Fast Graph Exploration by a Mobile Robot

Ajay D. Kshemkalyani  
Department of Computer Science  
University of Illinois at Chicago  
Chicago, USA  
Email: ajay@uic.edu

Faizan Ali  
Department of Computer Science  
University of Illinois at Chicago  
Chicago, USA  
Email: fali28@uic.edu

Abstract—Given an undirected, anonymous, port-labeled graph of \( n \) nodes, \( m \) edges, and degree \( \Delta \), we require a mobile robot to visit all the nodes in the graph. We give two algorithms to solve this problem in \( O(m) \) time. Algorithm Robot-Memory uses \( O(n \log \Delta) \) bits at the robot, and 1 bit at each node. Algorithm Node-Memory uses \( O(\log \Delta) \) bits at each node. The algorithms Robot-Memory and Node-Memory trade-off the memory requirements at the robot and at the nodes. The algorithms are capable of perpetual exploration or patrolling, and are variants of depth-first search (DFS). The algorithms fill in existing gaps in the trade-offs between robot memory, memory at each node, and exploration time, in the body of literature on the graph exploration problem.

Keywords—Mobile robot; graph exploration; graph traversal;

I. INTRODUCTION

A. Background

We are given an undirected graph \( G \) with \( n \) nodes, \( m \) edges, and diameter \( D \). The maximum degree of any node is \( \Delta \). The graph is anonymous, i.e., nodes do not have unique identifiers. At any node, its incident edges are uniquely identified by a label in the range \([0, \delta - 1]\), where \( \delta \) is the degree of that node. We refer to this label of an edge at a node as the port number at that node. We assume no correlation between the two port numbers of an edge. There is a single robot, whose objective is to visit all the nodes in the graph, and detect when this has been done. The algorithms should also be capable of repeated exploration (or patrolling) of the graph.

Graph exploration and patrolling by a robot is important for mission-critical applications such as rescue and surveillance in dangerous, harmful, and unknown environments.

B. Related Work

In the graph exploration problem, the objective is to visit all the nodes of the graph. There are many results for this problem. We do not allow the use of pebbles (pebbles are markers that can be left and collected from the nodes by the mobile robot). Several other works assume specific topologies such as trees [1], [3], [4], [6] or directed graphs. Panaite and Pelc [8] provide a fast algorithm that has \( m + 3m \) steps (time complexity) but uses node labels, i.e., the graph is not anonymous. Variants of the Panaite and Pelc algorithm, in which the agent assigns unique labels to nodes, are given in [9]. We do not compare with such works.

The following are the closest works on exploration in general graphs in our model. An adaptation of standard depth-first search (DFS) to a distributed system uses 1 bit at the robot and \( O(\Delta) \) bits per node. Fraigniaud et al. [5] showed that using only memory at a robot, the robot can explore an anonymous graph using \( \theta(D \log \Delta) \) bits based on an increasing depth-restricted DFS. They did not analyze the time complexity to visit all the nodes, which turns out to be \( \sum_{i=1}^{D} O(\Delta^i) = O(\Delta^{D+1}) \) which is very high. Their algorithm has no mechanism to avoid getting caught in cycles and the only way out of cycles is the depth-restriction on the DFS. The robot also requires knowledge of \( D \) to terminate. If \( D \) is not known, the algorithm will continue with \( x \)-depth restricted DFSs, for \( x \geq D + 1 \), without ever knowing that all the nodes have been visited. Cohen et al. [2] gave two DFS-based algorithms with \( O(1) \) memory at the nodes. The first algorithm uses \( O(1) \) memory at the robot and 2 bits memory at each node to traverse the graph. The 2 bits memory at each node is initialized by short labels in a pre-processing phase which takes time \( O(mD) \). Thereafter, each traversal of the graph takes up to \( 20m \) time steps. The second algorithm uses \( O(\log \Delta) \) bits at the robot and 1 bit at each node to traverse the graph. The 1 bit memory at each node is initialized by short labels in a pre-processing phase which takes time \( O(mD) \). Thereafter, each traversal of the graph takes up to \( O(\Delta^{10m}) \) time steps. Using the Rotor-Router algorithm [10] allowing only \( O(\log \Delta) \) bits per node, an oblivious robot (i.e., robot is not allowed any memory) that also has no knowledge of the entry port when it enters a node, can explore an anonymous port-labeled graph in \( 2nD \) time steps [10]. Menc et al. [7] proved a lower bound of \( \Omega(mD) \) on the exploration time steps for the Rotor-Router algorithm.

C. Our Results

Our results assume that the undirected graph, with \( m \) edges, \( n \) nodes, diameter \( D \), and degree \( \Delta \), is anonymous, i.e., nodes have no labels. However, the ports (leading to incident edges) at a node have locally unique labels.
We provide two time-efficient algorithms to solve graph traversal in our system model. As opposed to the $O(\Delta^{D+1})$ [5] or $20m + O(mD)$ [2] or $O(\Delta^{10}m) + O(mD)$ [2] or $O(mD)$ [10] time steps algorithms in the literature, all our algorithms are fast, requiring $4m - 2n + 2$ time steps. Both our algorithms are capable of ongoing exploration of the graph, i.e., perpetual exploration or repeated exploration of the graph, also known as graph patrolling. The following is an overview of our algorithms; the upper bound results are given in Table I.

1) We present algorithm Robot-Memory which needs $O(n \log \Delta)$ bits at the robot and 1 bit memory at each node, running in $4m - 2n + 2$ time steps per traversal. The algorithm is capable of repeated traversals of the graph.

2) Algorithm Node-Memory uses $O(\log \Delta)$ bits at each node and runs in $4m - 2n + 2$ time steps per traversal. If perpetual exploration is to be achieved, as opposed to a single traversal of the graph, then one bit is needed at the robot.

Both algorithms are variants of depth-first search (DFS) and their complexity is summarized in Table I along with that of the existing works in the literature. Although our algorithms are simple, we are not aware of any literature which has published these DFS variants. The algorithms fill in existing gaps in the trade-offs between robot memory, memory at each node, and exploration time, in the body of literature on the graph exploration problem.

### Table I

**Comparison of the Algorithms for Exploration on Graphs.**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Robot Memory</th>
<th>Memory at Each Node</th>
<th>Traversal Time Steps</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed DFS</td>
<td>$O(\Delta)$</td>
<td>- -</td>
<td>$2m$</td>
<td>distributed DFS</td>
</tr>
<tr>
<td>Fraigniaud et al. [5]</td>
<td>$O(D\log \Delta)$</td>
<td>- -</td>
<td>$O(\Delta^{D+1})$</td>
<td>need to know $D$ for termination</td>
</tr>
<tr>
<td>Cohen et al. [2]</td>
<td>$O(1)$</td>
<td>2</td>
<td>$20m + O(mD)$</td>
<td>$O(mD)$ time steps for pre-processing</td>
</tr>
<tr>
<td></td>
<td>$O(\log \Delta)$</td>
<td>1</td>
<td>$O(\Delta^{10}m) + O(mD)$</td>
<td>$O(mD)$ time steps for pre-processing</td>
</tr>
<tr>
<td>Rotor-Router [7], [10]</td>
<td>- -</td>
<td>$O(\log \Delta)$</td>
<td>$O(mD)$</td>
<td>in-port agnostic</td>
</tr>
<tr>
<td>Robot-Memory</td>
<td>$O(n \log \Delta)$</td>
<td>1</td>
<td>$4m - 2n + 2$</td>
<td>can do patrolling</td>
</tr>
<tr>
<td>Node-Memory</td>
<td>- -</td>
<td>$O(\log \Delta)$</td>
<td>$4m - 2n + 2$</td>
<td>$O(1)$ memory at robot needed for patrolling</td>
</tr>
</tbody>
</table>

II. **Traversal Using Robot Memory**

Algorithm 1 (Robot-Memory) gives the code for a robot to perpetually traverse the graph. In addition to the three persistent variables: (i) odd, (ii) state, and (iii) stack, the robot uses a non-persistent variable port$_{entered}$ to track the port through which it enters (and then exits) a node, except for the initial start node where this is set to $-1$ (line 10). Variable odd enables perpetual traversal; in odd numbered traversals, this is set to 1 and in even-numbered traversals, this is set to 0. The value of odd is flipped when a complete traversal of the graph is detected, in lines (23)-(24). Each node uses a single boolean variable, visited, which is used to track whether the node has been visited in the current traversal. In conjunction with odd, the boolean visited takes on different semantics in odd-numbered and even-numbered traversals. After an odd-numbered traversal, the visited value at each node is 1, and for the next (even-numbered) traversal, this initial value of 1 should be treated as “not visited”. Thus, in odd-numbered (even-numbered) traversals, visited $\equiv 0 (\equiv 1)$ means the node has not yet been visited by the robot. The stack is used to track the sequence of port numbers through which the DFS path traced by the robot can be backtracked. The port$_{entered}$ is pushed onto stack in forward exploration (line (16)), and popped from stack in backtracking (lines (19) and (28)). At the start node, port$_{entered}$ is set to $-1$ (line (10)) and this value is pushed on to stack (line (16)); so bottom of stack is $-1$.

At the initial node, the robot sets port$_{entered}$ to $-1$; else when the robot enters a node, it sets port$_{entered}$ to the entry port (lines 7-10).

- (lines 11-20) If the state is explore, then if the node has been visited before, the robot changes state to backtrack and moves back through the entry port (lines 12-14). Otherwise, the robot sets visited to odd to mark that the node has been visited, pushes port$_{entered}$ (the parent pointer of the node) onto the stack, and increments port$_{entered}$ in a modulo fashion. It then moves through port$_{entered}$ to continue forward exploration, unless the new value of port$_{entered}$ equals the entry port, in which case there is no further graph to explore from this node, and the robot backtracks through the entry port after changing state to backtrack and popping the stack.
- (lines 21-29) If the state is backtrack, the robot increments port$_{entered}$ in a modulo fashion, changes state to explore and resumes forward exploration by moving through port$_{entered}$, unless the new value of port$_{entered}$ is the parent pointer of the node (top(stack)), in which case the robot remains in state backtrack, pops the stack, and backtracks through port$_{entered}$. The robot also checks for complete traversal of the graph (line 23) and if detected, flips the odd bit.
Algorithm 1 Robot-Memory, code at robot $i$

1: Variables at robot:
2:   odd $\leftarrow 1 \in \{0,1\}$
3:   state $\leftarrow$ explore $\in \{\text{explore, backtrack}\}$
4: stack $\leftarrow \perp$ of type $\{-1,0,1,\cdots,\log \Delta - 1\}$

5: Variables at a node:
6: visited $\leftarrow 0 \in \{0,1\}$

7: if robot moves to current node then
8:     port_entered $\leftarrow$ entry port
9: else
10:     port_entered $\leftarrow -1$
11: if state $=$ explore then
12:   if visited $=$ odd then
13:     state $\leftarrow$ backtrack
14:     move through port_entered
15: visited $\leftarrow$ odd
16: push(stack, port_entered)
17: port_entered $\leftarrow (port_{entered} + 1) \mod \delta$
18: if port_entered $=$ top(stack) then
19:     state $\leftarrow$ backtrack; pop(stack)
20: move through port_entered
21: else if state $=$ backtrack then
22:     port_entered $\leftarrow (port_{entered} + 1) \mod \delta$
23:     if top(stack) $=$ -1 AND port_entered $=$ 0 then
24:         odd $\leftarrow$ odd
25:     if port_entered $\neq$ top(stack) then
26:         state $\leftarrow$ explore
27:     else
28:         pop(stack)
29: move through port_entered

Theorem 1: Algorithm 1 (Robot-Memory) achieves perpetual exploration of the graph in $O(m)$ steps per traversal, with $O(n \log \Delta)$ bits at each robot and 1 bit per node.

Proof: Observe that the robot executes a variant of a DFS in the traversal of the graph. The robot traverses each edge of the DFS two times (once forward, once backward), and each non-tree edge four times (once for exploration in each direction, and once for backtracking in each direction). So for a total of $4(m-(n-1))+2(n-1) = 4m-2n+2$ times. One traversal is completed when the start node is revisited (top(stack) = -1) and the next port via which to explore is port 0 (line (23)). The value of odd is then flipped (line (24)). Thus, the robot executes for $4m-2n+2$ steps in one traversal, so the running time is $O(m)$ per traversal.

Each node has a boolean, visited. The robot has booleans odd and state, and the stack. The maximum number of entries in stack is $n-1$ as the algorithm is a variant of DFS. Each stack entry is a port number at a visited node, and is thus bounded by $\log \Delta$ bits. Hence, the memory at the robot is bounded by $O(n \log \Delta)$.

The mechanism of flipping the value of odd for each new traversal (lines (23)-(24)), in conjunction with setting visited to odd each time a node is visited for the first time in a traversal (line (15)), allows for perpetual exploration of the graph.

III. TRAVERSAL USING NODE MEMORY

Algorithm 2 Node-Memory, code at robot $i$

1: Variables at robot (not needed for single exploration):
2:   odd $\leftarrow 1 \in \{0,1\}$

3: Variables at a node:
4: parent_ptr $\leftarrow -1 \in \{-1,0,1,\cdots,\log \delta - 1\}$
5: visited $\leftarrow 0 \in \{0,1\}$
6: port_last_forward $\leftarrow -1 \in \{-1,0,1,\cdots,\log \delta - 1\}$

7: if robot moves to current node then
8:     port_entered $\leftarrow$ entry port
9: else
10:     port_entered $\leftarrow -1$
11: if visited $=$ odd OR port_entered $\neq$ port_last_forward then
12:     move through port_entered
13: visited $\leftarrow$ odd
14: parent_ptr $\leftarrow$ port_entered
15: port_entered $\leftarrow (port_{entered} + 1) \mod \delta$
16: if port_entered $\neq$ parent_ptr then
17:     port_last_forward $\leftarrow$ port_entered
18:     move through port_entered
19: else if visited $=$ odd AND port_entered $=$ port_last_forward then
20:     port_entered $\leftarrow (port_{entered} + 1) \mod \delta$
21:     if parent_ptr $=$ -1 AND port_entered $=$ 0 then
22:         odd $\leftarrow$ odd
23:     if port_entered $\neq$ parent_ptr then
24:         port_last_forward $\leftarrow$ port_entered
25:         move through port_entered

Algorithm 2 (Node-Memory) gives the code for a robot to traverse the graph, with memory at the nodes. A single bit, odd, similar to that used in Algorithm 1, is used at the robot only if perpetual traversal is required. Three variables are used at each node: (i) parent_ptr: that is used to point to the parent node in the DFS traversal; (ii) visited: that indicates whether the node has been visited in the current traversal; and (iii) port_last_forward that in conjunction with port_entered (a temporary variable) and visited, is used to determine whether the robot is on its first visit to the node (i.e., robot is in explore state) or whether the robot is in
backtracking state, in this traversal. Variable odd at the robot enables perpetual traversal; in odd numbered traversals, this is set to 1 and in even-numbered traversals, this is set to 0. The value of odd is flipped when a complete traversal of the graph is detected, in lines (22)-(23). In conjunction with odd, the boolean visited takes on different semantics in odd-numbered and even-numbered traversals. After an odd-numbered traversal, the visited value at each node is 1, and for the next (even-numbered) traversal, this initial value of 1 should be treated as “not visited”. Thus, in odd-numbered (even-numbered) traversals, visited = 0 (=1) means the node has not yet been visited by the robot.

The boolean visited at each node is also necessary for the robot to know whether it is visiting a node in forward exploration mode or in backtracking mode. Rather than maintain a variable state at the robot, here port_last_forward is maintained at each node; it indicates the port number on which the robot most recently exited the node in forward exploration mode. It is updated in lines (18) and (25) just before moving out of the node in the forward exploration state. The initial value is –1. The robot visits the node in backtracking state if and only if (i) the node has been visited in the current exploration (i.e., visited = odd) and (ii) port_entered = port_last_forward. This test is used in line (20) to test for backtracking mode, and its complement is used in line (11) to test for forward exploration mode. This use of port_last_forward was given in [9].

At the initial node, the robot sets port_entered to −1; else when the robot enters a node, it sets port_entered to the entry port (lines 7-10).

• (lines 11-19) If the state is explore, determined by the test visited = odd OR port_entered ≠ port_last_forward, then after setting 0, the robot moves back through the entry port (lines 11-13). Otherwise, the robot sets visited to odd to mark that the node has been visited, sets parent_ptr to port_entered, and increments port_entered in a modulo fashion. It then moves through port_entered after setting port_last_forward to port_entered, to continue forward exploration, unless the new value of port_entered equals parent_ptr (the entry port), in which case there is no further graph to explore from this node, and the robot backtracks through the entry port port_entered.

• (lines 20-26) If the state is backrack, determined by the test visited = odd AND port_entered = port_last_forward, the robot increments port_entered in a modulo fashion, and resumes forward exploration by moving through port_entered after setting port_last_forward to port_entered, unless the new value of port_entered is the parent pointer of the node (parent_ptr), in which case the robot backtracks through port_entered. The robot also checks for complete traversal of the graph (line 22) and if detected, flips the odd bit.

Theorem 2: Algorithm 2 (Node-Memory) achieves perpetual exploration of the graph in \(O(m)\) steps per traversal, with \(O(\log \Delta)\) bits at each node and 1 bit at the robot. For a single traversal of the graph, no memory is required at the robot.

Proof: Observe that the robot executes a variant of a DFS in the traversal of the graph. The robot traverses each edge of the DFS tree two times (once forward, once backward), and each non-tree edge four times (once for exploration in each direction, and once for backtracking in each direction). So for a total of \(4(m - (n - 1)) + 2(n - 1) = 4m - 2n + 2\) times. One traversal is completed when the start node is revisited (parent_ptr = −1) and the next port via which to explore is port 0 (line (22)). The value of odd is then flipped (line (23)). Thus, the robot executes for \(4m - 2n + 2\) steps in one traversal, so the running time is \(O(m)\) per traversal.

Each node has a boolean, visited, and parent_ptr and port_last_forward variables of type port (\(\lfloor \log(\delta + 1) \rfloor\) bits each). Hence, the memory at each node is bounded by \(O(\log \Delta)\).

The single bit variable odd at the robot is required only for perpetual exploration. For a single traversal of the graph, it is not required. All instances of variable odd in the code get replaced by “1” and instances of odd get replaced by “0”. Lines (22)-(23) are not needed.

REFERENCES


