Tree Exploration with Little Memory

Krzysztof Diks* Pierre Fraigniaud† Evangelos Kranakis‡ Andrzej Pelc§

Abstract

A robot with k-bit memory has to explore a tree whose nodes are unlabeled and edge ports are locally labeled at each node. The robot has no *a priori* knowledge of the topology of the tree or of its size, and its aim is to traverse all the edges. While $O(\log \Delta)$ bits of memory suffice to explore any tree of maximum degree Δ if stopping is not required, we show that bounded memory is not sufficient to explore with stop all trees of bounded degree (indeed $\Omega(\log \log \log n)$ bits of memory are needed for some such trees of size n). For the more demanding task requiring to stop at the starting node after completing exploration, we show a sharper lower bound $\Omega(\log n)$ on required memory size, and present an algorithm to accomplish this task with $O(\log^2 n)$-bit memory, for all n-node trees.

1 Introduction

A robot (mobile agent) has to *explore* an undirected graph by visiting all its nodes and traversing all edges, without any *a priori* knowledge of the topology of the graph nor of its size. The task of visiting all nodes of a network is fundamental in searching for data stored at unknown nodes of a network, and traversing all edges is often required in network maintenance and when looking for defective components. If nodes and edges have unique labels, this can be easily done by depth-first search. However, in some navigation problems in unknown environments such unique labeling may not be available, or limited sensory capabilities of the robot may prevent it from perceiving such labels. Hence it is important to be able to program the robot to explore *anonymous* graphs, i.e., graphs without unique labeling of nodes or edges. Unfortunately, arbitrary graphs cannot be explored under such weak assumptions, as witnessed by the case of a cycle: without any labels of nodes and without the possibility of putting marks on them, it is clearly impossible to explore a cycle of unknown size and stop. If marking of nodes (e.g., by dropping and removing *pebbles*) is available then the problem can be solved even in directed graphs (cf. [4]). Otherwise, the class of graphs that can potentially be explored has to be restricted to connected graphs without cycles, i.e., to trees.

In this paper we study the problem of graph exploration under very weak assumptions: we do not assume any labels on nodes and do not allow marking, hence we restrict attention to exploration of trees. Clearly the robot has to be able to *locally* distinguish ports at a node: otherwise it is impossible to explore even the star with 3 leaves (after visiting the second leaf the robot cannot distinguish the port leading to the first visited leaf from that leading to the unvisited one). Hence we make a natural assumption that all ports at a node are locally labeled 1,...,d, where d is the degree of the node. No coherence between those local labelings is assumed.

In many applications, robots (mobile agents) are meant to be simple, often small, and inexpensive devices which limits the amount of memory with which they can be equipped. As opposed to numerous papers that imposed no restrictions on the memory of the robot and sought exploration algorithms minimizing time, i.e., the number of traversals, we investigate the minimum size of the memory of the robot that allows exploration of trees of given (unknown) size, regardless of the time of exploration. More precisely, we consider the following main tasks:

- exploration with stop: starting at any node of the tree, the robot has to traverse all edges and stop at some node;
- exploration with return: starting at any node of the tree, the robot has to traverse all edges and stop at the starting node.

In both cases we want to find an algorithm for a robot performing the given task using as little memory as possible.

1.1 Our results.

We first consider the auxiliary easier task of *perpetual exploration* in which the robot has to traverse all edges of the tree but is not required to stop.

*Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland. E-mail: *diks@mimuw.edu.pl*.

†CNRS, Laboratoire de Recherche en Informatique, Université Paris-Sud, 91405 Orsay, France. *http://www.lri.fr/~pierre*.

‡School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. *kranakis@scs.carleton.ca*.

§Département d'Informatique, Université du Québec à Hull, Hull, Québec J8X 3X7, Canada. *pelc@uqah.uquebec.ca*.

Perpetual exploration may be of independent interest, e.g., if regular control of a network for the presence of faults is required, and all edges must be periodically traversed over long periods of time. In our context, the study of perpetual exploration shows that from the point of view of memory use, the most demanding part of exploration with stop is indeed stopping. Here is why. We show an algorithm performing perpetual exploration of any tree of maximum degree Δ using $O(\log \Delta)$-bit memory. In particular, bounded memory is sufficient for perpetual exploration of all bounded degree trees. In contrast we show that this is not the case for exploration with stop. Indeed, $\Omega(\log \log \log n)$ bits of memory are needed for exploration with stop of some bounded degree trees of size n. The additional memory is essentially needed to decide when to stop in our anonymous environment. Moreover, our memory-efficient perpetual exploration algorithm yields algorithms for exploration with stop in special classes of trees, or for all trees under additional knowledge of a bound on size. Indeed, if we know a bound N on the size of the tree, we can explore it with stop using $O(\log N)$-bit memory, while if we know a bound m on the number of nodes of a given degree then we can explore a tree with stop using $O(\log \Delta + \log m)$-bit memory.

For exploration with return, we show a lower bound $\Omega(\log n)$ on the number of memory bits needed for trees of size n. As for upper bounds, a simple DFS-based algorithm performs exploration with return using $O(D \log \Delta)$ memory bits for trees of diameter D and maximum degree Δ. However, the memory used by this algorithm can be linear in the size n for some trees. Our main algorithm for exploration with return is much more memory-efficient in general: it uses only $O(\log^2 n)$ memory bits for all trees of size n.

1.2 Related work.

Exploration and navigation problems for robots in an unknown environment have been extensively studied in the literature (cf. the survey [14]). There are two groups of models for these problems. In one of them a particular geometric setting is assumed, e.g., unknown terrain with convex obstacles [7], or room with polygonal [9] or rectangular [3] obstacles. Another approach is to model the environment as a graph, assuming that the robot may only move along its edges. The graph setting can be further specified in two different ways. In [1, 4, 5, 10] the robot explores strongly connected directed graphs and it can move only in the direction from head to tail of an edge, not vice-versa. In [2, 8, 11, 12] the explored graph is undirected and the robot can traverse edges in both directions. In the graph setting it is often required that apart from completing exploration the robot has to draw a map of the graph, i.e., output an isomorphic copy of it.

The efficiency measure adopted in most papers dealing with exploration of graphs is the time of completing this task, measured by the number of edge traversals by the robot. On the other hand, there are no restrictions imposed on the memory of the robot. (This is precisely, where our setting differs: we are interested in minimizing the memory of the robot but do not restrict exploration time.)

Graph exploration scenarios considered in the literature differ in an important way: it is either assumed that nodes of the graph have unique labels which the robot can recognize, or it is assumed that nodes are anonymous. Exploration of directed graphs assuming the existence of labels was investigated in [1, 10]. In this case no restrictions on the robot moves were imposed, other than by directions of edges, and fast exploration and mapping algorithms were sought. Exploration of undirected labeled graphs was considered in [2, 8, 11, 12]. Since in this case a simple exploration based on depth-first search can be completed in time $2e$, where e is the number of edges, investigations concentrated either on further reducing time for an unrestricted robot, or on studying efficient exploration when moves of the robot are restricted in some way. The first approach was adopted in [12], where an exploration algorithm working in time $e + O(n)$, with n being the number of nodes, was proposed. Restricted robots were investigated in [2, 8, 11]. It was assumed that the robot has either a restricted tank [2, 8], forcing it to periodically return to the base for refueling, or that it is tethered, i.e., attached to the base by a rope or cable of restricted length [11]. It was proved in [11] that exploration and mapping can be done in time $O(e)$ under both scenarios.

Exploration of anonymous graphs presents a different type of challenges. In this case it is impossible to explore arbitrary graphs if no marking of nodes is allowed. Hence the scenario adopted in [4, 5] was to allow *pebbles* which the robot can drop on nodes to recognize already visited ones, and then remove them and drop in other places. The authors concentrated attention on the minimum number of pebbles allowing efficient exploration and mapping of arbitrary directed n-node graphs. (In the case of undirected graphs, one pebble suffices for efficient exploration.) In [5] the authors compared exploration power of one robot to that of two cooperating robots with a constant number of pebbles. In [4] it was shown that one pebble is enough if the robot knows an upper bound on the size of the graph, and $\Theta(\log \log n)$ pebbles are necessary and sufficient otherwise.

Our scenario is even weaker than that in [4, 5]: nodes do not have labels and no marking is allowed. Since the presence of even one cycle precludes exploration with stop, we restrict attention to the class of (undirected) graphs in which this task is possible under these very weak assumptions: the class of trees. Moreover, as previously mentioned, our scenario differs in that we optimize memory, not exploration time.

2 Terminology and preliminaries

A tree with locally labeled ports is an undirected tree whose nodes are unlabeled and edges incident to a node v have distinct labels $1,\dots,d$, where d is the degree of v. Thus every edge uv has two labels which are called its port numbers at u and at v. Port numbering is local: there is no relation between labels given to an edge uv at u and at v.

A robot with k-bit memory is a deterministic state machine with $K = 2^k$ states among which a specified state S_0 is called *initial* and some specified states are called *final*. Such a robot, when placed in the initial state in any node of a tree with locally labeled ports, operates as follows. If the robot is in a node in a non-final state S, the state determines a local port number i_S. The robot leaves the node by this port. Upon traversing the corresponding edge, the robot reads the port number at the node it enters and the degree of this node. This pair of integers is an input symbol that causes the transition from state S to S'. The robot continues moving in this way until it enters a final state for the first time. Then it stops.

As said in Section 1, we consider three tasks of increasing difficulty: *perpetual exploration* in which the robot has to traverse all edges of the tree but is not required to stop, *exploration with stop* in which starting at any node of the tree, the robot has to traverse all edges and stop at some node, and *exploration with return* in which starting at any node of the tree, the robot has to traverse all edges and stop at the starting node. A robot is said to perform one of the above tasks in a tree, if starting at *any* node of this tree in state S_0 it completes this task in finitely many steps. (Notice that in the case of perpetual exploration, completing this task after finitely many steps means only traversing all edges, not necessarily stopping after it.)

The way in which operation of the robot in a tree is defined implies that a robot exploring a star with Δ leaves needs at least Δ states: if it has fewer states then, starting at the center of the star it could never enter some ports. Hence we get the following trivial lower bound.

Proposition 2.1. *A robot exploring all trees of maximum degree Δ must have $\Omega(\log \Delta)$-bit memory.*

The following simple algorithm performs perpetual exploration using only $O(\log \Delta)$ memory bits:

Algorithm `CHOOSE-NEXT`. The robot leaves the starting node by port 1. After entering any node of degree d by port i, the robot leaves it by port $(i \bmod d)+1$.

It is easy to see that, using algorithm `CHOOSE-NEXT` the robot traverses all edges of an n-node tree after at most $2(n-1)$ steps. Hence we have

Proposition 2.2. *Algorithm* `CHOOSE-NEXT` *accomplishes perpetual exploration of any tree of maximum degree Δ using $O(\log \Delta)$ bits of memory.*

This observation implies algorithms for exploration with stop if some additional information about the tree is available. One such situation arises when an upper bound N on the number of nodes in the tree is known. Then the robot can perform algorithm `CHOOSE-NEXT`, additionally counting steps, and stop after $2(N-1)$ steps. This requires $\Omega(\log N)$-bit memory. Another type of additional information could be an upper bound m on the number of nodes of given degree d. Using algorithm `CHOOSE-NEXT` every such node is visited at most $2d$ times before all edges are traversed. Hence it is enough to perform this algorithm, additionally counting visits at nodes of degree d, and stop after $2md$ visits. This can be done with $O(\log \Delta + \log m)$-bit memory. For example, this enables exploration with stop of any rooted binary tree (whose root is the only node of degree 2) using bounded memory.

3 Exploration with stop

As we have seen, perpetual exploration can be performed in all bounded degree trees using bounded memory, i.e., the same robot can perform perpetual exploration in all trees of maximum degree bounded by a constant. This ability can be transfered to the more demanding task of exploration with stop for restricted classes of bounded degree trees, such as trees with bounded number of nodes of given degree. However, the main result of this section shows that exploration with stop of all trees of bounded degree, even of all trees of maximum degree 3, cannot be performed by the same robot. (Of course, all trees of maximum degree 2, i.e., paths, can be explored with stop by the same robot with only 3 memory bits).

Theorem 3.1. *For every robot there exists a tree of maximum degree 3 which this robot cannot explore with stop.*

Proof. Fix a robot with the set $\mathcal{S}$ of states, such that $|\mathcal{S}| = K$. The idea of the proof is to construct two trees such that if the robot explores one of them and stops then, when run on the other, it stops before exploring it. In fact, we will construct parts of the second (larger) tree by simulating actions of the robot in the first tree.

We restrict attention to the class of trees all of whose internal nodes have degree 3. Moreover we consider only *symmetric* port labelings, i.e., such that both labels on an edge joining internal nodes are equal. Hence we may speak of colors 1,2,3, of such edges, where the color number of an edge is equal to both port numbers at this edge. The color of an edge joining a leaf with an internal node is the port number at the internal node (the port number at a leaf is always 1). Notice that incident edges have different colors. Call such colored trees of degree 3 *proper trees.*

For proper trees the behavior of the robot is particularly simple. If the robot is in a non-final state and is situated at a leaf, it takes the unique port, reads the port number at the neighbor (equal to the edge color) and transits to some state. If the robot is in a non-final state and is situated at an internal node, the state determines the color of the edge to take, the robot moves, it only verifies if the neighbor is a leaf or an internal node (in both cases the entry port number is already known) and it transits to a new state. In particular, on a path consisting only of internal nodes, the state of the robot at the end of the path depends only on the state at the beginning of it.

Consider a proper infinite tree with exactly one leaf. Let the robot start at the leaf in the initial state. Without loss of generality we may assume that the robot visits nodes of this tree arbitrarily distant from the leaf (otherwise it is easy to construct a finite tree which the robot would not explore). For any positive integer x, define the following function $f_x : \mathcal{S} \times \{1,2,3\} \longrightarrow \mathcal{S} \times \{1,2,3\}$. Fix $S \in \mathcal{S}$ and $i \in \{1,2,3\}$. Suppose that the robot starts at the leaf in state S and that the color of the edge incident to the leaf is i. Let v be the first node at distance x from the leaf, visited by the robot. Cut the two edges incident to v different from the edge by which the robot entered v, thus making v a leaf. Let $f_x(S,i) = (S',j)$, where S' is defined as the state of the robot after entering the leaf v and j is defined as the color of the edge by which the robot enters v.

Let $g = 3 \cdot 3K^2(3K)^{3K}$. Consider numbers $K+1, \dots, K+d$. Since there are only $(3K)^{3K}$ possible functions f_x, there exist two even integers a and b with the property $K < a < a + 3K^2 < b < K + g$, such that $f_a = f_b$. Fix two such integers. They have the following property. Suppose that the robot starts at a leaf of a proper tree in any state and traverses a (non-necessarily simple) path in the tree such that the end of this path is a leaf and all internal nodes of the path are internal nodes of the tree. Then the state of the robot at the other end of the path and the color of the entry edge to it are the same, regardless of whether the distance between the beginning and the end of the path is a or b.

We now construct the two proper trees with the property mentioned in the beginning of the proof. The tree T is defined as follows. Take a node v of degree 3 and attach to each of its 3 neighbors a complete binary tree of height $a/2 - 1$. Each of these complete binary trees is called a *principal subtree* of T and v is called the center of T. The tree T is proper and has diameter a. The coloring of its edges is unique up to automorphism.

Suppose that the robot explores T and eventually stops, when starting at a leaf w. We define a *long trip* in T to be a part of the trajectory of the robot in T which starts and ends at leaves at distance a and traverses only internal nodes on the way. The robot must make at least 2 and at most K long trips before stopping. The first is clear and the second follows from the fact that if the robot is twice in the same non-final state in the same node then it never stops. More than K long trips would cause the robot to be in the same non-final state in the central node v at least twice.

We now construct a proper tree T' in which the robot stops before exploring all of it. The construction proceeds incrementally by adding consecutive pieces to the tree T' under construction. First construct an isomorphic copy T_1' of the principal subtree T_1 of T containing the leaf w_1 beginning the first long trip, with the same coloring as in the original. Let w_1' be the leaf corresponding to w_1. Add to T_1' a copy of the part of the trajectory of the robot before the first long trip in T. This trajectory may have nodes outside of T_1', corresponding to nodes in principal subtrees of T, other than T_1. However, the only leaves on this trajectory are those of T_1. Since there are no leaves at distance a on it, call this trajectory a *short trip.* Let S_1 be the state of the robot at w_1 at the beginning of the first long trip in T. Consider the infinite proper tree containing T_1' whose set of leaves is exactly the set of leaves of T_1'. Consider the run r' of the robot in this infinite tree starting at w_1' in state S_1, until hitting a node u_1' at distance b from w_1'.

Claim. During the run r' the robot does not visit any leaf of T_1'.

Let r be the long trip in T starting at w_1 in state S_1. This long trip corresponds to an initial segment of r'. By definition, the robot does not visit any leaf of T_1 during r. Let u_1 be the node at distance a from w_1 at the end of r and denote by p_1 the simple path between w_1 and u_1. Consider the state of the robot at the first visit of each node of this path. Since $a > K$, for some internal nodes x and y on the path p_1 this state is the same. Let x be the node first visited before y and let α be the distance between x and y. Since after first visiting x the robot made progress α on the path p_1 without visiting a leaf, after first visiting y it will make further progress α on this path without visiting a leaf because the state at the first visit of x and y was the same. Using the same reasoning for the run r' of the robot in the infinite tree, the robot will keep going farther from w'_1 first on the path corresponding to p and then on its extension in the infinite tree, without ever returning to leaves of T'_1, until the end of run r'. This proves the claim.

We continue the construction of the tree T' by adding the run r' (with appropriate coloring of edges) to the previously constructed part. Let T_2 be the principal subtree of T containing the leaf u_1 at the end of the first long trip in T. At the end of run r' add an isomorphic copy T'_2 of T_2 in such a way that the leaf u'_1 corresponding to u_1 is the node ending run r', and we reproduce the coloring from T_2. Notice that, in view of the properties of integers a and b, the state of the robot after hitting the leaf u'_1 in T' is the same as the state at the end of the first long trip in T (provided that all nodes of run r' have degree 3 in T', which we will guarantee at the end of the construction). Intuitively speaking, the robot does not know at this point if it is at u'_1 in T', or at u_1 in T.

Let p'_1 be the simple path of length b joining w'_1 with u'_1. Consider the state of the robot at the first visit of each internal node of this path. Since there are only K states, it follows that for two nodes v_1 and v_2 at distance at most K on path p'_1 this state is the same. Let v_1 precede v_2 on the path and let I be the segment of path p'_1 between these nodes. Consider the sequence of states at first visits of nodes of segment I. Consider the segment J of path p'_1 following v_2 and of the same length as I. The sequence of states at first visits of corresponding nodes of I and J and the sequence of corresponding edge colors are the same. Hence the sequence of edge colors on path p'_1 is periodic with period of length $s \leq K$. By definition of b there are at least $3K$ such periods on p'_1 between the root of T'_1 and the root of T'_2.

Let now w_2 be the leaf in T_2 from which the second long trip in T is started and let w'_2 be the corresponding leaf in T'_2. Consider the part of the trajectory of the robot in T between the end of the first long trip and the beginning of the second. Call it a short trip, as before: it has the same properties as those observed for the first short trip. Add an isomorphic copy of this short trip to T', between leaves u'_1 and w'_2 (again a part of it may be outside of T'_2). Consider the state S_2 of the robot at the beginning of the second long trip in T. In the tree T' under construction the robot is in the same state, at w'_2, after faithfully reproducing the short trip between u_1 and w_2 by an isomorphic run between u'_1 and w'_2 (provided that all nodes on this run which correspond to internal nodes of T have degree 3 in T', which we will guarantee at the end of the construction).

Consider the infinite proper tree containing the part of T' already constructed, with no leaves outside of T'_1 and T'_2. Consider the run r'_2 of the robot in this infinite tree starting at w'_2 in state S_2, until hitting a node u'_2 at distance b from w'_2. As before (see Claim), the robot does not visit any leaf of T'_2 during this run. Two cases are possible: either u'_2 is a leaf in T'_1 or u'_2 is some node of degree 3. In the first case we repeat the previous part of the construction, simulating the short trip of the robot following the second long trip in T by its run beginning at u'_2 and ending at a leaf w'_3 in T'_1. In the second case we continue the construction of the tree T' by adding the run r'_2 (with appropriate coloring of edges) to the previously constructed part of T'. Let T_3 be the principal subtree of T containing the leaf u_2 at the end of the second long trip in T. At the end of run r'_2 we add an isomorphic copy T'_3 of T_3 in such a way that the leaf u'_2 corresponding to u_2 is the node ending run r'_2, and we reproduce the coloring from T_3. (Notice that T_3 could be equal to T_1.)

We need to observe a property that will be crucial for continuing the construction inductively. Let p'_2 be the simple path between w'_2 and u'_2. Let p''_1 be the reverse of path p'_1. Let z be the last common node on these paths. Since there are at least $3K$ periods of edge colors on p'_1 (and hence also on p''_1) between the root of T'_1 and the root of T'_2, it follows that z is before the middle of p''_1 (and hence also of p'_2). Indeed, if it were in the second half of this path, there would be more than K periods before z. Consequently, for some earlier period, at the first visit of a node z' corresponding to z in this period, the robot would be in the same state as at the first visit of z. Hence the last common node on paths p'_2 and p''_1 would be z' rather than z.

We continue the construction of T', adding new runs of the robot in the infinite tree (corresponding to long trips and to short trips in T) and adding trees T'_i isomorphic to principal subtrees of T at the end of each run corresponding to a long trip. Suppose that $T'_1, ..., T'_i$ are already constructed and that we start the ith run, corresponding to the ith long trip in T, at a leaf w'_i of T'_i (as before, until hitting a node at distance b from w'_i). We must show that this run cannot hit any leaf of $T'_1, ..., T'_{i-2}$ because these leaves may be at distances other than b from w'_i, and we could not argue that the state after the ith run is identical to that after the ith long trip in T. This can be shown as follows. Let p'_j, for $1 \leq j \leq i$, be the simple path between a leaf of T'_j and a leaf of T'_{j+1}, corresponding to the jth run, and let p''_j be the reverse of this path. Suppose that during the ith run the robot hits a leaf in some T'_m, for $m \leq i-2$. This means that p'_i deviates from p''_{i-1} at the node t in which p'_{i-1} deviates from p''_{i-2}. However, similarly as argued before, t is before the middle of p'_{i-1}. On the other hand, t must be before the middle of p'_i, hence also before the middle of p''_{i-1}, i.e., after the middle of p'_{i-1}, which gives a contradiction. (Notice that the ith run may hit a leaf of T'_{i-1} but this is not a problem, because such a leaf is at distance b from w'_i.)

We add as many runs and trees T'_j in the above described manner as there are long and short trips in the operation of the robot in tree T. Suppose that after the run corresponding to the last long trip the robot is in leaf u'_i of tree T'_{i+1} in state S^*. Consider the behavior of the robot in tree T when it is situated in leaf u_i finishing the last long trip, and is in state S^*. The robot traverses some trajectory without ever hitting leaves in principal subtrees other than that to which u_i belongs (i.e., it makes another short trip), and then stops at some node. Add an isomorphic copy of this final short trip to the tree T' under construction (again some nodes of it may be outside of T'_{i+1}). Now the construction is almost finished. Let $\hat{T}$ be the tree constructed so far. Its maximum degree is 3. Let L be the set of all its leaves which are not leaves of any tree T'_j (these are nodes in which the robot returned during runs corresponding to long or short trips). Let M be the set of nodes of degree 2 in $\hat{T}$. To each node in L add two new neighbors of degree 1, and to each node in M add a new neighbor of degree 1. The resulting tree is T'.

The tree T' is a proper tree with the following property: all internal nodes of a run corresponding to a long trip in T are of degree 3. Hence, in view of the choice of integers a and b, the state of the robot in the leaf of T' ending a given run is the same as the state of the robot in the leaf of T ending the corresponding long trip. The definition of the final trajectory (after leaving u'_i) implies that the robot must stop at the end of it. However, it did not explore the entire tree T': indeed, it only explored the subtree $\hat{T}$, and $T' \setminus \hat{T}$ is nonempty because either the set L or the set M are nonempty. This concludes the proof.

The following result can be obtained by estimating the size of the tree T' (constructed in the above proof), which the robot fails to explore with stop.

THEOREM 3.2. *A robot which can explore with stop any n-node tree of maximum degree 3 must have $\Omega(\log\log\log n)$-bit memory.*

Proof. We use the notation and terminology from the proof of Theorem 3.1. Consider a simple path p'_i of length b joining two ends of a run r' of the robot in T', corresponding to a long trip in T.

Claim. During the run r' the robot is never at distance larger than K from the path p'_i.

Suppose it is, and let w be a node visited by the robot during run r', at distance larger than K from the closest node v of p'_i. Consider the state of the robot at the first visit of each node on the path q joining v and w. There are two internal nodes v_1 and v_2 on q for which this state is the same. Let v_1 be the node first visited before v_2 and let β be the distance between v_1 and v_2. Since after first visiting v_1 the robot made progress β on the path q before visiting the leaf u'_i ending p'_i, after first visiting v_2 it will make further progress β on this path before visiting u'_i because the state at the first visit of v_1 and v_2 was the same. The robot will keep going farther from v first on q and then on its extension in the infinite tree, before visiting u'_i. Hence the first node at distance b from w'_i visited by the robot will be outside of p'_i and hence different from u'_i, which contradicts the definition of u'_i. This proves the claim.

In the same way we can prove that during runs corresponding to short trips the robot is never farther than at distance K from the closest leaf of the tree T'_i where this run starts and ends.

Let T^* be the union of all trees T'_i and all simple paths p'_i defined during the construction of T'. Attach to each node u of degree 2 of T^* a new neighbor u' and attach to it a complete binary tree of height $K+1$ rooted at u'. Denote the resulting proper tree by T^{**}. The above claim and the remark following it imply that T^{**} includes T'.

We now estimate the size of T^{**}. Every path p'_i has length $b \leq K + 3 \cdot 3K^2(3K)^{3K}$. and every tree T'_i has size smaller than $2^a < 2^b$. There are at most K paths p'_i and at most $K+1$ trees T'_i. The total length of all paths is at most Kb and a tree of size at most 2^{K+2} is attached to each of their nodes. Consequently the number $n = 2^{(4K)^{4K}}$ is an overestimate of the size of T^{**} and hence also of T'. Hence a robot with k-bit memory, where $2^k = K$, fails to explore with stop some tree of maximum degree 3 and of size $n = 2^{(4K)^{4K}}$. This implies that in order to explore with stop all n-node trees of maximum degree 3, the robot must have $\Omega(\log\log\log n)$-bit memory.

4 Exploration with return

We now turn attention to the most demanding of our exploration tasks, when we require the robot to stop at the starting node after performing exploration. For this task we have a much sharper lower bound on memory than for exploration with stop. In fact, even a robot specifically constructed for exploration with return of an *a priori given* tree requires logarithmic memory.

THEOREM 4.1. *Let T be any tree of size n. A robot which can explore with return the tree T must have $\Omega(\log n)$-bit memory.*

Proof. Suppose that a robot with $k < \lfloor \log n \rfloor$ memory bits can perform exploration with return in T. Fix any node v of T. For any node u, let S_u be the state of the robot at the first visit of v when u is the node at which the robot starts. Since the robot has fewer than n states, for two different starting nodes u_1 and u_2 we have $S_{u_1} = S_{u_2}$. Hence the part of the trajectory of the robot after the first visit of v is the same, regardless of whether it started at u_1 or at u_2. Consequently, the node at which the robot stops is the same in both cases. This cannot be both u_1 and u_2, which contradicts the definition of exploration with return.

As for upper bounds, we present two algorithms. The first, Algorithm STACK-OF-PORTS, is a natural exploration with return, based on depth-first search. The label of the port by which a robot enters a new node is pushed on a stack and popped after the last visit of the node.

Algorithm STACK-OF-PORTS. The robot can be in two modes: F (forward) or B (backward) and uses a stack of integers.

1. The robot starts in mode F in some node v of degree d. It pushes a special symbol $\star$ on the empty stack and leaves v by port 1.
2. When the robot enters an internal node w of degree d_w by port i in mode F, it pushes the integer i on the stack and leaves by port $(i \bmod d_w)+1$.
3. When the robot enters a leaf it switches to mode B and returns to the neighbor of this leaf.
4. When the robot enters an internal node w of degree d_w by port i in mode B it compares i to the top of the stack.
   (a) If the top of the stack is an integer j and $i \not\equiv j-1 \bmod d_w$ then the robot switches to mode F and leaves by port $(i \bmod d_w)+1$.
   (b) If the top of the stack is an integer j and $i \equiv j-1 \bmod d_w$ then the robot pops the stack and leaves by port j.
   (c) If the top of the stack is $\star$ and $i < d_w$ then the robot switches to mode F and leaves by port $(i \bmod d_w)+1$.
   (d) If the top of the stack is $\star$ and $i = d_w$ then the robot stops.

The height of the stack never exceeds the diameter D of the tree and each label uses $O(\log \Delta)$ memory bits, where Δ is the maximum degree. Hence we have:

THEOREM 4.2. *Algorithm* STACK-OF-PORTS *accomplishes exploration with return of any tree of diameter D and maximum degree Δ using $O(D \log \Delta)$ bits of memory.*

In fact the bound on the memory required by Algorithm STACK-OF-PORTS can be slightly improved by observing that the total number of bits in all elements simultaneously on the stack is at most $O(\log c_1 + \cdots + \log c_k)$, where $c_1, ..., c_k$ is a sequence of degrees of nodes on a branch of the tree. However, this number of bits can still be linear in the size of the tree. This should be compared to our main algorithm for exploration with return which we present below. It accomplishes exploration with return in any n-node tree using only $O(\log^2 n)$ memory bits.

We first describe a family of exploration protocols. Each protocol is designed for a specific value of the number of nodes. A protocol running for any value of n while preserving the nice features of this family will be presented later. We need the following concepts:

DEFINITION 4.1. *Given a tree T, a node s of T, and $\ell \in \{0, 1, \ldots, \deg(s)\}$, the* open subtree *$T_{s,\ell}$ of T is T if $\ell = 0$, and it is the subtree of T consisting of the connected component of T containing s after the removal of its incident edge corresponding to port ℓ, otherwise.*

Exploring an open subtree $T_{s,\ell}$ means exploring T from s if $\ell = 0$, and otherwise it consists of the following two tasks: (1) explore $T_{s,\ell}$ from s and return to s, and (2) leave s through port ℓ. Note that $T_{s,\ell}$ might be a subtree of an unknown tree. Note also that the case $\ell = 0$ would not need a special treatment if we assumed that the expression "leave through port 0" means "stay here" (we recall that ports are labeled from 1 to the degree of the node).

LEMMA 4.1. *There exists a family of exploration protocols $\{\mathcal{E}_k, k \geq 0\}$, satisfying the following properties:*

1. *$\mathcal{E}_k$ allows exploration of any n-node open subtree $T_{s,\ell}$ of any unknown tree, for $n \leq 2^k$.*

2. *When the exploration terminates, the robot reports* SUCCESS *if $n \leq 2^k$, and* FAILURE *otherwise.*

3. *If the robot reports* SUCCESS *then it knows n.*

Proof. The construction of the family $\{\mathcal{E}_k, k \geq 0\}$ is by induction on k. Let us start by the description of the trivial protocol $\mathcal{E}_0$. We consider exploration of an open subtree $T_{s,\ell}$ by a robot $\mathcal{R}$. The case $\ell = 0$ is straightforward: if $\deg(s) = 0$, then $\mathcal{R}$ reports SUCCESS, otherwise it reports FAILURE. The case $\ell > 0$ is not much more involved: $\mathcal{R}$ leaves through port ℓ and reports FAILURE if $\deg(s) \geq 1$ and SUCCESS otherwise. It is easy to check that $\mathcal{E}_0$ satisfies all the properties of the lemma.

Let us now assume that we have constructed a sequence $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_k$ of protocols satisfying the properties of the lemma, and let us construct the protocol $\mathcal{E}_{k+1}$. We will later prove that $\mathcal{E}_{k+1}$ is correct, i.e., satisfies the properties of the lemma. *Construction.* We consider exploration of an open subtree $T_{s,\ell}$. The robot $\mathcal{R}$ can be in three states: EXPLORE, SUCCESS, or FAILURE. It starts in the state EXPLORE. $\mathcal{R}$ uses a variable `number` which counts the number of nodes that have been visited so far, and a stack `parent` used to store integers of variable size. It also uses a variable `hop-count` which counts the distance between s and the current position of $\mathcal{R}$. Initially `hop-count` $= 0$, `number` $= 1$, and `parent` is empty. In addition to these variables $\mathcal{R}$ also uses a constant number of variables to store port-labels.

We describe the actions of $\mathcal{R}$ in state EXPLORE, and currently at node u (possibly $u = s$). Let p be the port through which $\mathcal{R}$ entered u ($p = \ell$ if $u = s$). For $i = 1, \ldots, \deg(u)$, let v_i be the neighbor of u connected to u by the link corresponding to port i at u. Viewing T as rooted at u, let $T^{(i)}$ be the subtree of T rooted at v_i. $\mathcal{R}$ successively tries to explore every $T^{(i)}$ with the $\mathcal{E}_j$'s, $j \leq k$, by increasing order of the index j. More precisely, for every $i = 1, \ldots, \deg(u)$ let q_i be the port label at v_i of the link connecting v_i to u. For $i = 1$ to $\deg(u)$, including $i = p$, $\mathcal{R}$ does the following. Starting from $j = 0$, and increasing j by one in case of failure, $\mathcal{R}$ repeats (1) move to v_i, and (2) explore the open subtree $T^{(i)}_{v_i,q_i}$ with $\mathcal{E}_j$. This process is carried on until either there is $k_i \leq k$ for which $\mathcal{E}_{k_i}$ succeeds (i.e., for which $\mathcal{R}$ comes back to u in the state SUCCESS), or all $\mathcal{E}_j$'s, $0 \leq j \leq k$, fail to explore $T^{(i)}$. When $\mathcal{R}$ has completed the previous operations for each $T^{(i)}$, it continues as follows:

I1 If, for every subtree $T^{(i)}$, $i \neq p$, there is $k_i \leq k$ such that $\mathcal{E}_{k_i}$ succeeded to explore the open subtree $T^{(i)}_{v_i,q_i}$, then $\mathcal{R}$ switches to the state SUCCESS.

I2 If there is exactly one subtree $T^{(q)}$, $q \neq p$, for which all $\mathcal{E}_j$'s fail, then $\mathcal{R}$ stores q and stays in state EXPLORE.

I3 If there are at least two distinct subtrees $T^{(q)}$ and $T^{(q')}$, $q \neq p$ and $q' \neq p$, for which all $\mathcal{E}_j$'s fail, then $\mathcal{R}$ switches to the state FAILURE.

In the three cases, $\mathcal{R}$ sums the number of vertices of the subtrees $T^{(i)}$, $i \neq p$, whose exploration succeeded and adds it to `number`. If `number` $> 2^{k+1}$, then $\mathcal{R}$ switches to the state FAILURE (if not yet in that state). At this point of the exploration, two cases have to be considered, depending on whether $\mathcal{R}$ is in state EXPLORE or not.

Case 1: $\mathcal{R}$ is in state EXPLORE. In this case, only one subtree $T^{(q)}$ of u remains to be explored ($\mathcal{R}$ failed to explore that tree using $\mathcal{E}_k$, but exploration with $\mathcal{E}_{k+1}$ is potentially doable). Before leaving u, $\mathcal{R}$ proceeds as follows:

- If all $\mathcal{E}_j$'s, $0 \leq j \leq k$, fail to explore $T^{(p)}$, or if k_p is larger (strictly) than all the other k_i's, $i \neq q$, then nothing is stored;
- Otherwise, let $\nu_q(p)$ be the number of indices i such that either (i) $k_i > k_p$, or (ii) $k_i = k_p$ and $i \leq p$. The value of $\nu_q(p)$ is placed at the top of `parent`.

Once this is done, $\mathcal{R}$ increases `hop-count` by 1, and leaves through port q, to reach a new node u' from where the exploration carries on in the same way as it was performed at u.

Case 2: $\mathcal{R}$ is in state SUCCESS or FAILURE. Its objective is then to return back to s and to report the success or the failure of the exploration. For that purpose, $\mathcal{R}$ moves back along the edge labeled p. $\mathcal{R}$ will eventually return to s by using the information previously stored in `parent`. We describe bellow how this process is carried out.

Assume that $\mathcal{R}$ enters a node u via port q, in state SUCCESS or FAILURE, with `hop-count` $\neq 0$. $\mathcal{R}$ tries to compute the port p leading toward s. For that purpose, $\mathcal{R}$ repeats the same process as when $\mathcal{R}$ entered u in the state EXPLORE. In particular, $\mathcal{R}$ re-explores the subtrees rooted at u's neighbors. Again, viewing T as rooted at u, let $T^{(i)}$ be the subtree of T rooted at u's neighbor v_i, $i = 1, \ldots, \deg(u)$, and let q_i be the port label at v_i of the link connecting v_i to u. For $i = 1, \ldots, \deg(u)$, $i \neq q$, $\mathcal{R}$ does the following. Starting from $j = 0$, and increasing j by one in case of failure,

$\mathcal{R}$ repeats (1) move to v_i, and (2) explore the open subtree $T^{(i)}_{v_i,q_i}$ with $\mathcal{E}_j$. This process is carried on until either there is $k_i \leq k$ for which $\mathcal{E}_{k_i}$ succeeds, or all $\mathcal{E}_j$'s, $0 \leq j \leq k$, fail to explore $T^{(i)}$.

The index p is then obtained as follows:

- If there exists $i \neq q$ such that all $\mathcal{E}_j$'s, $0 \leq j \leq k$, fail to explore $T^{(i)}$, or if there exists $i \neq q$ such that k_i is larger (strictly) than all the other k_j's, $j \neq q$, then $p = i$;
- Otherwise the integer ν at the top of **parent** is popped out, and p is such that $\nu_q(p) = \nu$.

Once p has been identified, $\mathcal{R}$ decreases **hop-count** by 1, and leaves through port p.

When **hop-count** $= 0$, $\mathcal{R}$ leaves through port ℓ to report the success of the exploration together with the total number of nodes of the open subtree $T_{s,\ell}$ if it is in the state SUCCESS, or the failure of the exploration, if it is in the state FAILURE. This completes the description of $\mathcal{E}_{k+1}$. In the remaining, we show that this protocol satisfies the properties of the lemma.

Proof of correctness. If $T_{s,\ell}$ has at most 2^k nodes, then all explorations of open subtrees rooted at the neighbors v_i's of s, $i \neq \ell$, will succeed for some $\mathcal{E}_{k_i}$, $k_i \leq k$. Therefore, $\mathcal{E}_{k+1}$ will also succeed in this case by application of **I1**.

If $2^k < n \leq 2^{k+1}$, then, since $\mathcal{E}_k$ explores all trees of size at most 2^k, $\mathcal{R}$ can only apply **I1** or **I2** at s. If **I2** is applied, $\mathcal{R}$ moves to the root of the yet unexplored tree. $\mathcal{R}$ proceeds in this way until it reaches a node u^* in which every open subtree $T^{(i)}_{v_i,q_i}$, $i \neq p$, will be successfully explored by some $\mathcal{E}_{k_i}$, $k_i \leq k$. This will eventually occur because every time the robot goes "down" through the yet unexplored tree, the number of nodes of the tree currently explored decreases by at least one. Since all open subtrees $T^{(i)}_{v_i,q_i}$, $i \neq p$, of u^* have been explored, all nodes of the open subtree $T_{s,\ell}$ have been visited. At u^* the robot applies **I1** and enters state SUCCESS.

If $n > 2^{k+1}$, then $\mathcal{R}$ will eventually reach a node u^* where either two subtrees cannot be explored with $\mathcal{E}_k$ or the total number of nodes visited so far exceeds 2^{k+1}. In the former case, the robot applies **I3** and enters state FAILURE. In the latter case, since **number** $> 2^{k+1}$, the robot enters state FAILURE as well.

It remains to check that the robot returns to s from u^*, and then from s moves through link ℓ.

When **I1** or **I3** is executed, $\mathcal{R}$ still knows the label p of the port leading to s, and the protocol specifies that $\mathcal{R}$ leaves through that port. Now, let u be the current position of $\mathcal{R}$, and assume that $\mathcal{R}$ entered u through port q. $\mathcal{R}$ tries to find the port p leading to s by re-exploring the open subtrees $T^{(i)}$ rooted at u's neighbors. Since **I2** was applied in u when $\mathcal{R}$ was in state EXPLORE, there is at most one open subtree $T^{(i)}$ distinct from $T^{(q)}$ whose exploration may fail with $\mathcal{E}_k$. Indeed, $T_{s,\ell}$ is an open subtree of an unknown tree that can be arbitrarily large, and hence exploring $T^{(p)}$ may fail. However, when $\mathcal{R}$ was in state EXPLORE, it was able to explore all the other open subtrees $T^{(i)}$, $i \notin \{p,q\}$, each with some $\mathcal{E}_{k_i}$, $k_i \leq k$. From these remarks, it is easy to check that p can be determined either by using some asymmetry (i.e., exploring $T^{(p)}$ fails, or k_p is the largest of all k_i's) or simply by reading its index when ports are sorted according to the k_i's. Since $\mathcal{R}$ increases (resp. decreases) **hop-count** by one at every move away from (resp. towards) s, $\mathcal{R}$ is back at s when **hop-count** $= 0$. This completes the proof of the first property of the lemma.

The second property follows directly from the fact that if **number** $> 2^{k+1}$ then $\mathcal{R}$ enters state FAILURE.

If $\mathcal{R}$ returns to s in the state SUCCESS, then the whole tree has been explored, and the number of nodes has been accumulated in the variable **number**. Thus the third property is also satisfied, which completes the proof of the lemma.

We are now ready to present our main algorithm of exploration with return. Given the family $\{\mathcal{E}_k, k \geq 0\}$ described in Lemma 4.1, our protocol can simply be described as follows.

Algorithm EXPLORE. The robot starting at node s of a tree T successively executes $\mathcal{E}_k$, $k = 0, 1, \ldots$, until there is a k for which it returns to s in the state SUCCESS. Then it stops.

THEOREM 4.3. *Algorithm* **EXPLORE** *accomplishes exploration with return of any n-node tree using $O(\log^2 n)$ bits of memory.*

Proof. By Lemma 4.1, if $k = \lceil \log_2 n \rceil$, then $\mathcal{E}_k$ completes exploration with return of T. It remains to show that **EXPLORE** requires $O(\log^2 n)$ bits of memory for n-node trees.

Assume that $\mathcal{E}_k$ uses M_k bits of memory, and compute the memory requirement for $\mathcal{E}_{k+1}$. First, $\mathcal{E}_{k+1}$ reserves M_k bits of memory for the execution of the $\mathcal{E}_j$'s, $j \leq k$. In addition to that, $\mathcal{E}_{k+1}$ makes use of two counters (**number** and **hop-count**), each on $k+1$ bits since $\mathcal{E}_{k+1}$ does not count more than 2^{k+1} nodes, and does not explore further than 2^{k+1} hops away from the initial position s. $\mathcal{E}_{k+1}$ makes also use of a constant number of integers to store the labels of ports: e.g., the incoming port, the outgoing port, the label of the open subtree currently under consideration, etc. Since labels are between 1 and the degree of the nodes, all these integers are on $\lceil \log_2 n \rceil$ bits. The storage of k_p (if the exploration of $T^{(p)}$ succeeded) requires $O(\log k)$ bits, and the computation of $\nu_q(p)$ requires $O(\log k)$ bits as well. The remaining part of the memory is used to store the stack parent. Assume that the stack stores indices $\nu_1, \dots, \nu_r$ as follows: we use two boolean strings, both of $\sum_{i=1}^{r} \lceil \log_2 \nu_i \rceil$ bits. One is just the concatenation of the binary expressions of the ν_i's, and the other indicates the separations between two successive ν_i's. From the specifications of $\mathcal{E}_{k+1}$, an index ν_i is placed in the stack parent every time there exists at least one index $j \notin \{p, q\}$ such that $k_j \geq k_p$. More precisely, if ν_i is stored, then there are at least $\mu_i = \max\{\nu_i, 2\}$ indices j for which $k_j \geq k_p$. Each such index corresponds to a subtree of size at least $2^{k_p - 1}$, while $T^{(p)}$ is of size at most 2^{k_p}. Hence, if $T \setminus T^{(q)}$ is of size N, then $T^{(p)}$ is of size at most $\frac{2N}{\mu_i + 1}$. Therefore, $\Pi_{i=1}^{r} \frac{\mu_i + 1}{2} \leq 2^{k+1}$, and hence $\sum_{i=1}^{r} \lceil \log_2 \nu_i \rceil = O(k)$. Thus the stack parent requires $O(k)$ bits to be stored. In total, we have

$$M_{k+1} = M_k + O(k) + O(\log n) = O(k(k + \log n)).$$

Since Algorithm EXPLORE involves $\mathcal{E}_k$ for $k \leq \lceil \log_2 n \rceil$, we get that its memory requirement is at most $O(\log^2 n)$, which completes the proof.

5 Conclusion

The following table is a summary of our results. It gives upper and lower bounds on memory size (in # of bits) of a robot for three types of exploration (perpetual, with stop, with return) of a tree on n nodes, with maximum degree Δ. The upper bound for exploration with stop assumes that the robot knows an upper bound m on the number of nodes of a given degree. Without this knowledge, the best we know is performing exploration with return, using $O(\log^2 n)$ memory bits.

Exploration	Upper Bound	Lower Bound
Perpetual	$O(\log \Delta)$	$\lceil \log \Delta \rceil$
(with) Stop	$O(\log \Delta + \log m)$	$\Omega(\log \log \log n)$
(with) Return	$O(\log^2 n)$	$\Omega(\log n)$

The table shows two interesting open problems. Firstly, what is the exact complexity in terms of memory bits of exploration with return? Secondly, can we perform exploration with stop using stricly less memory space than exploration with return?

6 Acknowledgements

This work was done during K. Diks' stay at the Université du Québec à Hull, and while P. Fraigniaud was visiting Carleton University in Ottawa. P. Fraigniaud is supported by NATO. E. Kranakis is supported in part by NSERC grant. Andrzej Pelc is supported in part by NSERC grant OGP 0008136. The three last authors were supported in part by MITACS (Mathematics of Information Technology and Complex Systems) project CANCOM grant.

References

[1] S. Albers and M. R. Henzinger, Exploring unknown environments, SIAM Journal on Computing 29 (2000), 1164-1188.

[2] B. Awerbuch, M. Betke, R. Rivest and M. Singh, Piecemeal graph learning by a mobile robot, Proc. 8th Conf. on Comput. Learning Theory (1995), 321-328.

[3] E. Bar-Eli, P. Berman, A. Fiat and R. Yan, On-line navigation in a room, Journal of Algorithms 17 (1994), 319-341.

[4] M.A. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, Proc. 30th Ann. Symp. on Theory of Computing (1998), 269-278.

[5] M.A. Bender and D. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, Proc. 35th Ann. Symp. on Foundations of Computer Science (1994), 75-85.

[6] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen and M. Saks, Randomized robot navigation algorithms, Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (1996), 74-84.

[7] A. Blum, P. Raghavan and B. Schieber, Navigating in unfamiliar geometric terrain, SIAM Journal on Computing 26 (1997), 110-137.

[8] M. Betke, R. Rivest and M. Singh, Piecemeal learning of an unknown environment, Machine Learning 18 (1995), 231-254.

[9] X. Deng, T. Kameda and C. H. Papadimitriou, How to learn an unknown environment I: the rectilinear case, Journal of the ACM 45 (1998), 215-245.

[10] X. Deng and C. H. Papadimitriou, Exploring an unknown graph, Journal of Graph Theory 32 (1999), 265-297.

[11] C.A. Duncan, S.G. Kobourov and V.S.A. Kumar, Optimal constrained graph exploration, Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (2001), 807-814.

[12] P. Panaite and A. Pelc, Exploring unknown undirected graphs, Journal of Algorithms 33 (1999), 281-295.

[13] C. H. Papadimitriou and M. Yannakakis, Shortest paths without a map, Theoretical Computer Science 84 (1991), 127-150.

[14] N. S. V. Rao, S. Hareti, W. Shi and S.S. Iyengar, Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms, Tech. Report ORNL/TM-12410, Oak Ridge National Laboratory, July 1993.