefore, in every $O(D)$ rounds, the number of unsettled robots decrease by at least 1 and the number of nodes with settled robots increase by at least 1. Since there are in total $k \leq n$ robots, this whole process finishes in $O(k \cdot D)$ rounds. □

7. Extensions to crash faults

In this section, we consider that $f \leq k$ robots may experience crash faults. We describe how the DFS and BFS algorithms of Sections 5 and 6 extend to handle crash faults.

Modified Kwek's Exploration Algorithm. We can add the following the condition for crash-detection to the Kwek's exploration algorithm from Section 4 to handle crash faults. It is necessary to have crash detection, since there is a possibility of creating a parent pointer loop due to the crashed robots and the robots would not be able to explore the graph. Since Kwek's algorithm keeps a list of edges and nodes in the memory of each robot, it is easy to detect crashes. As long as the exploration head traverses an already traversed edge (u,v) such that u has not crashed, it can determine if there is a crash at v if the ID of the settled robot does not match with the one in the memory. The crash mitigation strategy is as follows. As soon as crash is detected, the exploration head starts the algorithm again from scratch, i.e., it deletes all the information about the graph stored in its memory and assumes the current node to be the new root.

Proof. Proof of Theorem 1(a) From Corollary 2, we have $O(dk^2+m)$ round Dispersion algorithm that must be restarted for each fault. For f crashes, the algorithm may run at most f times, and thus the total time required is $O(f \cdot d \cdot k^2)$ rounds. The memory requirement follows from Corollary 2.

DFS Algorithm. We extend Algorithm 2 to handle crash faults as follows. Crashing of an unsettled robot at any time during Algorithm 2 is not a problem. Therefore, crashing of robots does not affect the *explore* phase. However, crashing of settled robots is a problem because a robot performing *retrace*1 and *retrace*2 phases should never encounter an empty node. We have two situations: (i) The robot doing the *retrace*1 and *retrace*2 phases crash or (ii) the robots in the path of the robot doing those phases crash.

We deal with the first situation of the possible crash of the robot doing *retrace*1 and *retrace*2 phases by asking all the unsettled robots to perform those phases (not just one robot). In other words, no robot waits at the cycle closing node. All of them perform the *retrace*1 and *retrace*2 phases and the robot settled at the cycle closing node keeps information that it is a cycle closing node. We deal with the second situation of the robot(s) performing *retrace*1 and *retrace*2 encounter an empty node (must be the case that robots at that node was crashed) by starting a new DFS from that crashed node using $DFSID$ variable as a tuple $(roundNo, RID)$, where $roundNo$ is the round at which the

DFS is started and RID is the highest ID robot i among the robots on that node. This approach is also used when robots move from the cycle closing node to the backtrack target node to continue the *explore* phase. At round 1, all k' DFSs initiated in parallel have $roundNo = 1$ and RID is the highest ID robot on each of k' nodes. In the crash-free case, subsumption among multiple DFSs still happens based on RID since $roundNo$ remains the same (1) for all DFSs throughout the traversal. In other words, $roundNo$ parameter gets value greater than 1 for a DFS if that DFS is started after the encounter of a crashed node (robot). Therefore, a DFS meeting another is handled asking larger $DFSID$ DFS in the lexicographical order to subsume the other.

Lemma 18. *In the same setting of Lemma 16 having $1 \le f \le k$ robot crashes, the DFS algorithm solves* Dispersion *deterministically in $O(f \cdot k^2 \cdot \Delta_{out})$ rounds.*

Proof. If no crash, it is immediate from Lemma 16 that Dispersion finishes in $O(k^2 \cdot \Delta_{out})$ rounds. Since $f \le k$ robots may crash, each of them must crash before all k robots settle. Therefore, a robot must crash within first $O(k^2\Delta_{out})$ rounds, otherwise Dispersion was already achieved. When a new DFS starts, and there is no other crash, it finishes Dispersion in at most next $O(k^2\Delta_{out})$ rounds. Therefore, for $f = 1$, Dispersion finishes in $2 \cdot O(k^2\Delta_{out})$ rounds. If $f = 2$, then the 2nd crash must happen within first $2 \cdot O(k^2\Delta_{out})$ rounds. Arguing this way, for $f > 1$ crashes, there will be at most f subsumptions of the DFSs that start with $roundNo > 1$. Each subsumption happens in at most $O(k^2\Delta_{out})$ rounds and hence Dispersion is achieved in $k^2\Delta_{out} + f \cdot O(k^2\Delta_{out}) = O(f \cdot k^2\Delta_{out})$ rounds. □

Proof of Theorem 1(b). The time bound follows from the proofs of Lemmas 16 and 18. For the memory bound, in the fault-free case, there are total ten different variables maintained at each robot and each variable is of size either $O(1)$ or $O(\log k)$ or $O(\log \Delta_{out})$. Therefore, the total memory bound is $O(\log(k+\Delta_{out}))$ bits per robot. In the crash fault case, the tupled DFSID only adds $O(\log(k+\Delta_{out}))$ factor due to the variable $roundNo$, since $roundNo \le O(f \cdot k^2 \cdot \Delta_{out}) = O(k^3 \cdot \Delta_{out})$ given that $f \le k$, and thus can be represented by $O(\log(k+\Delta_{out}))$ bits. □

BFS Algorithm. We extend Algorithm 3 to handle crashes. We again have two cases. Case 1 stays the same. In Case 2, the matching may be interrupted due to the faulty robot(s). Therefore, in Stage 1, a source node v sends broadcast message repetitively in every round until a request message is received by v. If a node receives broadcast message from v multiple times, then it stores the latest copy among them and forwards it to its out-going neighbors. In Stage 2, if a request message can not reach to v, then some intermediate node (robot) in the path to v must have crashed. A non-crashed neighboring robot of the crashed robot on the path to v becomes a request node and sends a request message to v when it receives a broadcast message from v. In Stage 3, if a request node w finds that a node in its path $P(v,w)$ is crashed, the neighboring node becomes a new request node.

Theorem 9. *In the same setting of Theorem 8 having $f \le k$ robot crashes, the BFS algorithm solves* Dispersion *deterministically in $O(k \cdot D)$ rounds under the 1-hop communication model.*

Proof. In the fault-free case, at least one empty node is occupied by a robot in $O(D)$ rounds. We prove here that, for each crash fault, this matching can be delayed by at most $O(D)$ rounds. Therefore, with f faults, matching between a source and a request node will be delayed by at most $O(f \cdot D)$ rounds. If faults occur in the interval of $\le 3D$ rounds, then after one total delay of $O(f \cdot D)$ rounds, a matching happens in every $O(D)$ rounds. Since f robots already experienced fault, there will be only $k-f$ robots that need to be settled and they will settle in next $O((k-f) \cdot D)$ rounds, giving in total $O(k \cdot D)$ rounds time complexity. If faults occur in the interval of $> 3D$ rounds, then between two subsequent faults, one matching is already done and hence the total delay is again at most $O(f \cdot D)$ rounds. After f faults, there will be only $k-f$ non-faulty robots and they will need additional $O((k-f) \cdot D)$ rounds to settle. Therefore, Dispersion finishes in $O(f \cdot D + (k-f) \cdot D) = O(k \cdot D)$ rounds. □

Proof of Theorem 1(c). The time bound follows from the proofs of Theorems 8 and 9. For the memory bound, there can be at most $k/2$ source nodes. Therefore, a node may need to keep track of the parent information on the $k/2$ BFS trees that are built by those source nodes, which requires in total $O(k\log(k+\Delta_{out}))$ bits at each robot, combining the $O(\log k)$ bits required to remember the ID of each robot. The broadcast, request, and other messages are of $O(\log k)$ size. Therefore, in total the memory at each robot is $O(k\log(k+\Delta_{out}))$ bits. Even in the faulty case, the memory does not change since for the broadcast messages received from a source node multiple times, the previous records were discarded and only the latest message was stored in memory. □

8. Concluding remarks

In this paper, we have studied Dispersion of mobile robots for the first time in directed anonymous graphs. We have established an impossibility result as well as some time and memory lower bounds. We then presented three deterministic algorithms, the first Kwek's algorithm based, the second DFS traversal based, and the third BFS traversal based.

For the future work, it would be interesting to establish bounds for BFS algorithms in the local model (removing 1-hop assumption). Moreover, it would be interesting to improve the time bound of our DFS algorithm to $O((k+f)k \cdot \Delta_{out})$ (in the crash case) and the memory bound of our BFS algorithm to $O(\log(k+\Delta_{out}))$ bits/robot. Furthermore, it would be interesting to solve Dispersion in dynamic directed graphs. Finally, it would be interesting to extend our results to the semi-synchronous and asynchronous settings.

CRediT authorship contribution statement

Giuseppe F. Italiano: Writing – review & editing, Supervision, Conceptualization. **Debasish Pattanayak:** Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization. **Gokarna Sharma:** Writing – review & editing, Writing – original draft, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] G.F. Italiano, D. Pattanayak, G. Sharma, Dispersion of mobile robots on directed anonymous graphs, in: M. Parter (Ed.), Proceedings, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, in: Lecture Notes in Computer Science, vol. 13298, Springer, 2022, pp. 191–211.
[2] 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.
[3] 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.
[4] J. Augustine, W.K. Moses Jr., Dispersion of mobile robots: a study of memory-time trade-offs, in: ICDCN, 2018, pp. 1:1–1:10.

[5] A.D. Kshemkalyani, F. Ali, Efficient dispersion of mobile robots on graphs, in: ICDCN, 2019, pp. 218–227.
[6] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Fast dispersion of mobile robots on arbitrary graphs, in: ALGOSENSORS, 2019, pp. 23–40.
[7] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Dispersion of mobile robots in the global communication model, in: ICDCN, 2020, pp. 12:1–12:10.
[8] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Dispersion of mobile robots on grids, in: WALCOM, 2020, pp. 183–197.
[9] A.D. Kshemkalyani, G. Sharma, Near-optimal dispersion on arbitrary anonymous graphs, arXiv:2106.03943.
[10] A.R. Molla, W.K.M. Jr., Dispersion of mobile robots: the power of randomness, in: TAMC, 2019, pp. 481–500.
[11] S. Kwek, On a simple depth-first search strategy for exploring unknown graphs, in: G. Goos, J. Hartmanis, J. van Leeuwen, F. Dehne, A. Rau-Chaplin, J.-R. Sack, R. Tamassia (Eds.), Algorithms and Data Structures, vol. 1272, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 345–353.
[12] T. Shintaku, Y. Sudo, H. Kakugawa, T. Masuzawa, Efficient dispersion of mobile agents without global knowledge, in: SSS, 2020, pp. 280–294.
[13] A.D. Kshemkalyani, A.R. Molla, G. Sharma, Efficient dispersion of mobile robots on dynamic graphs, in: ICDCS, 2020, pp. 732–742.
[14] D. Pattanayak, G. Sharma, P.S. Mandal, Dispersion of mobile robots tolerating faults, in: ICDCN, 2021, pp. 133–138.
[15] A.R. Molla, K. Mondal, W.K. M. Jr., Efficient dispersion on an anonymous ring in the presence of weak Byzantine robots, in: ALGOSENSORS, 2020, pp. 154–169.
[16] A.R. Molla, K. Mondal, W.K. M. Jr., Byzantine dispersion on graphs, in: IPDPS, IEEE, 2021, pp. 942–951.
[17] 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.
[18] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, D. Peleg, Label-guided graph exploration by a finite automaton, ACM Trans. Algorithms 4 (4) (2008) 42:1–42:18.
[19] D. Dereniowski, Y. Disser, A. Kosowski, D. Pajak, P. Uznański, Fast collaborative graph exploration, Inf. Comput. 243 (C) (2015) 37–49.
[20] 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.
[21] A.D. Kshemkalyani, F. Ali, Fast graph exploration by a mobile robot, in: AIKE, 2018, pp. 115–118.
[22] A. Menc, D. Pajak, P. Uznanski, Time and space optimality of rotor-router graph exploration, Inf. Process. Lett. 127 (2017) 17–20.
[23] P. Fraigniaud, L. Gasieniec, D.R. Kowalski, A. Pelc, Collective tree exploration, Networks 48 (3) (2006) 166–177.
[24] M.A. Bender, D.K. Slonim, The power of team exploration: two robots can learn unlabeled directed graphs, in: FOCS, 1994, pp. 75–85.
[25] M.A. Bender, A. Fernández, D. Ron, A. Sahai, S.P. Vadhan, The power of a pebble: exploring and mapping directed graphs, in: STOC, 1998, pp. 269–278.
[26] S. Albers, M.R. Henzinger, Exploring unknown environments, in: STOC, ACM, 1997, pp. 416–425.
[27] K. Foerster, R. Wattenhofer, Lower and upper competitive bounds for online directed graph exploration, Theor. Comput. Sci. 655 (2016) 15–29.
[28] S. Kutten, Stepwise construction of an efficient distributed traversing algorithm for general strongly connected directed networks or: traversing one way streets with no map, in: J. Raviv (Ed.), Computer Communication Technologies for the 90's, Proceedings of the Ninth International Conference on Computer Communication, Tel Aviv, Israel, October 30 - November 3, 1988, International Council for Computer Communication/Elsevier, 1988, pp. 446–452.
[29] X. Deng, C. Papadimitriou, Exploring an unknown graph, in: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science, vol. 1, 1990, pp. 355–361.
[30] R. Fleischer, G. Trippen, Exploring an unknown graph efficiently, in: G.S. Brodal, S. Leonardi (Eds.), Algorithms – ESA 2005, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 11–22.
[31] Y. Elor, A.M. Bruckstein, Uniform multi-agent deployment on a ring, Theor. Comput. Sci. 412 (8–10) (2011) 783–795.
[32] M. Shibata, T. Mega, F. Ooshita, H. Kakugawa, T. Masuzawa, Uniform deployment of mobile agents in asynchronous rings, in: PODC, 2016, pp. 415–424.
[33] L. Barriere, P. Flocchini, E. Mesa-Barrameda, N. Santoro, Uniform scattering of autonomous mobile robots in a grid, in: IPDPS, 2009, pp. 1–8.
[34] P. Poudel, G. Sharma, Time-optimal uniform scattering in a grid, in: ICDCN, 2019, pp. 228–237.
[35] G. Cybenko, Dynamic load balancing for distributed memory multiprocessors, J. Parallel Distrib. Comput. 7 (2) (1989) 279–301.
[36] P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Oblivious Mobile Robots, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2012.
[37] P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Mobile Entities, Theoretical Computer Science and General Issues, vol. 1, Springer, 2019.
[38] Y. Disser, F. Mousset, A. Noever, N. Skoric, A. Steger, A general lower bound for collaborative tree exploration, CoRR, arXiv:1610.01753.

Giuseppe F. Italiano is Professor of Computer Science at Luiss University. He has been Visiting Professor at Columbia University, Université Paris-Sud, Max–Planck–Institut fur Informatik and Hong Kong University of Science & Technology. Most of his research is centered around algorithms, and in 2016 he was nominated Fellow of the European Association for Theoretical Computer Science for "fundamental contributions to the design and analysis of algorithms for solving theoretical and applied problems in graphs and massive datasets, and for his role in establishing the field of algorithm engineering".

Debasish Pattanayak is currently a Postdoctoral Researcher at the University of Ottawa, Canada. He earned his Ph.D. from the Department of Mathematics at the Indian Institute of Technology Guwahati in 2020 and his B.Tech. in Mathematics and Computing from the same department in 2014. In 2019, he visited University of Vienna under SERB Overseas Visiting Doctoral Fellowship. In 2020, he was a Visiting Scientist at the Indian Statistical Institute in Kolkata. From October 2020 to October 2022, he was a Postdoctoral Researcher at LUISS Guido Carli, Rome. Then in 2022-23, he was a Postdoctoral Researcher at University of Quebec, Ottawa and in 2023-24, at Carleton University, Ottawa. His research focuses on designing distributed algorithms, with particular interest in autonomous mobile robots.

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 ACM and IEEE.