Efficient Collaborative Tree Exploration with Breadth-First Depth-Next

Romain Cosson

Inria, Paris, France

Laurent Massoulié

Inria, Paris, France

Laurent Viennot

Inria, Paris, France

Abstract

We study the problem of *collaborative tree exploration* introduced by Fraigniaud, Gasieniec, Kowalski, and Pelc [10] where a team of k agents is tasked to collectively go through all the edges of an unknown tree as fast as possible and return to the root. Denoting by n the total number of nodes and by D the tree depth, the $\mathcal{O}(n/\log(k)+D)$ algorithm of [10] achieves a $\mathcal{O}(k/\log(k))$ competitive ratio with respect to the cost of offline exploration which is at least $\max\{2n/k, 2D\}$. Brass, Cabrera-Mora, Gasparri, and Xiao [1] study an alternative performance criterion, the competitive overhead with respect to the cost of offline exploration, with their $2n/k+\mathcal{O}((D+k)^k)$ guarantee. In this paper, we introduce 'Breadth-First Depth-Next' (BFDN), a novel and simple algorithm that performs collaborative tree exploration in $2n/k+\mathcal{O}(D^2\log(k))$ rounds, thus outperforming [1] for all values of (n, D, k) and being order-optimal for trees of depth $D=o(\sqrt{n})$. Our analysis relies on a two-player game reflecting a problem of online resource allocation that could be of independent interest. We extend the guarantees of BFDN to: scenarios with limited memory and communication, adversarial setups where robots can be blocked, and exploration of classes of non-tree graphs. Finally, we provide a recursive version of BFDN with a runtime of $\mathcal{O}_\ell(n/k^{1/\ell}+\log(k)D^{1+1/\ell})$ for parameter $\ell \geq 1$, thereby improving performance for trees with large depth.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Distributed algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases collaborative exploration, online algorithms, trees, adversarial game, competitive analysis, robot swarms.

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements The authors thank the anonymous reviewers for their useful remarks and the entire Argo team at Inria for enlightening discussions. RC thanks Pierre Fraigniaud for precious advice and Maxime Cartan for his implementation of a Python demo (available at https://github.com/Romcos/BFDN). This work was supported by PRAIRIE ANR-19-P3IA-0001 and Tempogral ANR-22-CE48-0001. A brief announcement appeared in PODC' 23 [4] and some extensions are available on arXiv [3].

1 Introduction

Problem setting. A team of robots[1], initially located at the root of an unknown tree, is tasked to collectively go through all the edges of a tree as fast as possible and then return to the root. At each round, the robots move synchronously along one incident edge to reach a neighbour, thereby discovering new adjacent edges. Following [10], we consider two distinct communication models. The *complete communication* model, in which communications

[1] the term "robots" is often preferred over "agents" in line with the initial work of [10].

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are unrestricted and consequently the team takes decisions in a centralized fashion. The *write-read* communication model, in which robots communicate through whiteboards that are located at all nodes and must thus take decisions in a distributed fashion.

Main results. In this paper, we present a simple and novel algorithm that achieves collaborative tree exploration with k agents in $\frac{2n}{k} + D^2(\min\{\log(k), \log(\Delta)\} + 3)$ rounds for any tree with n nodes, depth D and maximum degree Δ. This algorithm can be implemented in the complete communication model and the write-read communication model.

The algorithm is called "Breadth-First Depth-Next" (abbreviated `BFDN`) and the behaviour of the robots can be described synthetically as follows: when located at the root, a robot is sent to the highest unexplored edge (as in a breadth-first search). Upon arrival, the robot changes behaviour until it reaches the root again, it goes through unexplored edges when adjacent to one and goes up towards the root otherwise (as in a depth-first search).

Our analysis involves a simple zero-sum two-player game with balls in urns. An immediate application of this analysis is in resource allocation in the face of uncertainty. Given k workers and k (parallelizable) tasks requiring each an unknown amount of work, we show that the strategy of reassigning idle workers to the least crowded task is competitive in terms of number of times a worker will have to switch between tasks. More precisely, we show that this number is at most $k \log(k) + 2k$.

The `BFDN` algorithm is easy to implement and we provide it with extensions to more complex settings, such as i) exploration of specific classes of non-tree graphs, ii) scenarios with constrained communications and memory, and iii) setups where an adversary chooses at each time step which robots are allowed to move. Finally, in an attempt to improve dependence in the tree depth D, we propose $\texttt{BFDN}_\ell$, a recursive version of `BFDN` in the complete communication model that explores the tree in time $\mathcal{O}_\ell\left(\frac{n}{k^{1/\ell}} + \min\{\log(k), \log(\Delta)\} D^{1+1/\ell}\right)$ where $\ell \geq 1$ is some constant provided as input.

Useful context and related works. In the case of a single robot, the "Depth First Search" (`DFS`) algorithm is optimal for traversing the edges of a tree. It can be implemented both *offline* (the tree is known in advance) and *online* (edges are revealed when reached). One way to describe DFS in an online fashion is to have the robot go through an adjacent unexplored edge if possible and go up towards the root otherwise. After $2(n-1)$ rounds, where n is the number of nodes, all edges have been traversed (twice) and the robot is at the root.

In the multi-robot setting, i.e. with $k \geq 2$, traversing all the edges of a tree in an *offline* manner requires at least $\max\{2n/k, 2D\} \geq n/k + D$ synchronous rounds [7, 13]. This is because every edge has to be traversed in both directions and some robot has to reach the deepest node before returning to the root. A simple algorithm [7, 13] matches this bound up to a factor 2, with a runtime of at most $2(n/k + D)$: consider a depth-first search path from the root of length $2(n-1)$, and divide it in k segments each of length $\lceil 2(n-1)/k \rceil$, then assign one robot to reach and traverse each segment. The optimal offline k-traversal is NP-hard to compute as [10] gave a reduction from `3-PARTITION` to this problem.

To analyze the *online* problem (i.e. collective tree exploration), the literature initially focused on the *competitive ratio* which is the worst-case ratio between the cost an online algorithm and the optimal offline algorithm. For an online algorithm $\mathcal{A}_k$ using $k \geq 2$ robots, this ratio is defined up to a constant factor as $\max_{n,D\in\mathbb{N}} \max_{T\in\mathcal{T}(n,D)} \texttt{Runtime}(\mathcal{A}_k, T)/(n/k + D)$ where $\mathcal{T}(n, D)$ denotes the set of all trees with n nodes and depth D. The algorithm proposed initially by [10] `CTE` (Collective Tree Exploration) runs in $O(\frac{n}{\log k} + D)$ rounds for any tree $T \in \mathcal{T}(n, D)$ and therefore has a competitive ratio of $O(\frac{k}{\log k})$. Furthermore, it can be implemented in the write-read communication model [10]. It was later shown by [11] that the competitive analysis of CTE is tight as they provided a simple construction of a tree with $n = kD$ edges that CTE would take $\frac{Dk}{\log_2(k)}$ time-steps to explore. To date, no algorithm is known to have a better competitive ratio than CTE, while the best lower-bound known on the competitive ratio, for deterministic exploration algorithms, is in $\Omega(\frac{\log k}{\log \log k})$ by [9].

The limited progress on the analysis of the competitive ratio as a function of k led most subsequent works to investigate algorithms with super-linear dependence in (n, D), usually assuming complete communication [13, 1, 8, 6, 5, 11]. In this spirit, [13] derived a recursive algorithm called Yo* that runs in $\mathcal{O}(2^{\mathcal{O}(\sqrt{\log D \log \log k})} \log(k)(\log(k) + \log(n))(n/k + D))$ rounds. On the other hand, [1] proposed a novel analysis of CTE yielding a guarantee of $\frac{2n}{k} + \mathcal{O}((k+D)^k)$, displaying optimal dependence in n at the cost of large additive dependence in (k, D). The algorithm we propose with its guarantee of $\frac{2n}{k} + \mathcal{O}(D^2 \log(k))$ complements this line of work. Our guarantee yields a strict improvement over [1] for all values of (n, k, D), and improves upon CTE and Yo* for the specific range of parameters as depicted in Figure 1.

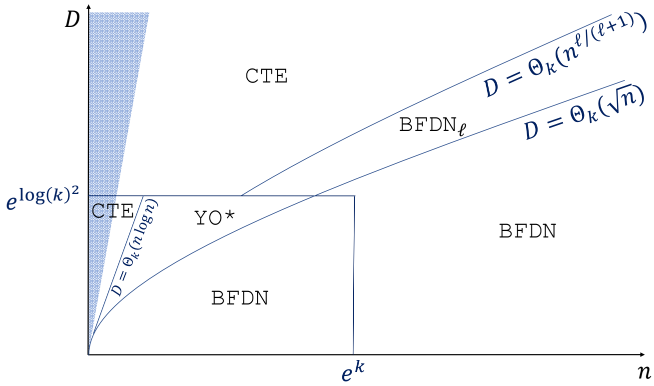

Figure 1 Regions of (n, D) where either of CTE, Yo*, BFDN and BFDN$_\ell$ has the best runtime guarantee. The runtime of algorithm Yo* was simplified to improve readability. ℓ must satisfy $\ell \leq \text{cst}(\log k / \log \log k)$. No trees defined in shaded region $n \leq D$. See Appendix A for details.

Collaborative tree exploration has also been studied under additional assumptions. For example, for trees which can be embedded in the 2-dimensional grid, [8] obtained an algorithm running in $\mathcal{O}(\sqrt{D}(\frac{n}{k} + D))$ rounds. The setting where the number of robots k is very large, specifically $k \geq Dn^c$ for some constant $c > 1$, was also investigated by [5]. Assuming global communication, their algorithm achieves exploration in $\frac{c}{c-1}D + o(D)$ rounds. Interestingly, their guarantees also apply to the challenging and less studied collaborative graph exploration problem; see also [1, 2].

Open directions. In line with [1], our work advocates for the study of the *competitive overhead* of collaborative exploration in complement to its *competitive ratio*. Recently [6] showed that (deterministic) collaborative exploration with $k = n$ requires at least $\Omega(D^2)$, implying that no algorithm can have a $\frac{2n}{k} + \mathcal{O}(D^c)$ guarantee for $c < 2$. On the other hand, a simple algorithm explores any tree in $\mathcal{O}(D^2)$ rounds as soon as $k \geq \frac{n}{D}$ [13]. In view of these results, our $\frac{2n}{k} + \mathcal{O}(D^2 \log(k))$ guarantee seems close-to-optimal. We highlight the open question of whether there exists a $\frac{2n}{k} + \mathcal{O}(D^2)$ exploration algorithm, or even a guarantee of the form $\frac{2n}{k} + \mathcal{O}(f(D))$, for some real-valued function f.

Structure of the paper. Section 2 defines algorithm BFDN and provides the main result for the complete communication setting. Section 3 analyzes a two-player zero-sum board game, an essential ingredient in our analysis of BFDN. Section 4 contains extensions of BFDN to settings with: limited communications; adversarial interruption of robots; and more general graph exploration. Finally, Section 5 provides a recursive version of BFDN that yields improved runtime guarantees when the tree depth D gets larger compared to n.

Notations. $\log(\cdot)$ refers to the natural logarithm and $\log_2(\cdot)$ to the logarithm in base 2. For an integer k we use the abbreviation $[k] = \{1, \dots, k\}$.

A tree $T = (V, E)$ is defined by its set of nodes V and edges $E \subset V \times V$; it is rooted at some specific node denoted $\texttt{root} \in V$ from which all robots start the exploration. For a node $v \in V$, $\delta(v)$ is the distance of v to the root and $T(v)$ denotes the sub-tree of T rooted at v containing all the descendants of v. The depth of T is $D = \max_{v \in V} \delta(v)$. We will also use a notion of *partially explored tree* (defined in Section 2) that enjoys the same definitions.

2 The Breadth-First Depth-Next algorithm

Our main result on BFDN, which is described below, is the following

▶ **Theorem 1.** BFDN *achieves online exploration of any tree with k robots in at most*

$$\frac{2n}{k} + D^2(\min\{\log(\Delta), \log(k)\} + 3)$$

rounds, where Δ is the maximum degree, n is the number of nodes, and D is the depth.

Following [10], we shall start by showing the guarantee in the complete communication model, and we later present in Section 4 how BFDN can be adapted to the write-read model.

Partially explored tree. At a given exploration round, V denotes the set of *explored nodes*, i.e. nodes that have been occupied by at least one robot in the past, and E denotes the set of *discovered edges*, i.e. edges that have at least one explored endpoint. The discovered edges that have exactly one explored endpoint are called *dangling edges*. Such edges can be viewed as a pair $(u, ?)$, with $u \in V$. The *partially explored tree* or *discovered tree* $T_{\text{online}} = (V, E)$ contains all the information gathered by the robots at some point of exploration. If there are no dangling edges in T_{online}, it means that exploration is complete and that the partially explored tree equals the underlying tree $T_{\text{offline}} \in \mathcal{T}(n, D)$.

Collaborative exploration algorithm. A collaborative exploration algorithm in the complete communication model is formally defined as a function that maps a partially explored tree $T = (V, E)$ as well as the list of positions of the agents $p_1, \dots, p_k \in V^k$ and their past movements to a list of *selected edges* $e_1, \dots, e_k \in (E \cup \{\perp\})^k$ that the agents will use for their next move. Each selected edge $e_i \in E$ must be adjacent to the position p_i. Dangling edges may be selected. By convention, $e_i = \perp$ is used to indicate that agent i will not move at the next round. In pseudo-code, the routine $\texttt{SELECT}(\texttt{Robot}_i, e)$ performs the assignment $e_i \leftarrow e$. When all agents have selected a next move, the routine MOVE is applied and all agents move along their selected edge synchronously. The partially explored tree (V, E) is then updated with the new information provided by the agents that have traversed a dangling edge. Exploration always starts with all agents located at the root, $V = \{\texttt{root}\}$ and E the set of all dangling edges that are adjacent to the root. The collaborative exploration algorithm is applied iteratively. Exploration terminates when the explored tree (V, E) contains no dangling edges and when the position of all agents is back at the root. The runtime of an exploration algorithm is defined as a function of (n, D) by the number of rounds required before termination on any tree with n nodes and depth D.

Breadth-First Depth-Next Algorithm. We now provide a brief description of BFDN, Algorithm 1. When located at the root, a robot indexed by $i \in [k]$ and denoted $\texttt{Robot}_i$ is assigned an *anchor* $v_i \in V$ which is a node that is adjacent to at least one dangling edge. If no such node exists, the anchor is the root itself. The exact anchor assignment is specified by procedure Reanchor which gives the priority to nodes that are the closest to the root and that have the least number of anchored robots. $\texttt{Robot}_i$ then attains this anchor in a series of breadth-first moves performed with procedure BF. When the anchor is reached, the robot only makes depth-next moves with procedure DN, until it returns to the root. In a sequence of depth-next moves, the robot always goes through a dangling edge if one is available (i.e. adjacent and not already selected as next move by another robot), and goes one step up towards the root otherwise. This will result in a depth-first-like exploration inside $T(v_i)$ followed by a direct travel from v_i to the root. The algorithm stops when all robots are at the root and are not assigned a new anchor because there are no more dangling edges.

The reason why we ask that the robots go back all the way to the root before being reassigned a new anchor, rather than having them use a shortest path from their previous anchor to their next anchor, will become apparent when we adapt the algorithm to the distributed write-read communication setting. In that setting, the root will play the role of a central planner, gathering information on the advancement of exploration thanks to returning robots.

2.1 Analysis of BFDN and proof of Theorem 1

We first prove the correctness and termination of BFDN and then bound its runtime.

Correctness. In Algorithm 1, the do-while loop is interrupted when no robot changes position at some round (line 14). Note that the root is the only place where robots may stay at the same position because direction up is interpreted as $\perp$ at the root only (line 23). Thus all robots are at the root when the algorithm stops. Also note that the selection of direction up by all robots at the root implies that there are no dangling edges in the tree. Thus the tree has been entirely explored and all robots have returned. The algorithm is correct.

Termination. To prove termination, we show that while the algorithm runs, a node is discovered every $3D$ rounds at least. Since there are n nodes in the tree, the algorithm must terminate after at most $3D \times n$ rounds. Assume by contradiction that no node is discovered in a sequence of $3D$ rounds. After $2D$ rounds, all robots have attained the root because all DF moves are directed up. Then, either one robot is assigned an anchor that is adjacent to an unexplored edge which will be traversed in the coming D rounds, or the algorithm stops. In both cases we have a contradiction.

We now provide the following lemma which will be proved in Section 3.

▶ **Lemma 2.** *In an execution of* BFDN*, for any $d \in \{1, \dots, D-1\}$, the number of calls to procedure* Reanchor *which return an anchor at depth d is at most* $k(\min\{\log(k), \log(\Delta)\} + 3)$.

Time complexity. During the execution, a given $\texttt{Robot}_i$ anchored at v_i can spend time in two different ways (1) being idle at the root (2) moving along a selected edge. We denote by $\mathsf{T}_i^1, \mathsf{T}_i^2$ the time (number of rounds) spent by $\texttt{Robot}_i$ in each of these phases. We have that $\sum_{i \in [k]} (\mathsf{T}_i^1 + \mathsf{T}_i^2) = k\mathsf{T}$ where T is the total number of rounds of the algorithm as the k robots operate in parallel. We now prove a series of claims.

▶ **Claim 1.** *The total number of rounds when some robot does not move is at most $D + 1$.*

Algorithm 1 `BFDN` "Breadth-First Depth-Next"

Ensure: The robots traverse all edges and return to the root.

V = list of explored nodes ; E = list of discovered edges
$v_i \leftarrow$ `root` $\forall i \in \{1, \dots, k\}$ ▷ Initialize anchors.
$S_i \leftarrow [\,]$ $\forall i \in \{1, \dots, k\}$ ▷ Initialize empty stacks.
do ▷ Round t.
  for $i = 1$ to k **do** ▷ Sequential decisions.
    if `Robot`$_i$ is at `root` **then**
      $v_i \leftarrow$ `Reanchor`(i)
      Stack in S_i the list of edges that lead to v_i ▷ Reverse order.
    if S_i is not empty **then**
      `BF`(i)
    else
      `DN`(i)
  `MOVE` all robots on their selected edge and update (V, E) ▷ Synchronous moves.
while some robot changes position

procedure BF(i)
  Unstack $e \in E$ from S_i and `SELECT`(`Robot`$_i$, e)

procedure DN(i)
  if `Robot`$_i$ is adjacent to some dangling and unselected edge $e \in E$ **then**
    `SELECT`(`Robot`$_i$, e)
  else
    `SELECT`(`Robot`$_i$, `up`) ▷ If `Robot`$_i$ is at the root, `up` is interpreted as $\bot$.

procedure Reanchor(i)
  $U = \{v \in V \;\; s.t. \;\; v$ is adjacent to some dangling edge with $\delta(v)$ minimal$\}$
  if $U \neq \emptyset$ **then** ▷ Choose anchor of minimum load.
    $v_i \leftarrow \arg\min_{v \in U} n_v$ where $\forall v \in V : n_v = \#\{j \in [k] \;\; s.t. \;\; v_j = v\}$
  else ▷ The tree is explored.
    $v_i \leftarrow$ `root`

Proof of Claim 1. Recall that if a robot does not move, it must be anchored at the root and have selected direction `up` with procedure `DN`. This only occurs in two cases (1) there are no more dangling edges in the discovered tree (this happens at most D times because all robots are on their way back) (2) there are still dangling edges that are adjacent to the root, but they are all selected (this happens at most once because at the next time-step, all edges adjacent to the root will be explored). The number of time-steps when a robot may not move is thus at most $D + 1$. ◀

▶ **Claim 2.** *In the round when a dangling edge is explored for the first time, it is traversed by a single robot.*

Proof of Claim 2: All breadth-first moves (with procedure `BF`) are through previously explored edges because they lead from the root to a previously explored node. Thus dangling edges are only explored in depth-next moves (with procedure `DN`). In this procedure, two robots cannot select the same dangling edge. ◀

▶ **Claim 3.** *Consider a sequence of moves by some* $\texttt{Robot}_i$ *that starts at the root with the assignment of an anchor* v *of depth* $\delta(v) = d$ *and that ends with the return of* $\texttt{Robot}_i$ *to the root after* T_x *rounds. In this sequence,* $\texttt{Robot}_i$ *explored exactly* $(\mathsf{T}_x - 2d)/2$ *dangling edges.*

Proof of Claim 3. The sequence of moves, denoted x, has the following structure. First, $\texttt{Robot}_i$ uses a shortest path from the $\texttt{root}$ to v which takes d moves through previously explored edges. Then the robot performs moves inside $T(v)$ by going down through dangling edges if some are available and going up towards the root otherwise. Note that exactly half of the moves inside $T(v)$ must be through dangling edges as there must be as many moves down as moves up in $T(v)$. Finally, the robot goes back from v to the root in again d moves through explored edges. Exactly $(\mathsf{T}_x - 2d)/2$ dangling edges are explored in this sequence. ◀

We now assemble the claims and Lemma 2 together to bound the runtime of $\texttt{BFDN}$. Using Claim 1, we have that $\sum_i \mathsf{T}_i^1 \leq k(D+1)$. Then, we write $\sum_i \mathsf{T}_i^2 = \sum_{d \leq D-1} \sum_{x \in X_d} \mathsf{T}_x$ where X_d is the list of all sequences of moves x that start with the assignment of an anchor v at depth $\delta(v) = d$ to some robot and that end with the return of that robot to the root. Using Claim 2 and Claim 3, we have that $\sum_{d \leq D-1} \sum_{x \in X_d} (\mathsf{T}_x - 2d)/2 \leq n - 1$. Consequently,

$$\sum_{i \in [k]} \mathsf{T}_i^2 \leq 2(n-1) + 2 \sum_{d \leq D-1} \sum_{x \in X_d} d.$$

By Lemma 2, the cardinality of X_d is at most $k(\min\{\log(k), \log(\Delta)\}+3)$, for $d \in \{1, \ldots, D-1\}$. Thus, $\sum_{d \leq D-1} \sum_{x \in X_d} d \leq \frac{D(D-1)}{2} k(\min\{\log(k), \log(\Delta)\} + 3)$. Finally, using $\sum_{i \in [k]} (\mathsf{T}_i^1 + \mathsf{T}_i^2) = k\mathsf{T}$, we obtain $k\mathsf{T} \leq 2(n-1) + D(D-1)k(\min\{\log(\Delta), \log(k)\} + 3) + (D+1)k$, which proves that the algorithm stops after at most

$$\mathsf{T} \leq \frac{2n}{k} + D^2(\min\{\log(\Delta), \log(k)\} + 3)$$

steps, thus completing Theorem 1's proof.

Though it is not required for the the analysis above, we conclude this section with a final claim that provides useful intuition on the algorithm.

▶ **Claim 4.** *At all rounds, all dangling and unexplored edges, are in* $\cup_{i \in [k]} T(v_i)$*.*

Proof of Claim 4. Consider some dangling edge e and its explored endpoint $v \in V$. At the round when v was explored by a robot, that robot was performing a depth-next move because its anchor was at least as high as v which is still adjacent to a dangling edge. That robot cannot have left $T(v)$ before the edge e was traversed. Consequently, it is still rooted at some ancestor v_i of v, thus $e \in \cup_{i \in [k]} T(v_i)$. ◀

3 A two-player zero-sum game with balls in urns

In this section we introduce a two-player zero-sum board game that essential to the analysis of $\texttt{BFDN}$. A strategy for the player of the game is given and analyzed in Theorem 3. Its connection with $\texttt{BFDN}$ is detailed in Section 3.2 where a proof of Lemma 2 is given.

3.1 Game of balls in urns.

Game description. At time $t \in \mathbb{N}$, the board of the game is a list of k integers $(n_1^t, \ldots, n_k^t)$ that represent the load of k urns with a total of k balls. When the game starts at $t = 0$, we have $n_i^0 = 1$ and at every instant t we have $\sum_{i\in[k]} n_i^t = k$ and $n_i^t \geq 0$. At time t, player A (the adversary) chooses a ball in an urn $a_t \in [k]$ that is not empty, i.e. such that $n_{a_t}^t \geq 1$, and then player B (the player) chooses an urn $b_t \in [k]$ and moves that ball from urn a_t to urn b_t. At the beginning of time $t+1$, the board satisfies $n_{a_t}^{t+1} = n_{a_t}^t - 1$ and $n_{b_t}^{t+1} = n_{b_t}^t + 1$.

Goal of the game. At a given time t, we denote by U_t the set of urns that have never been selected by the adversary, $U_t = \{1, \dots, k\} \setminus \{a_0, \dots, a_{t-1}\}$. The game stops when all urns in U_t contain at least Δ balls, i.e. $n_i^t \geq \Delta, \forall i \in U_t$. If $\Delta \geq k$, the game stops when all urns have been chosen, i.e. $U_t = \emptyset$. The goal of player B is to end the game as soon as possible, while the goal of the adversary is to play for as long as it can.

Strategy of the player. At time t, the player picks the urn b_t that contains the least number of balls among the urns that were never chosen by the adversary, i.e. $b_t \in \arg\min_{i\in[k]\setminus\{a_0,\dots,a_t\}} n_i^t$. For this strategy, we state the main result of this section.

▶ **Theorem 3.** *Under this strategy, the game ends after at most $k \min\{\log(\Delta), \log(k)\} + 2k$ steps.*

Interpretation of the game. While the main focus of this paper is on collective tree exploration, a more immediate application of the above result is in resource allocation in the face of uncertainty. Given k workers and k (parallelizable) tasks of unknown length, our analysis shows that the 'best' way to reassign idle workers online is to reassign them to the unfinished task which has the least number of workers working on it. Using this simple rule, the number of times a worker changes task is at most $\log(k) + 2$ times the optimum (which is of order k) irrespective of the individual task lengths.

Proof. The set U_t does not increase with time. We denote its cardinality $u_t = |U_t|$. Denoting $N_t = \sum_{i\in U_t} n_i^t$ the total number of balls in urns of U_t, the possible number of balls for an urn of U_t lies in $\{\lceil \frac{N_t}{u_t} \rceil, \lfloor \frac{N_t}{u_t} \rfloor\}$. The game thus stops as soon as $\frac{N_t}{u_t} \geq \Delta$ and the quantity $x_t := \Delta u_t - N_t$, must thus be positive as long as the game lasts. We distinguish two options for the adversary at any step t:

(a) The adversary chooses an urn a_t that it previously chose ($a_t \notin U_t$). In this case, $u_{t+1} = u_t$ and $N_{t+1} = N_t + 1$. Note that this option is available to the adversary only if some ball lies outside of U_t, i.e. if $N_t \leq k - 1$.

(b) The adversary chooses an urn a_t that it has never chosen before ($a_t \in U_t$). In this case, $u_{t+1} = u_t - 1$ and $N_{t+1} = N_t - n_{a_t}^t + 1$.

We now will establish that the adversary always prefer option (a) to option (b). For parameters $u, N \in \{0, \dots, k\}$, we denote by $R(N, u)$ the largest number of steps that the game may still last after player B's move led to a configuration where $N_t = N$ and $u_t = u$ at any time t. Note that by the discussion above, this value is the same for all such configurations of the game. Clearly, $\Delta u - N \leq 0 \Rightarrow R(N, u) = 0$. Besides, in view of the options (a) and (b) just listed, one has the following, assuming $\Delta u - N > 0$:

$$N < k \Rightarrow R(N,u) = 1 + \max \begin{cases} R(N+1, u), \\ R(N - \lceil N/u \rceil + 1, u - 1), \\ R(N - \lfloor N/u \rfloor + 1, u - 1). \end{cases} \tag{1}$$

$$N = k \Rightarrow R(N,u) = 1 + \max \begin{cases} R(N - \lceil N/u \rceil + 1, u - 1), \\ R(N - \lfloor N/u \rfloor + 1, u - 1). \end{cases} \tag{2}$$

We now establish the following,

▶ **Lemma 4.** *For any* $(u, N) \in \{0, \ldots, k\}$*, it holds that:*
i) Function $M \to R(M, u)$ *is non-increasing, and*
ii) The maximum in (1) *for* $N < k$ *is always achieved by* $R(N + 1, u)$*.*

Proof. For $u = 0$, $R(M, u) \equiv 0$ and there is nothing to prove. Assume that the two properties i) and ii) hold for $v = u - 1 \geq 0$. We will show that ii) holds for u. Consider $N < k$. By the monotonicity assumption i),

$$R(N - \lceil N/u \rceil + 1, u - 1) \geq R(N - \lfloor N/u \rfloor + 1, u - 1).$$

Assume thus that the adversary moves first to configuration $(N - \lceil N/u \rceil + 1, u - 1)$. By assumption ii) at rank v, its next best move is to configuration $(N - \lceil N/u \rceil + 2, u - 1)$. If alternatively the adversary had made a first move to $(N + 1, u)$, it could then move to $(N + 1 - \lceil (N + 1)/u \rceil + 1, u - 1)$. Now by the monotonicity assumption ii) this can only improve the adversary's reward if $N - \lceil N/u \rceil + 2 \geq N + 1 - \lceil (N + 1)/u \rceil + 1$, which is obviously true. We have thus established ii) at rank u. Monotonicity i) at rank u readily follows, since we now have that $R(N + 1, u) = R(N, u) - 1$ if $\Delta u - N > 0$. ◀

From the lemma above, we conclude that a strategic adversary always prefer option (a) over option (b) when it is available. Playing option (b) grants the adversary a budget to choose option (a) for another $\lceil \frac{N_t}{u_t} \rceil - 1$ time steps. In such game, u_t is thus decremented by 1 every $\lceil \frac{k}{u_t} \rceil$ steps. The game stops if $u_t \leq \frac{k}{\Delta}$, thus right after $u_t = \lceil \frac{k}{\Delta} \rceil$. Assuming $\Delta \leq k$, the game then lasts a total time of at most $\lceil \frac{k}{k} \rceil + \lceil \frac{k}{k-1} \rceil + \cdots + \lceil \frac{k}{\lceil k/\Delta \rceil} \rceil \leq \sum_{h=\lceil k/\Delta \rceil}^{k} \left(\frac{k}{h} + 1 \right) \leq k \sum_{h \geq k/\Delta+1}^{k} \frac{1}{h} + 2k \leq k \int_{k/\Delta}^{k} \frac{dx}{x} + 2k \leq k(\log(k) - \log(k/\Delta)) + 2k = k \log(\Delta) + 2k$. Instead assuming $k < \Delta$, the game will stop after $u_t = 1$ and the sum is thus bounded by $k \int_1^k \frac{dx}{x} + 2k \leq k \log(k) + 2k$. Overall, the game ends in at most $k \min\{\log(\Delta), \log(k)\} + 2k$ steps. ◀

3.2 Connection to BFDN

We start by giving some intuition to connect the game above to `BFDN` and then provide a proof of Lemma 2. The general picture is that balls of the game will correspond to robots exploring the tree whereas urns of the game will correspond to the anchors at the working depth d, i.e. the minimum depth of a dangling edge. Note that in `BFDN`, procedure `Reanchor` applies the strategy for the player of the game described above, by reassigning the current robot to the anchor of smallest load within set U, which is defined line 26 of Algorithm 1 by,

$$U = \{v \in V \;\; s.t. \;\; v \text{ is adjacent to some dangling edge and } \delta(v) = d\}. \tag{3}$$

▶ **Lemma 2** (Restated). *In an execution of* `BFDN`*, for any* $d \in \{1, \ldots, D - 1\}$*, the number of calls to procedure* `Reanchor` *returning a node at depth* d *is at most* $k(\min\{\log(k), \log(\Delta)\} + 3)$*.*

Proof. We start the proof of the lemma by the following claim on `BFDN`.

▶ **Claim 5.** *At some round, if all anchors are at depth at most* $d - 1$*, all nodes* v *explored at depth* d *are in either of these (non-exclusive) situations: their sub-tree* $T(v)$ *is entirely explored, or their sub-tree* $T(v)$ *hosts exactly one robot.*

Proof of Claim 5. Consider an explored node v at depth d that contains a dangling edge in its sub-tree $T(v)$. We show that $T(v)$ hosts one robot. The dangling edge must have an explored endpoint $v' \in T(v)$ that was attained by a robot performing depth-next moves.

This robot cannot have left $T(v') \subset T(v)$ because v' is still adjacent to a dangling edge, thus that robot is still in $T(v)$. At most one robot is in $T(v)$ because v can only have been attained by a single robot, since all anchors are at depth $d-1$ or above. ◀

We now provide a reduction of the analysis of BFDN to the urns and balls game. We fix some depth $d \geq 1$ and bound the number N_d of times a robot is reanchored at depth d. We denote by U_0 the set U, defined by (3), in the first round when it consists of nodes at depth d. Since all anchors were at depth less than $k-1$ before that round, using Claim 5 we have that $|U_0| \leq k$ (in fact, $|U_0| \leq k-1$ because at least one robot must be at the root). Since all edges at depth less than $d-1$ are explored, we note that U_0 contains all nodes which are possible candidates for anchors at depth d and that $U \subset U_0$ for as long as it concerns nodes at depth d. For each candidate anchor in U_0, we formally re-anchor the robot exploring the corresponding sub-tree to this anchor. This does not change the algorithm's evolution because there are no more dangling edges at depth less than d so all robots head back directly to the root when they have finished explored below the associated candidate anchor.

We then increment counter c at every call of the procedure `Reanchor`, with possibly multiple increments within a single round. For counter value c, we denote by $a_c \in U_0$ the vertex to which the robot was previously anchored, and by $b_c \in U$ the vertex to which it is anchored next. Note that all nodes in $\{a_1, \dots, a_c\}$ can no longer be adjacent to a dangling edge. We stop the increment the last time a robot is anchored at depth d, which happens when there does not remain any node at depth d that is adjacent to some dangling edge.

Consider the number of calls C when for each node in U_0, either a robot returning from it has reached the root, or at least Δ robots are anchored at it. Then C is the duration of a run of the previous two-player game, initialized with one urn containing $k-u$ balls and u urns each containing one ball, where $u = |U_0| \in \{0, \dots, k-1\}$ and where player B implements the balancing strategy. Indeed the re-anchoring strategy of BFDN balances the numbers of robots assigned per anchor. A direct adaptation of our analysis also holds for this modified initial condition of the game, yielding the upper bound on C of $k(\min\{\log \Delta, \log k\} + 2)$. Once C assignments at depth d were made, at least Δ robots are assigned to nodes at depth d that are still adjacent to a dangling edge. In the subsequent d rounds BFDN can anchor each robot at most one last time before there is no more dangling edge at depth d. This yields the announced bound of $k(\min(\log(k), \log(\Delta)) + 3)$ on N_d. ◀

4 Extensions of BFDN to alternative settings

We now consider three settings where a BFDN strategy enjoys non-trivial runtime guarantees.

4.1 Restricted memory and communications

In this section, we study a setting where robots are allowed to communicate with a central planner only when they are located at the root and where they have access to $\Delta + D\log(\Delta)$ bits of internal memory. This setting encompasses the write-read communication model of [10] as detailed in Remark 5. Formally, we precise the setting as follows. At every node, the *ports*, which are defined as the endpoints of the adjacent edges, are numbered from 0 to $\Delta - 1$ where Δ is the maximum degree. A node v at depth $d \leq D$ is identified by the sequence of ports that leads to it from the root with $d \log_2(\Delta)$ bits. For every node distinct from the root, we assume that port number 0 leads to the root. As before, robots operate in rounds. All robots arriving at the root at some round t have their memory read and stored by the planner along with their identifier. The planner can then perform any computation and update the memory of the robots. All robots arriving at some node v distinct from the root at some round t can observe the list of all ports at v from which a robot has returned (these will be called “finished ports”) and are given two choices: `SELECT` a port number as next move, or use a local routine `PARTITION`(v) enjoying the following properties,

- No two robots calling `PARTITION`(v) will ever be sent to the same port $j \geq 1$.
- If a robot calling `PARTITION`(v) at round t is sent to port $j \geq 0$, it means that `PARTITION`(v) has previously sent a robot to all ports $j' \geq j$ at round t or before.

In this model, `BFDN` is implemented as follows. In a stack of d port numbers (each represented by $\log_2(\Delta)$ bits) the central planner assigns to $\texttt{Robot}_i$ an anchor v_i at depth d that it will reach by unstacking port numbers and applying routine `SELECT`. When the robot reaches this node, the stack is empty and the robot will make consecutive calls to routine `PARTITION` that will eventually lead it back to the root. We ask that $\texttt{Robot}_i$ stores the finished port numbers of v_i using its additional Δ bits of memory. This information will be used by the central planner to update its candidates for future anchors, i.e. the value of the set U, as specified by Algorithm 2 below.

▶ Remark 5. The present model encompasses the classical write-read communication model of [10] where robots with unbounded memory communicate by synchronously writing and then synchronously reading information on whiteboards (of infinite size) located at each node of the tree. In this model, the information gathered at the root allows each robot located at the root to emulate the decision taken by the central planner regarding its next anchor assignment. Furthermore, since robots can log their passages at any node (see [10]) the local procedure `PARTITION` can easily be implemented, and the assumption that robots access the list of adjacent port number from which no robot has returned is granted.

▶ **Proposition 6.** *In this restricted communication model, the version of* `BFDN` *described above achieves tree exploration in at most* $\frac{2n}{k} + D^2(\min\{\log(k), \log(\Delta)\} + 3)$ *rounds.*

Proof. We note that the algorithm described above is the same as Algorithm 1, with a minor difference in the definition of U in procedure `Reanchor` line 26, which must now be computed using only information gathered at the root (see Algorithm 2 for details). Informally, U now denotes the set of all nodes at working depth d which *could* be adjacent to a dangling edge, given information collected at the `root`.

The key observation is that a candidate anchor v can be withdrawn from U as soon as a robot which had been anchored at v returns to the root. Consider again the urns-in-balls assignment rule $b_c = \arg\min_{v \in U \setminus \{a_1,\ldots,a_c\}} n_v^c$, where n_v^c denotes the number of robots anchored at v upon increment c, but where nodes in U remain eligible as anchors until some robot has returned to the root from them. The proof of Theorem 3 entails that, for such a modified assignment rule, a robot will have returned from all nodes of U after at most $k(\min\{\log(k), \log(\Delta)\} + 3)$ reassignments, after which the root knows that there can be no more dangling edges at depth d.

Algorithm 2 below precises how the central planner uses information gathered by returning robots to update its knowledge of eligible anchors at the working depth d. Denoting the list of all possible anchors at depth d by A and the list of anchors at depth d from which a robot has returned by R, the planner implements `Reanchor` with set $U = A \setminus R$. When $A \setminus R = \emptyset$, a robot has returned from all anchors at depth d and d is incremented. The planner keeps track of $U' = A' \setminus R'$, which contains the children of A that may be adjacent to a dangling edge, or equivalently the ports of A that are not known to be finished. This update is performed using the memory of the returning robots. ◀

Algorithm 2 BFDN "Breadth-First Depth-Next" (central planner at the root)

Require: At most k robots arriving at the root at some round.
Ensure: Assigns a node v, represented by a sequence of port numbers, to each robot.

d = working depth ;
A = list of anchors at depth d ;
R = nodes of A from which a robot has returned ;
A' = list of children of nodes in A ;
R' = nodes of A' from which a robot has returned ;
Read memory of returning robots and update R, A', R'.
if $A \setminus R = \emptyset$ **then**
  if $A' \setminus R' = \emptyset$ **then**
    Exploration is finished and robots wait at the root.
  else
    $d \leftarrow d + 1$
    $A \leftarrow A' \setminus R'$ ▷ contains at most k elements.
    $R, A', R' \leftarrow \emptyset$
Reanchor the robots to nodes of minimum load in $A \setminus R$, such that after this operation the numbers of robots per anchor differ by at most one.

4.2 Adversarial robot break-downs

So far we assumed that all robots traverse exactly one edge per time-step. We relax this assumption in the present section, assuming instead that some adversary decides at each time-step and for each robot whether the robot actually moves, or instead incurs a break-down, being stalled at its current location. Our aim remains to to explore the tree in as few moves as possible. However we no longer require that the robots return to the root at the end of exploration, because the adversary could decide to break-down some robot indefinitely.

Formally, at each round $t \in \mathbb{N}$, robot i is allowed to make a move if some variable $M_{ti} = 1$ whereas it is blocked at its current position if $M_{ti} = 0$. For this adversarial model, we assume that $\mathbb{M} = (M_{ti})_{t\in\mathbb{N}, i\in[k]}$ is an arbitrary sequence of binary values that takes only a finite number of 1 (allowed moves). We denote the average distance travelled by the robots $A(\mathbb{M})$ which equals $A(\mathbb{M}) = \frac{1}{k}\sum_{t\in\mathbb{N}}\sum_{i\in[k]} M_{ti}$.

For this setting, we consider BFDN as specified in Algorithm 1, with the minor modification that at each round t the only robots taking part in the assignment process are those which are allowed to move. More precisely, we replace the `for` loop of Algorithm 1 (**for** $i \in \{1, \dots, k\}$ **do**) with an iteration over all robots that may move (**for** $i \in \{i : M_{ti} = 1\}$ **do**). This modification is introduced to ensure that when multiple robots are at the same location, blocked robots do not prevent unblocked robots from traversing dangling edges.

▶ **Proposition 7.** *For any sequence of allowed moves $\mathbb{M} \in \{0,1\}^{\mathbb{N}\times[k]}$ satisfying $A(\mathbb{M}) \geq \frac{2n}{k} + D^2(\log(k) + 3)$ all edges of the tree will be visited by the above variant of* BFDN.

Proof. Again, the proof is very similar to that of Theorem 1 and all claims 1-5 all naturally adapt to this setting. As an example, we adapt the third claim as follows.

▶ **Claim 3** (Restated). *Consider a sequence of moves by some $\texttt{Robot}_i$ that starts at the root with the assignment of an anchor v of depth $\delta(v) = d$ and that ends with the return of $\texttt{Robot}_i$ to the root after T_x allowed moves of $\texttt{Robot}_i$. In this sequence, $\texttt{Robot}_i$ has explored exactly $(\mathsf{T}_x - 2d)/2$ dangling edges.*

The adversarial nature of the urns and balls game of Section 3 makes it applicable to the present setup, and Lemma 2 straightforwardly holds except for the $\log(\Delta)$ guarantee. Indeed, the adversary could choose to block all robots at a specific anchor until all k robots reach that anchor, which happens after at most $k(\log(k)+3)$ anchor assignments. ◀

▶ Remark 8. Other adversarial settings could be considered, for instance with an adversary that observes the moves that the robots have selected before choosing which robots to block. Another extension of interest would consist in relaxing the slotted time assumption to consider instead continuous time evolution, which could capture more realistic scenarios.

4.3 Collaborative exploration of non-tree graphs

The algorithm `BFDN` described above can be executed on any graph if it undergoes a minor modification: that any robot traversing on a dangling edge and arriving on a node explored earlier by another robot should go back from where it came and "close" the corresponding edge (this edge will never be used again). A similar technique was already proposed by [1] to adapt the algorithm of [10] to graphs. Unfortunately, without further assumption, the guarantees of `BFDN` do not generalize to graphs with n *edges* and *radius* D, where the radius is defined as the maximum distance between a node and the origin of the robots.

We therefore make the additional assumption that at any given node, a robot knows its distance to the origin in the underlying graph. Though restrictive, this assumption holds in some contexts of interest. It is for instance satisfied for the exploration of grid graphs with rectangular obstacles considered in [12] because the distance of any node with coordinates $(i,j)\in\mathbb{N}^2$ to the origin is equal to the so-called Manhattan distance $i+j$.

In that context, consider the following variant of `BFDN`: a robot traversing a dangling edge e will backtrack and "close" this edge if either of these two conditions is satisfied: (1) e led to a node that is already explored (2) e led to a node that is not strictly further to the origin than its first endpoint. In the case of (2), the node that is reached by the edge over which the respective robot backtracks is not considered as explored.

▶ **Proposition 9.** *Given a graph $G=(V,E)$ with n edges, diameter D and maximum degree Δ, assuming that the k robots are aware at all times of their distance to the origin and implement the above variant of* `BFDN`*, collaborative graph exploration is completed in at most $\frac{2n}{k}+D^2(\min\{\log(\Delta),\log(k)\}+3)$ rounds.*

Proof. It is clear that at the end of the execution of this algorithm, the edges which have never been closed form breadth-first tree of the graph with depth D. This tree is explored efficiently by `BFDN` while other edges are traversed at most twice by a single robot (or once by two robots, each coming from both endpoints, that will swap their identities). This leads to a total runtime of at most $\frac{2n}{k}+D^2(\min\{\log(\Delta),\log(k)\}+3)$. ◀

5 Recursive Algorithms for Improved Dependence on Depth D

In this section we develop a general recursive construction of so-called *anchor-based algorithms* which, applied to `BFDN`, yields the following result. It can be seen as a generalization of Theorem 1 as, for $\ell=1$, it provides the same upper-bound up to a factor 4.

▶ **Theorem 10.** *For any integer $\ell\geq 1$, $BFDN_\ell$, an associated recursive version of BFDN, explores a tree with n nodes, depth D, maximum degree Δ with k robots in $\frac{4n}{k^{1/\ell}}+2^{\ell+1}(\ell+1+\min\{\log(\Delta),\log(k)/\ell\})\,D^{1+1/\ell}$ rounds.*

To describe our recursive construction we need the following definitions. Given a node v in a tree T, $P_T[v]$ denotes the path from v to the root of T, and $P_T(v) = P_T[v] \setminus \{v\}$. Given two nodes u, v in a tree T, $\mathrm{LCA}_T(u, v)$ denotes their lowest common ancestor in T. We say that a explored node is *open* as long as it has at least one dangling adjacent edge. We say that it is *closed* as soon as a robot has traversed its last dangling edge. Note that open nodes are the parents of dangling edges. We decompose the exploration of an edge into two edge events as follows. An *edge event* occurs when a robot traverses an edge from parent to child for the first time, or when a robot traverses an edge from child to parent for the first time. There are thus at most $2(n-1)$ edge events in any exploration. Edges for which only one event has occurred are said to be *half explored*.

Anchor-based algorithm. Given k robots, an activity parameter $k^* \in [k]$, and a depth d, an *anchor-based* algorithm $\mathcal{A}(k^*, k, d)$ is by definition an exploration algorithm by k robots meeting the following requirements. Each robot is in one of the two states *active* or *inactive*. Each active robot i is assigned to a node v_i of the tree called its *anchor*. The algorithm must explore the tree so as to bring anchors at depth d while maintaining a list of invariants. The full list of so-called "Anchor-based invariants" is given in Appendix B. It mainly includes a variant of Claim 4 called *Open Node Coverage* which specifies that all open nodes must always be in $\cup_{i\in A} T(v_i)$ where A is the set of active robots. Other invariants mainly specify properties of the positions of the robots with respect to the partially explored tree and ensure that we can start an execution of an anchor-based algorithm after having interrupted the execution of another anchor-based algorithm.

Initially, the algorithm starts from any partially explored tree, with all robots active and anchored at the root. Robots must be in so-called *Parallel DFS Positions*, a requirement ensuring that all invariants are initially satisfied (see Appendix B). Active robots are allowed to move and explore the tree while inactive robots must be at depth at most d and wait. We distinguish two phases in the execution of the algorithm. As long as some anchor is at depth less than d or is not closed, we say that the algorithm runs *shallow*. During this first "shallow" phase, the algorithm must have at least k^* active robots at all rounds. When all anchors are at depth d and are all closed, we say that the algorithm runs *deep*. In this second "deep" phase, it is required that all active robots trigger an edge event at each round. However, the number of active robots may get below k^* during that phase. At any round, the algorithm may turn a robot into inactive or active as long as the requirements for the two phases are met. Finally, the algorithm can terminate when all robots are inactive. The Open Node Coverage invariant implies that the tree is then completely explored (see Appendix B).

Divide depth functor. We now define the *divide depth functor* $\mathcal{D}$, a map that takes an anchor-based algorithm and transforms it into another anchor-based algorithm as follows. Given an anchor-based algorithm $\mathcal{A}(k^*, k', d')$, a number n_{team} of teams and a number n_{iter} of iterations, we construct the exploration algorithm $\mathcal{D}[\mathcal{A}(k^*, k', d'); n_{team}; n_{iter}]$ for terminating the exploration of a partially explored tree. It uses $k = n_{team}k'$ robots for exploring the tree up to depth $d = n_{iter}d'$ in n_{iter} iterations where each iteration makes anchors progress d' deeper. More precisely, the i-th iteration runs parallel instances of $\mathcal{A}(k^*, k', d')$ in at most n_{team} sub-trees rooted at nodes with depth $(i-1)d'$. We assume that the previous iteration has terminated with a set R of at most $k^* \le n_{team}$ anchors at depth $(i-1)d'$. Relying on the Open Node Coverage invariant, we then restrict the exploration to the sub-trees rooted in R. Robots are thus partitioned into n_{team} teams of k' robots each. Each node $r \in R$ is taken in charge by a distinct team which runs an instance $\mathcal{A}_r(k^*, k', d')$

of $\mathcal{A}(k^*, k', d')$ on $T(r)$. When $|R| < n_{team}$, all robots in unassigned teams are inactive and wait at their position until the end of the current iteration. All other teams explore in parallel their sub-trees. We interrupt all running instances simultaneously when the overall number of active robots gets below k^* so that we can use their anchors as roots in the next iteration. As any single instance has activity parameter k^* this cannot happen until all anchors are at depth d' in each sub-tree, that is depth $i \cdot d'$ in T. After n_{iter} iterations, this guarantees that all nodes up to depth d have been closed and that exploration finally continues in at most k^* sub-trees rooted at depth d. See Appendix C for a formal description of the resulting anchor-based algorithm $\mathcal{B}(k^*, k, d) = \mathcal{D}[\mathcal{A}(k^*, k', d'); n_{team}; n_{iter}]$.

We say that an anchor-based algorithm $\mathcal{A}(k^*, k, d)$ has *f-shallow efficiency* for parameter f if it triggers at least $k^*(\mathsf{T} - f)$ edge events when running shallow during T rounds where parameter f may depend on k and d. We then have the following

▶ **Proposition 11.** *Given an anchor-based algorithm $\mathcal{A}(k^*, k', d')$, integers $n_{team} \geq k^*$ and $n_{iter} \geq 1$, $\mathcal{D}[\mathcal{A}(k^*, k', d'); n_{team}; n_{iter}]$ is correct and it is an anchor-based exploration algorithm $\mathcal{B}(k^*, k, d)$ for $k = n_{team}k'$ robots with depth $d = n_{iter}d'$. If moreover $\mathcal{A}(k^*, k', d')$ has f'-shallow efficiency, then $\mathcal{D}[\mathcal{A}(k^*, k', d'); n_{team}; n_{iter}]$ has f-shallow efficiency with $f = n_{iter}f' + n_{iter}^2 d' = n_{iter}(f' + d)$.*

Its proof is deferred to Appendix C. The reason for f-shallow efficiency is the following. Consider the i-th iteration of $\mathcal{D}_{\mathcal{A},k',d'}(k^*, k, d)$. Moving robots towards their associated root takes $2(i-1)d'$ rounds. Now, count the number T^1 of rounds where at least one of the instances has not run deep. As such an instance has run shallow during T^1 rounds, it has triggered at least $k^*(\mathsf{T}^1 - f')$ edge events by f'-shallow efficiency of $\mathcal{A}(k^*, k, d)$. During the remaining T^2 rounds of the iteration, all instances run deep. As this continues as long as k^* robots or more are active, at least k^* edge events are triggered per round, that is $k^*\mathsf{T}^2$ or more in total. Letting $\mathsf{T}_i = 2(i-1)d' + \mathsf{T}^1 + \mathsf{T}^2$ denote the number of rounds spent in the ith iteration, the number of edge events triggered during that iteration is thus at least $k^*(\mathsf{T}_i - f' - 2(i-1)d')$. The algorithm runs shallow during the n_{iter} iterations which last overall $\mathsf{T} = \sum_{i=1}^{n_{iter}} \mathsf{T}_i$. By summation, we get that it then triggers at least $k^*(\mathsf{T} - n_{iter}f' - n_{iter}^2 d')$ edge events as $\sum_{i=1}^{n_{iter}}(i-1) < n_{iter}^2/2$.

BFDN. Our first candidate for applying the divide depth functor is the following variant of Algorithm 1, denoted, $\texttt{BFDN}_1(k, k, d)$, where the procedure `Reanchor` is modified for assigning anchors at depth at most d. Precisely, we replace Line 26 with:

$$U = \{v \in V \;\; s.t. \;\; v \text{ is adjacent to some unexplored edge and } \delta(v) \text{ is minimal and } \delta(v) \leq d\}.$$

Note that this modification implies that when there are no more dangling edges at depth at most d, robots start to be anchored to the root and are then considered as inactive. Note that according to Claim 5 for depth $d+1$, there still remains exactly one robot in each sub-tree rooted at depth $d+1$ which is not entirely explored. These robots remain active until they have completely explored their sub-tree. $\texttt{BFDN}_1(k, k, d)$ thus terminates only when the tree has been fully explored. We also slightly modify the anchoring of robots: when a robot i is anchored at v_i it might happen that there are no more dangling edges at depth $\delta(v_i)$ or less thanks to the exploration of other robots. If this happens when $v_i \in P(u_i)$ and $\delta(v_i) < d$, we re-anchor robot i at the children of v_i in $P[u_i]$. This modification does not change the movements of robot i as it is then in a sequence of depth-next moves and will go up when reaching v_i anyway. However, this modification will ensure the preservation of the Partial Exploration invariant defined in Appendix B. It also implies that when there are no more dangling edges at depth at most d, all anchors are then at depth d.

One can then easily check that $\texttt{BFDN}_1(k,k,d)$ is an anchor-based algorithm. For example, the Open Node Coverage invariant is shown as Claim 4; see Appendix B for more details. We also note that $\texttt{BFDN}_1(k,k,d)$ has $c_1(k)d^2$-shallow efficiency where $c_1(k) = \min\{\log\Delta, \log k\}+2$. Indeed, $\texttt{BFDN}_1(k,k,d)$ runs exactly as Algorithm 1 as long as there are dangling edges at depth at most d, that is as long as the algorithm is running shallow. If this phase lasts T rounds, it triggers at least $k(\mathsf{T} - c_1(k)d^2)$ edge events. The proof is similar to that of Theorem 1 using Lemma 2 with the slight subtlety that we count edge events. The reason is that when starting from a partially explored tree where robots are in Parallel DFS Positions, the moves when robots go up still trigger edge events although no new edge may be discovered.

The $\texttt{BFDN}_\ell(k^*,k,d)$ anchor-based algorithm. We construct recursively a series of algorithms $\texttt{BFDN}_\ell(k^{1/\ell},k,d)$ for $\ell \geq 1$ as follows. Assuming that k and d are both ℓ-th powers of integers, we define for $\ell \geq 2$ the algorithm $\texttt{BFDN}_\ell(k^*,k,d) := \mathcal{D}[\texttt{BFDN}_{\ell-1}(k^*, k/n_{team}, d/n_{iter}); n_{team}; n_{iter}]$ with $k^* = n_{team} = k^{1/\ell}$ and $n_{iter} = d^{1/\ell}$. We let $k' = k/n_{team} = k^{(\ell-1)/\ell}$ and $d' = d/n_{iter} = d^{(\ell-1)/\ell}$ denote the parameters used for $\texttt{BFDN}_{\ell-1}$. Note that k' and d' are both $(\ell-1)$-th powers of integers and recursive calls all have integer-valued parameters. The activity parameter of instances $\texttt{BFDN}_{\ell-1}(k^*,k',d')$ indeed satisfies $(k')^{1/(\ell-1)} = k^{1/\ell} = k^*$. As we use $n_{team} = k^*$, we indeed respect the constraint $k^* \leq n_{team}$. We can bound its shallow efficiency according to the following statement:

▶ **Lemma 12.** *Given an integer $\ell \geq 2$, two integers k and d that are both ℓth powers of integers, $\texttt{BFDN}_\ell(k^{1/\ell},k,d)$ is $c_\ell(k)d^{1+1/\ell}$-shallow efficient with $c_\ell(k) = c_1(k^{1/\ell}) + \ell - 1$.*

Proof. As $\texttt{BFDN}_1(k^{1/\ell},k^{1/\ell},d^{1/\ell})$ is $c_1(k^{1/\ell})d^{2/\ell}$-shallow efficient, by induction (Proposition 11) $\texttt{BFDN}_j(k^{1/\ell},k^{j/\ell},d^{j/\ell})$ is $(c_1(k^{1/\ell})+j-1)d^{(j+1)/\ell}$-shallow efficient for $j = 2,\ldots,\ell$. ◀

▶ **Definition 13** (of $\texttt{BFDN}_\ell$)**.** *If k is the ℓ-th power of an integer, consider the sequence of depths $d_j = 2^{j\ell}$ for $j = 1,2,\ldots$ Algorithm $\texttt{BFDN}_\ell$ consists in running $\texttt{BFDN}_\ell(k^{1/\ell},k,d_1)$, interrupting it right after its last iteration (without running deep further), then running $\texttt{BFDN}_\ell(k^{1/\ell},k,d_2)$ with the current robot positions and anchor assignments until its last iteration finishes, and so on. When running $\texttt{BFDN}_\ell(k^{1/\ell},k,d_j)$ with $j = \lceil \frac{\log_2 D}{\ell} \rceil$, all anchors reach depth D and the algorithm terminates. If k is not an integer to the power ℓ, we use $K = \lfloor k^{1/\ell} \rfloor^\ell \leq k$.*

Proof. (of Theorem 10) Assume first that k is the ℓ-th power of some integer. In a run of $\texttt{BFDN}_\ell$, denote by T_j the number of rounds that the call to $\texttt{BFDN}_\ell(k^{1/\ell},k,d_j)$ lasts. This call triggers at least $k^{1/\ell}(\mathsf{T}_j - c_\ell(k)d_j^{1+1/\ell})$ edge events by applying Lemma 12. We can thus bound the overall running time $\mathsf{T} = \sum_{j=1}^{\lceil (\log_2 D)/\ell \rceil} \mathsf{T}_j$ by summing over all calls: $2n \geq k^{1/\ell}\left(\mathsf{T} - c_\ell(k)\sum_{j=1}^{\lceil (\log_2 D)/\ell \rceil} d_j^{1+1/\ell}\right)$. As we have $\sum_{j=1}^{\lceil (\log_2 D)/\ell \rceil} d_j^{1+1/\ell} = \sum_{j=1}^{\lceil (\log_2 D)/\ell \rceil} 2^{(\ell+1)j} \leq \frac{2^{(\ell+1)((\log_2 D)/\ell+2)}-1}{2^{\ell+1}-1} \leq 2^{\ell+1}D^{1+1/\ell}$, we obtain $\mathsf{T} \leq \frac{2n}{k^{1/\ell}} + 2^{\ell+1}c_\ell(k)D^{1+1/\ell}$. For arbitrary k, with $K = \lfloor k^{1/\ell} \rfloor^\ell$, using $K^{1/\ell} \geq k^{1/\ell}/2$, we obtain a time bound of $\mathsf{T} \leq \frac{4n}{k^{1/\ell}} + 2^{\ell+1}(\ell - 1 + c_1(k^{1/\ell}))D^{1+1/\ell}$, yielding the runtime bound announced in Theorem 10 since $c_1(k^{1/\ell}) = 2 + \min\{\log(\Delta), \log(k)/\ell\}$. ◀

References

1 Peter Brass, Flavio Cabrera-Mora, Andrea Gasparri, and Jizhong Xiao. Multirobot tree and graph exploration. *IEEE Trans. Robotics*, 27(4):707–717, 2011. doi:10.1109/TRO.2011.2121170.

2 Peter Brass, Ivo Vigan, and Ning Xu. Improved analysis of a multirobot graph exploration strategy. In *13th International Conference on Control Automation Robotics & Vision, ICARCV 2014, Singapore, December 10-12, 2014*, pages 1906–1910. IEEE, 2014. doi:10.1109/ICARCV.2014.7064607.

3 Romain Cosson, Laurent Massoulié, and Laurent Viennot. Breadth-first depth-next: Optimal collaborative exploration of trees with low diameter. *arXiv preprint arXiv:2301.13307*, 2023.

4 Romain Cosson, Laurent Massoulié, and Laurent Viennot. Brief announcement: Efficient collaborative tree exploration with breadth-first depth-next. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 24–27. ACM, 2023. doi:10.1145/3583668.3594568.

5 Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pajak, and Przemyslaw Uznanski. Fast collaborative graph exploration. *Inf. Comput.*, 243:37–49, 2015. doi:10.1016/j.ic.2014.12.005.

6 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Skoric, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theor. Comput. Sci.*, 811:70–78, 2020. doi:10.1016/j.tcs.2018.03.006.

7 Miroslaw Dynia, Miroslaw Korzeniowski, and Christian Schindelhauer. Power-aware collective tree exploration. In Werner Grass, Bernhard Sick, and Klaus Waldschmidt, editors, *Architecture of Computing Systems - ARCS 2006, 19th International Conference, Frankfurt/Main, Germany, March 13-16, 2006, Proceedings*, volume 3894 of *Lecture Notes in Computer Science*, pages 341–351. Springer, 2006. doi:10.1007/11682127_24.

8 Miroslaw Dynia, Jaroslaw Kutylowski, Friedhelm Meyer auf der Heide, and Christian Schindelhauer. Smart robot teams exploring sparse trees. In Rastislav Kralovic and Pawel Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, volume 4162 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2006. doi:10.1007/11821069_29.

9 Miroslaw Dynia, Jakub Lopuszanski, and Christian Schindelhauer. Why robots need maps. In Giuseppe Prencipe and Shmuel Zaks, editors, *Structural Information and Communication Complexity, 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007, Proceedings*, volume 4474 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2007. doi:10.1007/978-3-540-72951-8_5.

10 Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006. doi:10.1002/net.20127.

11 Yuya Higashikawa, Naoki Katoh, Stefan Langerman, and Shin-ichi Tanigawa. Online graph exploration algorithms for cycles and trees by multiple searchers. *J. Comb. Optim.*, 28(2):480–495, 2014. doi:10.1007/s10878-012-9571-y.

12 Christian Ortolf and Christian Schindelhauer. Online multi-robot exploration of grid graphs with rectangular obstacles. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 27–36. ACM, 2012. doi:10.1145/2312005.2312010.

13 Christian Ortolf and Christian Schindelhauer. A recursive approach to multi-robot exploration of trees. In Magnús M. Halldórsson, editor, *Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, volume 8576 of *Lecture Notes in Computer Science*, pages 343–354. Springer, 2014. doi:10.1007/978-3-319-09620-9_26.

A Comparisons between Algorithms CTE, Yo* and BFDN

We provided in Figure 1 a picture of how BFDN compares in terms of runtime with other state-of-the art algorithms for collaborative tree exploration. The regions are defined up to multiplicative constants that only depend on k. We included in the figure only algorithms requiring no assumptions on the tree structure. Four algorithms thus appear in the figure: the original "collaborative tree exploration" CTE algorithm of [10] with runtime $\mathcal{O}(\frac{n}{\log(k)} + D)$, the recursive algorithm Yo* of [13] with runtime $\mathcal{O}(2^{\mathcal{O}(\sqrt{\log D \log\log k})} \log k(\log n + \log k)(n/k + D))$, which we reduced to smaller quantities to simplify the picture, BFDN with runtime $2n/k + D^2 \log(k)$ as well as its recursive variant $\texttt{BFDN}_\ell$.

Figure 1 highlights that BFDN is the only algorithm to outperform CTE of [10] in an unbounded range of parameters (n, D). Indeed, the other competitor, Yo*, is outperformed by CTE when $n \geq e^k$ or when $D \geq e^{\log(k)^2}$. Yet, CTE remains the most efficient algorithm for trees with small depth. We detail below the calculations that led to Figure 1.

Comparison between BFDN and CTE. Since the runtime of any collaborative tree algorithm exceeds n/k and D, it is sufficient to compare the suboptimal terms of both algorithms which are $D^2 \log(k)$ and $n/\log(k)$ for BFDN and CTE respectively. It therefore turns out that BFDN is faster than CTE in the range $D^2 \log(k)^2 \leq n$.

Comparison between CTE and Yo*. First, we simplified the runtime of Yo* to $\mathcal{O}(\log(n)n/k + D)$, which gives that it can outperform the $\mathcal{O}(n/\log(k) + D)$ of [10] only in the range $n \leq e^{k/\log(k)}$ which we extend to $n \leq e^k$ in the picture. After, we simplified the runtime of Yo* to $\mathcal{O}(e^{\sqrt{\log(D)}} n/k + D)$ to obtain the range $D \leq e^{\log(k)^2}$. Finally, we simplified the runtime of Yo* to $D \log(n) \log(k)$ to get that CTE outperforms Yo* for trees satisfying $D \geq \frac{n}{\log(n)} \log(k)^2$.

Comparison between BFDN and Yo*. We used the comparisons above for $e^k \leq n$ or $e^{\log(k)^2} \leq D$, and completed by the following simplification of the runtime of Yo* to $\mathcal{O}(\log(k)n/k + D)$. BFDN is thus faster than Yo* when $\log(k)D^2 \leq \log(k)n/k$, that is when $kD^2 \leq n/k$.

Comparison between $\texttt{BFDN}_\ell$ and CTE. We note that $\texttt{BFDN}_\ell$ may outperform CTE only if $k^{1/\ell} > \log(k)$, or equivalently if $\ell < \frac{\log(k)}{\log(\log(k))}$, which we assumed in the caption of the Figure. Under this condition, $\texttt{BFDN}_\ell$ outperforms CTE if $2^\ell \log(k) D^{1+1/\ell} < \frac{n}{\log(k)}$. Since we have $2^\ell < k$, this condition is met if $D < \frac{1}{k \log(k)^2} n^{\ell/(\ell+1)}$.

Comparison between $\texttt{BFDN}_\ell$ and BFDN. If $n/k > D^2$, if is clear that BFDN outperforms $\texttt{BFDN}_\ell$. On the other hand, if $n/k^{1/\ell} < D^2$, $\texttt{BFDN}_\ell$ outperforms BFDN.

B Formal description of Anchor-based Invariants

During the execution of an anchor-based algorithm, it is required that the partially explored tree, the set $A \subseteq [k]$ of active robots, the anchor assignment $(v_i)_{i \in A}$, and the positions $(u_i)_{i \in [k]}$ of the robots always satisfy the following invariants:

- all open nodes of the currently explored tree are in $\cup_{i \in [k]} P_T[u_i]$, (DFS Open Coverage)
- for any two robots $i \neq j$, all nodes in $P_T(\mathrm{LCA}_T(u_i, u_j))$ are closed, (Parallel Positions)

- for all active robot i such that $v_i \in P_T[u_i]$, all edges in the path from v_i to u_i are half explored, (Partial Exploration)
- for all active robot $i \in A$, $\delta(v_i) \leq d$, (Limited Anchor Depth)
- all inactive robots are located at depth at most d, (Inactive Depth)
- all open nodes of the currently explored tree are in $\cup_{i \in A} T(v_i)$, (Open Node Coverage)
- if $\exists i \in A$ such that either $\delta(v_i) < d$ or v_i is open, then at least k^* robots are active, (Shallow Activity)
- if all anchors $\{v_i : i \in A\}$ are at depth d and are close, each active robot triggers an edge event at each round. (Deep Activity)

Initially, robots are said to be in *Parallel DFS Positions* when DFS Open Coverage, Parallel Positions and Partial Exploration are all three satisfied when assuming that all robots are active and anchored at the root. One can easily check that other invariants are then also satisfied.

Properties of an anchor-based algorithm. The Open Node Coverage invariant implies that all nodes at depth less than d' are closed where $d' = \min_{i \in A} \delta(v_i)$ is the minimum depth of an anchor. The Shallow Activity invariant implies that the number of active robots may decrease below k^* only when all anchors are at depth d and consequently when all nodes up to depth d are closed. The Open Node Coverage invariant also implies that for any dangling edge adjacent to a explored node w, there exists at least one active robot i such that w is in $T(v_i)$. This implies that if all anchors are at depth d and if i is the last robot with anchor v_i, it cannot become inactive unless $T(v_i)$ has been completely explored. This indeed implies that the algorithm cannot terminate unless the full tree has been completely explored: as long as there remains an open node w, some robot i must be active with an ancestor of w as anchor. Recall that we require that the algorithm cannot terminate unless all robots are inactive.

BFDN $\texttt{BFDN}_1(k, k, d)$ is an anchor-based algorithm. Indeed, the Open Node Coverage invariant is shown as Claim 4; the DFS Open Coverage and Partial Exploration invariants come from the similarity of `DN` moves with a DFS traversal, while the Parallel Positions invariant comes from the selection of distinct dangling edges when several robots are located at the same node. The Limited Anchor Depth and Inactive Depth invariants are satisfied by the modification of anchor selection. The Shallow Activity invariant comes from the fact that all robots are active as long as there remain some dangling edge at depth at most d. Finally, the Deep Efficiency invariant comes from Claim 5 as when the algorithm runs deep, each sub-tree at depth $d + 1$ which is not completely explored contains exactly one robot performing a DFS-like traversal of the sub-tree.

We also note that we can start $\texttt{BFDN}_1(k, k, d)$ from any partially explored tree where robots are in Parallel DFS Positions as long as each robot i, which is in a position u_i with open ancestors, gets anchored to a node v_i of $P[u_i]$ such that all nodes of $P(v_i)$ are closed. Such a situation occurs in `BFDN` when a robot is performing `DN` moves. It is thus possible to start a robot in any such situation so that it will then behave similarly as in `BFDN`. The other robots see only closed nodes and thus get to the root according to Algorithm 1 where they get re-anchored.

C Divide-depth Algorithm

Algorithm 3 Divide depth algorithm $\mathcal{D}[\mathcal{A}(k^*, k', d'); n_{team}; n_{iter}]$

Require: An anchor-based exploration algorithm $\mathcal{A}(k^*, k', d')$, integers $n_{team} \geq k^*$ and $n_{iter} \geq 1$, a partially explored tree T with $k = n_{team}k'$ robots in Parallel DFS Positions and such that at most k^* robots are at depth greater than 0.

Ensure: All nodes are explored and closed.

1: $R \leftarrow \{\texttt{root}(T)\}$ ▷ Set of sub-tree roots in next iteration.
2: $A \leftarrow \{i \in [k] : u_i \neq \texttt{root}(T)\}$ ▷ Set of robots having already progressed in T.
3: All robots are active and have $\texttt{root}(T)$ as anchor.
4: **for** $i = 1, \ldots, d/d'$ **do**
5: ▷ Iteration i:
6: For all $r \in R$, let $k_r = |\{i \in A : v_i = r\}|$ be the number of robots having progressed in $T(r)$.
7: Partition robots into $|R|$ teams $(B_r)_{r \in R}$ of k' robots each, one per node $r \in R$:
8: each robot $i \in A$ is assigned to v_i,
9: for all $r \in R$, $k' - k_r$ robots in $[k] \setminus A$ are assigned to r. ▷ We rely on $k_r \leq k'$ and $|R| \leq n_{team}$.
10: All robots in team B_r are assigned to anchor r: we set $v_i \leftarrow r$ for all $i \in B_r \setminus A$.
11: All robots in $\cup_{r \in R} B_r \setminus A$ are turned to active, and move to their anchor in $2(i-1)d'$ rounds. ▷ Moves for rebalancing robots.
12: All robots in $[k] \setminus \cup_{r \in R} B_r$ are turned to inactive and wait at their current position.
13: Each team associated to $r \in R$ initializes independently an instance $\mathcal{A}_r(k^*, k', d')$ for exploring $T(r)$.
14: At any round, we let A_r denote the set of active robots among the team exploring $T(r)$.
15: **while** $|\cup_{r \in R} A_r| \geq k^*$ **do**
16: Run in parallel one round of all instances $\mathcal{A}_r(k^*, k', d')$ for $r \in R$.
17: **end while**
18: $A \leftarrow |\cup_{r \in R} A_r|$ ▷ Overall set of active robots.
19: $R \leftarrow \{v_i : i \in A\}$ ▷ Roots of sub-trees not fully explored yet.
20: Continue running instances $\mathcal{A}_r(k^*, k', d')$ of the last iteration for all $r \in R$. ▷ Running deep.

Proof of Proposition 11. We first check that all invariants are preserved by induction on the iteration number i. The main argument is that all anchors are at depth $i \cdot d'$ after Iteration i. We require that the DFS Open Coverage, Parallel DFS Positions and Partial Exploration invariants are satisfied by the initial positions of robots. All remaining invariants are also satisfied as the only initial anchor is at depth zero. Assume that all invariants are satisfied up to the beginning of Iteration i, and that nodes in R are at depth $(i-1)d'$.

The Inactive Depth invariant ensures that inactive robots at the end of the previous iteration are at depth $(i-1)d'$ or less, and moving them according to Line 11 can indeed be done within $2(i-1)d'$ rounds. Moreover, the Open Node Coverage invariant ensures that all nodes at depth less than $(i-1)d'$ are closed, and these movements preserve the DFS Open Coverage and Parallel Positions invariants. The Partial Exploration invariant is also preserved since these robots are not located in the sub-tree of their anchor. These $(i-1)d'$ rounds also preserve Anchor Depth and Open Node Coverage invariants as the anchors R

of nodes active in the last round of the previous iteration remain their anchor, while other nodes are assigned to one of the anchors in R.

The fact that robots are initially in Parallel DFS Positions in each instance $\mathcal{A}_r(k^*, k', d')$ for $r \in R$ comes from the preservation of the DFS Open Coverage, Parallel Positions, and Partial Exploration invariants at the end of the previous round as the root r was the anchor of robots that are not located at r. Now, as all instances $\mathcal{A}_r(k^*, k', d')$ for $r \in R$ run in disjoint sub-trees, the DFS Open Coverage, Parallel Positions, Partial Exploration, Anchor Depth and Open Node Coverage invariants are also preserved during the rest of the iteration since each $\mathcal{A}_r(k^*, k', d')$ is anchor-based. Similarly, the Inactive Depth invariant is satisfied as its variant in instances $\mathcal{A}_r(k^*, k', d')$ imply that inactive nodes are at depth $(i-1)d' + d' = i \cdot d' \leq d$ at most. The Shallow Activity invariant is preserved as long as at least one instance $\mathcal{A}_r(k^*, k', d')$ is not running deep according to the Shallow Activity invariant for that instance. This means that the number of overall active robots can drop below k^* only when all instances are running deep, implying that all anchors are then at depth $(i-1)d' + d' = i \cdot d'$. Note that the Open Node Coverage invariant then implies that all open nodes are in the sub-trees rooted at the anchors of the robots that were active in the last round. The exploration can thus be reduced to these at most k^* sub-trees as claimed in the description of the divide depth functor.

Finally, the algorithm starts running deep only when all anchors are at depth d and are all closed. This can happen only towards the end of the last iteration when all instances are running deep. The reason is that if an instance is not running deep, it has at least k^* active robots by the Shallow Activity invariant and the termination condition of the inner while loop at Line 15 is not met. The Deep Activity invariant then follows from the fact that instances are running in pairwise disjoint sub-trees and all satisfy the Deep Activity invariant.

This completes the proof that $\mathcal{D}[\mathcal{A}(k^*, k', d'); n_{team}; n_{iter}]$ is correct and that it is an anchor-based exploration algorithm.

The proof for f-shallow efficiency is given in Section 5. ◀