Remembering without Memory:
Tree Exploration
by Asynchronous Oblivious Robots

Paola Flocchini[1,*], David Ilcinkas[2,**], Andrzej Pelc[3,*], and Nicola Santoro[4,*]

[1] University of Ottawa, Canada
`flocchin@site.uottawa.ca`
[2] CNRS, Université Bordeaux I, France
`david.ilcinkas@labri.fr`
[3] Université du Québec en Outaouais, Canada
`pelc@uqo.ca`
[4] Carleton University, Canada
`santoro@scs.carleton.ca`

Abstract. In the effort to understand the algorithmic limitations of computing by a swarm of robots, the research has focused on the minimal capabilities that allow a problem to be solved. The weakest of the commonly used models is ASYNCH where the autonomous mobile robots, endowed with visibility sensors (but otherwise unable to communicate), operate in Look-Compute-Move cycles performed asynchronously for each robot. The robots are often assumed (or required to be) oblivious: they keep no memory of observations and computations made in previous cycles.

We consider the setting when the robots are dispersed in an anonymous and unlabeled graph, and they must perform the very basic task of *exploration*: within finite time every node must be visited by at least one robot and the robots must enter a quiescent state. The complexity measure of a solution is the number of robots used to perform the task.

We study the case when the graph is an arbitrary tree and establish some unexpected results. We first prove that there are n-node trees where $\Omega(n)$ robots are necessary; this holds even if the maximum degree is 4. On the other hand, we show that if the maximum degree is 3, it is possible to explore with only $O(\frac{\log n}{\log \log n})$ robots. The proof of the result is constructive. Finally, we prove that the size of the team is asymptotically *optimal*: we show that there are trees of degree 3 whose exploration requires $\Omega(\frac{\log n}{\log \log n})$ robots.

1   Introduction

An important goal of theoretical research on computing by autonomous mobile robots has been to understand the algorithmic limitations of computing in such settings. The research has thus focused on the minimal capabilities that allow a problem to be solved by a swarm of robots. In the investigations, three models are commonly used: SYNCH, SSYNCH, and ASYNCH; the fully synchronous model SYNCH being the strongest, the asynchronous model ASYNCH being the weakest, the semi-synchronous model SSYNCH lying in between (see, e.g., [2,3,4,5,9,13,14,15,16]).

In ASYNCH the autonomous mobile robots are endowed with visibility sensors (but otherwise unable to communicate), are anonymous, are oblivious, and operate in asynchronous Look-Compute-Move cycles. In one cycle, a robot uses its sensors to obtain a snapshot of the current configuration (Look); then, based on the perceived configuration, it computes a destination (Compute), and moves there (Move); if the destination is the current position, the robot is said to perform a null move. Cycles are performed asynchronously for each robot; this means that, even if the operations are instantaneous, the time between Look, Compute, and Move operations is finite but unbounded, and is decided by the adversary for each action of each robot. The robots are oblivious: they keep no memory of observations and computations made in previous cycles. All robots are identical and execute the same algorithm. It is usually assumed that robots are capable of multiplicity detection: during a Look operation, a robot can determine if at some location there are no robots, there is one robot, or there are more than one robots; however, in the latter case, the robot might not be capable of determining the exact number of robots.

The asynchrony implies that a robot $\mathcal{R}$ may observe the position of the robots at some time t; based on that observation, it may compute the destination at some time $t' > t$, and Move to its destination at an even later time $t'' > t'$; thus it might be possible that at time t'' some robots are in different positions from those previously perceived by $\mathcal{R}$ at time t, because in the meantime they performed their Move operations (possibly several times). Since robots are oblivious, i.e., they do not have any memory of past observations, the destination is decided by a robot during a Compute operation solely on the basis of the location of other robots perceived in the previous Look operation.

In the literature, the ASYNCH model is used by researchers in the study of the coordination and control of autonomous mobile robots in the two-dimensional plane, which we shall term the *continuous* scenario. The computational capabilities of these robots when the spacial universe is a network or a graph, a scenario that we shall term *discrete*, has been recently investigated in [8,11], where the graph is a ring. In the discrete scenario, the computed destination in each cycle is either the node where the robot is currently situated or a node adjacent to it.

An important feature of the discrete scenario is the fact that the graph is totally anonymous: not only nodes but also edges are unlabeled, and there are no port numbers at nodes. This gives additional power to the adversary at the time when a robot must move. Indeed, it may happen that two or more edges

incident to a node v currently occupied by the deciding robot look identical in the snapshot obtained during the last Look action , i.e., there is an automorphism of the tree which fixes v, carries empty nodes to empty nodes, occupied nodes to occupied nodes, and multiplicities to multiplicities, and carries one edge to the other. In this case, if the robot decides to take one of the ports corresponding to these edges, it may take any of the identically looking ports. We assume the worst-case decision in such cases, i.e., that the actual port among the identically looking ones is chosen by an adversary. This is a natural worst-case assumption and our algorithm is also resistant to such adversarial decisions.

We continue the study of computational capabilities of robots under the discrete scenario by considering the very basic task of *exploration*: within finite time every node must be visited by at least one robot and the robots must enter a quiescent state. The complexity measure of a solution is the number of robots used to perform the task. The problem of exploring a graph has been extensively studied in the literature under a variety of assumptions (e.g. see [1,6,7,10,12]) but not in the setting considered here. The only exception is [8] where we proved that the minimum number $\rho(n)$ of robots that can explore a ring of size n is $O(\log n)$ and that $\rho(n) = \Omega(\log n)$ for arbitrarily large n.

In this paper we consider the case when the graph is an arbitrary tree and establish some unexpected results. We first prove that, in general, exploration cannot be done efficiently. More precisely we prove that there are n-node trees where $\Omega(n)$ robots are necessary; this holds even if the maximum degree is 4. We then prove the existence of a *complexity gap*. We show that if the maximum degree of the tree is 3 then it is possible to explore it with only $O(\frac{\log n}{\log \log n})$ robots. The proof of the result is constructive. Finally, we show that the size of the team used in our solution is asymptotically *optimal*: there are trees of degree 3, whose exploration requires $\Omega(\frac{\log n}{\log \log n})$ robots.

Due to space limitations, the proofs are omitted.

2   Terminology and Preliminaries

We consider a n-node anonymous unoriented tree. Some nodes of the tree are occupied by robots. We will always assume that in an initial configuration of robots there is at most one robot in each node. The number of robots is denoted by k. A *complete d-ary tree* is a rooted tree, all of whose internal nodes have d children and all of whose leaves are at the same distance from the root. Nodes v and w are *similar* if there exists an automorphism of the tree T which carries v to w.

In order to formally define what a robot perceives during a Look action, we introduce the notion of the *view* of a rooted tree T occupied by robots, from its root v. This is defined by induction on the height of the tree T. If T consists only of v then $View(T, v) = (x, \emptyset)$, where $x = 0$, $x = 1$, or $x = *$, if there is 0, 1, or more than 1 robot in v, respectively. If T is of positive height, let $v_1, \ldots, v_m$ be children of the root v, and let $T_1, \ldots, T_m$ be subtrees rooted at $v_1, \ldots, v_m$, respectively . Then $View(T, v) = (x, \{View(T_1, v_1), \ldots, View(T_m, v_m)\})$, where x has the same meaning as before. Now, the snapshot taken by a robot located

at v is simply $View(T, v)$. This formalism captures two essential assumptions about the perceptions of robots. First, a robot can distinguish between nodes occupied by 0, 1, or more than 1 robot, but cannot distinguish between numbers larger than 1 of robots located at the same node. Second, subtrees rooted at children of a node are not ordered: this is captured by considering the set of respective views, and not their sequence, in the recursive definition.

Two robots located at nodes v and w are called *equivalent*, if $View(T, v) = View(T, w)$. A node that is not occupied by any robot is called *empty*. When a node is occupied by more than one robot, we say that there is a *tower* in this node. A robot that is not a part of a tower is called *free*.

An exploration algorithm is a function whose arguments are views, and whose value for any given view $View(T, v)$ is either v or the equivalence class of one of its neighbors, with respect to the following equivalence relation $\sim$: $w_1 \sim w_2$ if there exists an automorphism f of the tree which fixes v, carries empty nodes to empty nodes, free robots to free robots, towers to towers, and such that $f(w_1) = w_2$. Note that $w_1 \sim w_2$ is equivalent to $View(T, w_1) = View(T, w_2)$. If the equivalence class returned by the algorithm for some view has more than one element then the choice of the neighbor in this class to which the robot will actually move, belongs to the adversary. If the value is v, we say that the move of the robot for the given view is the *null move*.

We say that exploration of a n-node tree is possible with k robots, if there exists an algorithm which, starting from any initial configuration of k robots without towers, and for any behavior of the adversary controling asynchrony and choices between equivalent neighbors, explores the entire tree and brings all robots to a configuration in which they all remain idle, i.e., there exists a time t after which all nodes are explored and all subsequent moves of robots are null moves. In fact, our negative results hold even for this weak (implicit) stopping condition, and our positive results (algorithms) are valid even with the following stronger (explicit) stopping condition: for any execution of the algorithm, there exists a time t after which all nodes are explored, and each robot knows that no non-null move of any robot (including itself) will ever occur. Obviously, if $k = n$, the exploration is already accomplished, hence we always assume that $k < n$.

3   Exploration of Trees

In this section we prove that, in general, exploration of n-node trees might require $\Omega(n)$ robots (even if the max degree is 4); we prove that, on the other hand, the minimum number of robots sufficient to explore all n-node trees of maximum degree 3 is $\Theta(\frac{\log n}{\log \log n})$.

3.1   Exploration of Arbitrary Trees

We show that there are arbitrarily large trees of maximum degree 4 whose exploration requires $\Omega(n)$ robots.

Theorem 1. *Exploration of a n-node complete 3-ary tree requires $\Omega(n)$ robots.*

3.2   Exploration of Trees of Maximum Degree 3: Upper Bound

We prove the following upper bound on the size of the team of robots capable to explore all n-node trees of maximum degree 3.

Main Theorem. *There exists a team of $O(\frac{\log n}{\log \log n})$ robots that can explore all n-node trees of maximum degree 3, starting from any initial configuration.*

This result is proved by showing an exploration algorithm using $O(\frac{\log n}{\log \log n})$ robots.

Overview of the Algorithm. The main idea of the algorithm is the following. The entire tree is partitioned into two or three subtrees, the number of parts depending on the shape of the tree. Parts are explored one after another by a team of three robots that sequentially visit leaves of this part. Since individual robots do not have memory, a specially constructed, dynamic configuration of robots, called the "brain", keeps track of what has been done so far. More precisely, the brain counts the number of already visited leaves and indicates the next leaf to be visited. It is also the brain that requires most of the robots used in the exploration process. The reason why $\Theta(\log n / \log \log n)$ robots are sufficient for exploration, is that the counting process is efficiently organized. The counting module of the brain consists of disjoint paths of logarithmic lengths, which are appropriately marked by groups of robots of bounded size. Paths are of logarithmic lengths because longer paths cannot be guaranteed to exist in all trees of maximum degree 3. Inside each of these paths a tower moves, indicating a numerical value by its position in the path. The combination of these values yields the current value of the number of visited leaves. Since the number of leaves may be $\Theta(n)$, we need a number x of paths, which can produce $\Theta(n)$ combinations of values, i.e., such that $(\Theta(\log n))^x = \Theta(n)$. This is the reason of constructing $\Theta(\log n / \log \log n)$ paths and thus using $\Theta(\log n / \log \log n)$ robots. We show how to construct these paths in any tree of maximum degree 3, and how to organize the counting process. The latter is complicated by the asynchronous behavior of the robots. During the switch of the counter from value i to $i+1$ robots move in the paths and a snapshot taken during the transition period shows a "blurred" picture: the old value is already destroyed while the new one is not yet created. This could confuse the robots and disorganize the process. Thus we use two counters acting together. They both indicate value i, then one of them keeps this value and the other transits to $i+1$. When this is completed, the first counter transits to $i+1$ and so on. This precaution permits to keep track of the current value during the process of incrementation. During the exploration of one part of the tree, the brain is located in another part and controls exploration remotely. After completing the exploration of one part, the brain is moved to the already explored part in order to let the exploring agents visit the rest of the tree.

There are two main difficulties in our algorithm. The first is to break symmetries that can exist in configurations of robots, in order to let them act independently and reach appropriate target nodes, in particular during the construction

of the brain. The second challenge is the construction and relocation of the brain, as well as organizing its proper functioning by coordinating the two counters, regardless of the behavior of the adversary that controls asynchrony.

The algorithm is divided into the following phases. Phase 1 consists in moving all robots down the tree oriented in a specific way, without creating a tower, in order to create a large zone free of robots. When no robot can move further down, a tower is created to mark the end of Phase 1. In Phase 2, robots are moved from one part of the tree and create the brain in another part. If there are local symmetries, a leader is elected and breaks them by relocating to specific nodes of the tree. This is done to let the robots move independently from one part of the tree to another and occupy target positions. As a consequence, one part becomes almost empty, which facilitates its exploration. Phase 2 ends when the brain is at its place, properly initialized, and there remain only a tower and a free robot in the other part, that will explore this part. Phase 3 is the actual exploration of the part not containing the brain (or the larger of the two parts not containing the brain). This is done by visiting its leaves, one similarity class after another. Inside a similarity class, leaves are explored in a DFS manner, the brain keeping track of the progress of exploration. This phase ends when the brain indicates that the exploring robots are at the last leaf of the explored part. In Phase 4 the brain is relocated to the already explored part, and the exploring robots move to one of the unexplored parts. Again, Phase 4 ends when all robots are in their places and the brain is properly reinitialized (with the indication that one part is already explored). Finally, in the remaining phases the rest of the tree is explored, similarly as in Phase 3. There is a mechanism in the algorithm that enables robots to see what is the current phase, in order to avoid circular behavior. This is implemented by a special arrangement of robots, called signal, whose value increments from phase to phase.

Tools and Basic Properties. Before giving a detailed description of the algorithm we present some concepts that we will use in this description, and prove their basic properties. Let T be a n-node tree of maximum degree 3. Consider a team of k robots, where $c \log n / \log \log n \leq k \leq 2c \log n / \log \log n$, for an appropriately chosen constant c, and $k \equiv 5 \pmod 6$. The conditions on the constant c are explicitly given after the description of the algorithm.

Pieces

For each internal node v, consider the number of nodes in each of the subtrees rooted at neighbors of v, and let n_v be the maximum of these numbers. It is well known that either there exists exactly one node v for which $n_v \leq (n-1)/2$ (the centroid), or there is exactly one edge $\{v, w\}$, for which $n_v = n_w = n/2$ (the bicentroid). In each case we consider the oriented tree from the centroid or bicentroid down to the leaves. We will say that the tree is rooted in the centroid or bicentroid and use the usual notions of parent and children of a node.

Next we define the subtrees of T, called its *pieces*. If T has a centroid of degree 2 then there are two pieces T_1 and T_2 which are rooted at children of the centroid. If T has a centroid of degree 3 then there are three pieces T_1, T_2 and T_3 which are rooted at children of the centroid. Finally, if T has a

bicentroid then there are two pieces T_1 and T_2 which are rooted at nodes of the bicentroid. Without loss of generality we assume that sizes of T_1, T_2 and T_3 are non-increasing. Hence $(n-1)/3 \leq |T_1| \leq n/2$ and $n/4 \leq |T_2| \leq n/2$. For every piece, we define its *weight* as the number of robots located in it. Thus we talk about the heaviest piece, a heavier piece, etc. A piece T_i is called *unique* if there is no other piece whose root has the same view as the root of T_i.

Core Zone
A node in a piece is *a core node*, if the size of the subtree rooted at this node is larger than the size k of the entire team of robots. The set of core nodes in a piece is called the *core zone* of the piece.

Lemma 1. *In any rooted tree of size x and such that every internal node has at most two children, the size of the core zone is at least $\frac{x+1}{k+1} - 1$.*

Since the size of any of the two largest pieces is at least $n/4$, Lemma 1 implies that the size of the core zone of any of these pieces is at least $\frac{n \log \log n}{10 c \log n}$.

Descending Paths
The basic component of the brain is a *descending path*. This is a simple path in a piece Q, whose one extremity is its node closest to the root of Q. It will be called the beginning of the path. The other extremity will be called its end. The size of such a path is the number of its nodes. We need sufficiently many pairwise disjoint descending paths, each sufficiently long, for all parts of the brain. The construction is a part of the proof of the following lemma.

Lemma 2. *For any sufficiently large m, every tree of maximum degree 3 and of size m contains at least $\log^2 m$ pairwise disjoint descending paths of size at least $\frac{1}{4} \log m$.*

The core zone is a tree of maximum degree 3, rooted in the root of the piece and has size $m \geq \frac{n \log \log n}{10 c \log n}$. Hence, for sufficiently large n, Lemma 2 guarantees the existence of at least $\log^2 m \geq \log n \geq 5 \log n / \log \log n$ pairwise disjoint descending paths of size at least $\frac{1}{4} \log m \geq \frac{1}{8} \log n$ in any of the two largest pieces.

Modules of the Brain
The brain consists of four parts: two *counters*, the *semaphore* and the *garbage*. Descending paths forming these parts will be situated in the core zone of a piece, each of the paths at distance at least 3 from the others, in order to allow correct pairing of beginnings and ends.

We now describe the structure of a counter. This is a collection of $q \in \Theta(\frac{\log n}{\log \log n})$ pairwise disjoint descending paths, of sizes $L + 1, L + 2, \ldots L + q$, where $L \in \Theta(\log n)$. We take paths of different lengths in order to easily distinguish them. Nodes of the ith path are numbered 1 to $L + i$ (where 1 corresponds to the beginning). Two towers will be placed in the first and third nodes of each path, thus marking its beginning. Similarly, three towers will be placed at the end of each path, separated by empty nodes, thus marking its end. Moreover there will be two or three robots moving from node 7 to node $L - 8$ of each path. If these robots are located in the same node (thus forming a tower), their

position codes a numerical value. By combining these values on all paths, we obtain the value of the counter. Since on each path there are $L - 14$ available positions, the value of the counter is the resulting integer written in base $L - 14$.

Let $q = \lceil 2 \log n / \log \log n \rceil$ and $L_1 = \frac{1}{10} \log n$. Take q of the descending paths described in the proof of Lemma 2 (chosen in an arbitrary deterministic way, identical for all robots, and excluding p_1), and in the ith path, where $1 \leq i \leq q$, take the lower part of size $L_1 + i$. These will be the descending paths of the first counter. Similarly, let $L_2 = L_1 + q + 1$. Take a set of q descending paths, other than those used for the first counter and other than p_1. In the ith path, where $1 \leq i \leq q$, take the lower part of size $L_2 + i$. These will be the descending paths of the second counter.

Another module of the brain is the semaphore consisting of two of the descending paths constructed in the proof of Lemma 2 (again excluding p_1). In each of these paths take the lower part of distinct constant sizes. The beginning and end of each path is marked similarly as in the counter. Likewise, there are two or three robots moving in each of these paths, their possible locations restricted to node 7 and 8 in each path. In each path, if these robots are located in the same node (thus forming a tower), they code one bit. Thus the semaphore has 4 possible values $00, 01, 11, 10$.

Finally, the garbage is the first descending path p_1 constructed in the proof of Lemma 2. This path has the property that its beginning is at the root of the piece. This path has length at least $\frac{1}{8} \log n$, and thus larger than the total number of robots, for sufficiently large n. The role of the garbage is to store all robots of the brain not used for the counters and the semaphore. The garbage is filled by putting a tower or a robot every 5 nodes in the path, until all robots are disposed. Therefore the end of the path is marked similarly as for paths in the counter and the semaphore, but the beginning is left unmarked.

Ordering of Robots

We first define a total order $\sqsubset$ on the set of all views. Let $V = View(T, v)$ and $V' = View(T', v')$. If the height of T is smaller than the height of T' then $V \sqsubset V'$. Otherwise, if the height of both trees is 0 then $(x, \emptyset) \sqsubset (x', \emptyset)$, if $x \leq x'$, where $0 < 1 < *$. If the height of both trees is positive, the order of views is the lexicographic order on the sequences $(x, View(T_1, v_1), \ldots, View(T_m, v_m))$, where views at children are ordered increasingly by induction.

We now define the following total preorder $\leq$ on the robots in the rooted tree T. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two robots located at nodes v_1, v_2, at distances d_1 and d_2 from the root. (In the case of the bicentroid, we consider the distance to its closer extremity.) We let $\mathcal{R}_1 \leq \mathcal{R}_2$, if and only if, $d_1 < d_2$, or $d_1 = d_2 \wedge View(T, v_1) \sqsubset View(T, v_2)$. Note that the equivalence relation induced by this preorder is exactly the equivalence between robots defined previously. We say that a robot is larger (smaller) than another one meaning the preorder $\leq$. A robot not equivalent to any other is called *solitaire*.

Lemma 3. *The number of equivalent robots in any piece is either 1 or even.*

It follows from Lemma 3 that any unique piece with an odd weight must contain a solitaire.

Description of Algorithm Tree-exploration

Phase 1. There is no tower in the snapshot.
Goal: Empty the core zones of all pieces and create one tower in one piece.

We first free the core zones by moving every robot to an empty child, as long as such a child exists, except for up to two robots that may move from one piece to another. As described below, these exceptional robots are solitaires. The objective here is to have a unique heaviest piece with the additional property that it is either of odd weight or completely occupied by robots (i.e. every node of the piece is occupied by a robot). This is always possible because $k \equiv 5 \pmod 6$. Indeed, if there are two heaviest pieces, then there must exist a third piece of odd weight, and thus a solitaire of this piece (whose existence is guaranteed by Lemma 3) can move to one of the heaviest pieces, thus breaking the tie. If there is a unique heaviest piece, but of even weight and not completely occupied by robots, then there must exist another piece of odd weight, and thus a solitaire of this piece (whose existence, again, is guaranteed by Lemma 3) can move to the heaviest piece. Note that the case of three heaviest pieces is impossible because k is not divisible by 3.

As soon as the required properties hold in a piece P and the core zones are empty (except for possibly one robot in the core zone of P), a tower is created outside the core zone of P by moving a solitaire to an occupied node in such a way that at least half the robots in P, including a solitaire, are located outside the subtree rooted at the tower. The latter precaution is taken to have enough robots to form and subsequently move towers in Phase 2 using the solitaire. The way this is done will be described in the sequel.

Phase 1 has been clearly identified by the absence of towers in the snapshot. Such an easy characterization is not available in the subsequent phases, hence we use a gadget called *signal* to identify them. A signal is a largest set of at least 4 towers situated on a descending path inside a piece, such that consecutive towers are separated by two empty nodes. The value of a signal is $x - 1$, where x is the number of towers in it. This value will indicate the number of the current phase.

Phase 2. There is at least one tower and no signal in the snapshot.

In this phase piece P can be recognized as the unique piece where there is a tower outside the core zone and Q as the largest among pieces other than P (in the case of a tie Q is any piece with robots in the core zone.) Notice that, at the beginning of Phase 2, the core zone of Q does not contain any robots. Hence there is room in it for the brain.
Goal: Construct and initialize the brain in the core zone of piece Q, prepare the other pieces for exploration, and create the signal.

Stage 1. Goal: Move robots from P in order to construct the brain in Q and prepare P for exploration.

We now describe the way to form towers in P and move them to appropriate places in the descending paths forming the brain in Q. Robots migrate from piece P to piece Q, one or two robots at a time. The next robot or pair of robots starts its trip from P to Q only after the previous one is at its place. The aim

is to occupy target nodes by towers. Nodes in descending paths are filled one path after another in a DFS post-order of beginnings of the paths. Thus a tower occupies a node v only after all robots in the subtree rooted at v are in their target positions. This rule applies to all descending paths of the brain, except the garbage. The latter is constructed at the end, after all other parts of the brain are completed. This is possible because the descending path containing the garbage starts at the root of the piece (path p_1 described in the proof of Lemma 2). The above migration of towers is done until there remains only a single tower and a solitaire in P. This prepares P for exploration.

There are two difficulties in performing this migration, both due to symmetries in configurations of robots. The first difficulty is to form towers consisting of only two robots in P and the other is to place such a tower in a specific target node in Q. (We want to restrict the size of towers in order to be able to create many of them using the available robots).

The essence of the first difficulty is that equivalence classes of robots can be large and thus it may be difficult to form a single small tower. (For example, if all robots in a piece are equivalent and occupy the same level, a single small tower cannot be formed without outside help.) We solve this problem by using a solitaire to break symmetry between two equivalent robots. More precisely, the solitaire moves to meet one of the equivalent robots thus creating a tower of two robots. At the same time the other equivalent robot becomes a solitaire.

The essence of the second difficulty is that if there are at least two equivalent target positions that a tower could occupy, the adversary could break the tower at the time when the tower tries to go down from the least common ancestor of these target nodes, sending each of the robots forming the tower to a different target node. We solve this problem by using a solitaire to first break the symmetry between these target positions. This solitaire, called the *guide* of the tower, is placed in one of these positions, thus indicating that the tower should go to the closest of the equivalent positions. As soon as the tower reaches its target, the solitaire is again available to break other symmetries, either those encountered when forming towers in P or when placing them in Q.

Stage 1 ends with the brain constructed in the core zone of piece Q. Moreover, in piece P there remain only a single tower and a robot without towers in its ancestors.

Stage 2. Goal: Empty the third piece P' (other than P and Q), if it exists.

This is done as follows. A largest robot of P', not in the root of P' (either a free robot or in a tower) goes to its parent. When there are no robots outside the root, the robots from the root of P' go to the garbage in Q. This way of merging all robots of P' at the root of this piece prevents accidental creation of a signal. Stage 2 ends when the ending condition of Stage 1 holds and piece P', if it exists, is empty.

Stage 3. Goal: Create the signal.

The signal is created at the bottom of the garbage (without considering towers marking its end). Towers descend in the garbage one at a time, until two sequences consisting of 4 towers, each at distance 3 from the preceding one, are

created. These two sequences are separated by 5 empty nodes. Since there is no longer sequence of this type in the entire tree, the value of the newly created signal is 3. This completes Stage 3 and the entire Phase 2. (Note that we use two sequences forming a signal, rather than just one, in order to be able to move one of these sequences later on, without destroying the value of the signal. In fact we also need to leave two additional towers between these sequences, in order to update the value of signal from 3 to 4, when passing to Phase 4.)

From now on all towers in the entire tree are separated by at least one empty node. Hence if a tower moves and the adversary breaks it by holding back some of the robots of the tower, this can be recognized in subsequent snapshots and the moving tower can be reconstructed. Note that from now on we need not specify the existence of a tower in the snapshot, since the signal contains towers.

Phase 3. The value of signal is 3.
Goal: Explore P'': the largest of the pieces other than Q. (We explore this piece first to be able to relocate the brain into it in Phase 4: the other piece could be too small.)

At the beginning of Phase 3 both counters indicate value 0. Piece P'' will be explored by the free robot and the tower that are currently outside Q. They will be called exploring robot and exploring tower, respectively. These two entities explore leaves of P'' one similarity class after another in increasing order, induced by any total preorder of the nodes, with the following property: the equivalence classes induced by this preorder are the previously defined similarity classes. The entities move only if both counters indicate the same value i. Suppose that the jth class has size s_j. Let r be such that $i = s_1 + \cdots + s_d + r$, with $r \leq s_{d+1}$. Hence the brain indicates that the next leaf to be explored is the rth leaf in the $(d+1)$st class. If $r = 1$, the exploring robot goes to any leaf of the $(d+1)$st class. Otherwise, consider two cases. If r is even then let u be the leaf where the exploring robot is located. In this case the exploring tower goes to the (unique) closest leaf in the same similarity class. If r is odd then let v be the leaf where the exploring tower is located. In this case the exploring robot goes to the leaf w determined as follows. Let j be the length of the longest sequence of 1's counted from the right (least significant bit) of the binary expansion of the integer $(r-3)/2$. Order all leaves of the similarity class of v in any non-decreasing order of distances from v. The leaf w is the 2^{j+1}th node in this order. Notice that w is the closest leaf from v not yet explored.

Incrementing values of both counters from i to $i+1$ and moving the exploring robots according to those increments are complex actions involving relocation of many robots. Due to asynchrony, snapshots can be taken during these complex actions, potentially disorganizing the process. To ensure correct exploration, we artificially synchronize these actions using the semaphore. Its values change in the cycle $00, 01, 11, 10, 00, \ldots$. Note that the changes of values of the semaphore do not need additional synchronization, as each change involves a move of only one robot or tower. In the case of a move of a tower, the adversary can split the tower by delaying some of its robots and moving others, hence the value of the

corresponding bit is unclear and robots must decide which value should be set. Nevertheless this is never ambiguous: for example, if the value of the first bit is 0 and the second is unclear, it must be set to 1 because, when the first bit is 0, the only possible change of the second bit is from 0 to 1. Other cases are similar.

At the beginning of Phase 3 the semaphore is at 00. This indicates that the first counter has to modify its value to $i + 1$, where i is the current value of the second counter. When this is done, the value of the semaphore changes to 01. This indicates that the second counter has to modify its value to the current value of the first counter. When this is done, the value of the semaphore changes to 11. This indicates that the exploring robot or the exploring tower (depending on the parity of the value shown by the counters) has to move to the neighbor of the leaf it occupies. When this is done, the value of the semaphore changes to 10. This indicates that the exploring entity which is in an internal node (i.e., the one that has just moved) has to move to the leaf indicated by the value of both counters, as explained above. When this is done, the value of the semaphore changes to 00.

Phase 3 is completed when the semaphore has value 11 and both counters have value $f + 1$, where f is the number of leaves in piece P''. At this time the value of signal is changed from 3 to 4 (by moving an additional tower down the garbage), thus marking the end of this phase. Note that, when both counters have value $f + 1$, all leaves of P'' are explored. There are two cases. If $P'' = P$ then at least one path between the root and a leaf of P'' has been explored when P was evacuated. Otherwise, at least one path between the root and a leaf of P'' has been explored when the exploring solitaire came from P to explore P''. Hence in both cases all leaves and at least one path between the root and a leaf have been explored. Since by the description of the exploration the explored part of P'' is connected, this implies that the entire piece has been explored.

Phase 4. The value of signal is 4.
Goal: Relocate the brain from Q to P'' (except when there are only two pieces and Q has few leaves, in which case exploration of Q is done immediately: see Subcase 2.2).

While the brain is relocated to P'', piece Q is emptied and thus ready to be explored. Piece Q is emptied in reverse order of its filling in Phase 2, i.e., robots that came last to Q leave it first. We will need the exploring solitaire and tower in piece Q in order to perform exploration during Phase 5. Hence while towers forming the old brain move from Q to P'', the solitaire and the exploring tower move in the opposite direction. This creates a problem when the tree has a long path of nodes of degree two, between the old brain and piece P'': there is no room to cross on this path. Hence for this class of trees we will use a particular technique. Consider two cases.

Case 1. There exist nodes v and w outside P'' such that the path from the root of P'' to each of them does not contain robots and there exists a path from a tower in Q to the root of P'' not containing robots and not containing v or w.

In this case there is no crossing problem. The solitaire and the exploring tower from Q can hide in v and w to let the towers from Q (that formed the old brain) move to P''.

Case 2. There are no nodes v and w as described in Case 1.

Let M be the largest integer such that $10c \log M / \log \log M \geq \log M$.

– Subcase 2.1. The number of leaves in piece Q is larger than M.

Since any tree of maximum degree 3 containing f leaves has height at least $\log f$, the condition on integer M implies that there exists a descending path in Q, with beginning u, satisfying the following properties:

(1) it is able to store all towers needed to explore Q, leaving distance 4 between consecutive towers. (We leave distance 4 not to confuse the sequence of towers with a signal.)

(2) there exist two leaves outside the tree rooted at u.

All towers from Q are moved to the above descending path leaving 3 empty nodes between consecutive towers, with the following exception. When moving the first five towers, the value of the signal is recreated using these towers. This is done before moving the second sequence of the signal created in Phase 2. After moving $2c \log M / \log \log M$ towers, all additional towers from this path are collapsed to one tower. After this compacting the condition of Case 1 holds because of property (2).

– Subcase 2.2. The number of leaves in piece Q is at most M.

In this case there are so few leaves that we can explore all of them without using a brain. First we recreate the signal in P with value 4, to record the phase number. Then all robots from Q go to the leaves. When all leaves are occupied, all robots go towards the root of Q forming a tower in this root, thus exploring the remaining nodes of Q. At this point the algorithm stops (explicit stopping condition).

Thus, after a finite number of moves in Case 2, either the exploration is completed (Subcase 2.2) or the algorithm transits to Case 1. From now on we suppose that the condition of Case 1 holds.

We continue Phase 4 by creating a signal with value 4 in piece P''. This is done by moving towers from the top of the garbage in Q and placing them outside the core zone in P''. The path forming the signal is of bounded length and thus there is enough space outside the core zone to place it. Moreover we place three additional towers in this path to be able to subsequently increase the value of the signal up to 7. After this is done we create the new brain in P'', similarly as in Phase 2. In particular, we use the solitaire as a guide to direct the towers coming from Q to their target positions. Note that all towers and robots in the core zone of Q are alone in their equivalence classes and thus there is no need to break symmetries using solitaires. When the counters and the semaphore of the new brain are created in P'', all robots from Q, except the exploring tower and solitaire are moved to the garbage of the new brain. Note that all the above actions are possible, since the solitaire and towers are able to move between pieces Q and P'' without crossing problems.

When there is only the exploring tower and solitaire in Q, the value of signal in P'' is incremented to 5. This ends Phase 4.

Phase 5. The value of signal is 5.
Goal: Explore piece Q and stop if there are only two pieces.

We proceed exactly as in Phase 3, this time exploring piece Q instead of P''. When the brain indicates that all leaves are explored, two situations are possible. If there are only two pieces in the tree, all nodes are already explored and the algorithm stops (explicit stopping condition). If there are three pieces, the value of signal is incremented to 6. This ends Phase 5.

Phase 6. The value of signal is 6.
Goal: Reinitialize the brain and relocate the exploring solitaire to the unexplored piece.

Both counters in the brain are reset to 0, the semaphore is reset to 00. The exploring solitaire moves to the root of the unexplored piece. The value of signal is incremented to 7. This ends Phase 6.

Phase 7. The value of signal is 7.
Goal: Explore the last piece and stop.

The piece containing only a solitaire is explored (using this solitaire and the tower from Q). This is done again as in Phase 3. When the brain indicates that all leaves are explored, exploration is completed and the algorithm stops (explicit stopping condition).

It remains to give the conditions on the constant c such that the number k of robots satisfies $c \log n / \log \log n \leq k \leq 2c \log n / \log \log n$. The constant c should be chosen so that there are sufficiently many robots to form the brain (including the markers of descending paths' extremities) and the exploring team. Note that if there are three pieces in the tree, and robots are initially equally divided among them, only $k/3$ robots will be used.

3.3 Exploration of Trees of Maximum Degree 3: Lower Bound

We now prove a lower bound on the number of robots necessary for exploration of complete binary trees, that matches the upper bound given by Algorithm Tree-exploration.

Theorem 2. $\Omega(\frac{\log n}{\log \log n})$ *robots are required to explore n-node complete binary trees.*

4 Concluding Remarks and Open Problems

A natural next research step would be the investigation of the exploration problem when the visibility of the robots is limited, e.g., to the immediate neighborhood. Notice that, in this case, exploration is not generally possible. Hence a limited visibility scenario could only work for some subset of initial configurations. Another line of research would be to equip robots with very small (e.g.,

constant) memory of past events and study how this additional power influences feasibility of exploration with limited or unlimited visibility. Finally, it would be interesting to extend our study to the case of arbitrary graphs, as well as to the stronger SSYNCH model.

References

1. Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM J. on Comput 29, 1164–1188 (2000)
2. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Trans. on Robotics and Automation 15, 818–828 (1999)
3. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the robots gathering problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1181–1196. Springer, Heidelberg (2003)
4. Cohen, R., Peleg, D.: Local algorithms for autonomous robot systems. In: Proc. 13th International Colloquium on Structural Information and Communication Complexity, (SIROCCO 2006). LNCS, vol. 3221, pp. 29–43. Springer, Heidelberg (2006)
5. Czyzowicz, J., Gasieniec, L., Pelc, A.: Gathering few fat mobile robots in the plane. In: Proc. 10th International Conference on Principles of Distributed Systems (OPODIS 2006). LNCS, vol. 4288, pp. 744–753. Springer, Heidelberg (2006)
6. Dessmark, A., Pelc, A.: Optimal graph exploration without good maps. Theoretical Computer Science 326, 343–362 (2004)
7. Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 11–22. Springer, Heidelberg (2005)
8. Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Computing without communicating: Ring exploration by asynchronous oblivious robots. In: Tovar, E., Tsigas, P., Fouchal, H. (eds.) OPODIS 2007. LNCS, vol. 4878, pp. 105–118. Springer, Heidelberg (2007)
9. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theoretical Computer Science 337, 147–168 (2005)
10. Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 585–594 (2007)
11. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoretical Computer Science 390, 27–39 (2008)
12. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. Journal of Algorithms 33, 281–295 (1999)
13. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theoretical Computer Science 384, 222–231 (2007)
14. Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 333–349. Springer, Heidelberg (2006)
15. Sugihara, K., Suzuki, I.: Distributed algorithms for formation of geometric patterns with many mobile robots. Journal of Robotic Systems 13(3), 127–139 (1996)
16. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: formation of geometric patterns. SIAM J. Comput. 28, 1347–1363 (1999)