Journal of Algorithms **33,** 281–295 (1999)
Article ID jagm.1999.1043, available online at http://www.idealibrary.com on IDEAL®

Exploring Unknown Undirected Graphs

Petrişor Panaite and Andrzej Pelc*

Département d'Informatique, Université du Québec à Hull, Hull, Québec J8X 3X7, Canada
E-mail: petrisor@cadabratech.com, pelc@uqah.uquebec.ca

Received October 18, 1997

A robot has to construct a complete map of an unknown environment modeled as an undirected connected graph. The task is to explore all nodes and edges of the graph using the minimum number of edge traversals. The *penalty* of an exploration algorithm running on a graph $G = (V(G), E(G))$ is the worst-case number of traversals in excess of the lower bound $|E(G)|$ that it must perform in order to explore the entire graph. We give an exploration algorithm whose penalty is $O(|V(G)|)$ for every graph. We also show that some natural exploration algorithms cannot achieve this efficiency. © 1999 Academic Press

1. INTRODUCTION

A robot has to construct a complete map of an unknown environment modeled as an undirected connected graph. To this end it has to *explore* all nodes and edges of the unknown graph starting from some node and moving along edges. When the robot traverses an edge, it explores this edge and both of its ends. The robot does not have any *a priori* knowledge of the graph. During the exploration, the robot draws a partial map consisting of all nodes and edges that it has already explored. It assigns labels to these components and can recognize them when encountered again. In particular, after coming to an already explored node v incident to an explored edge e, the robot knows the location of v and of the other end of e on the partial map. At any stage of exploration the robot also knows the number of unexplored edges incident to an explored node but does not know the other ends of these edges. It has two options for leaving an explored node: either use a specific explored edge or an arbitrary unex-

* Andrzej Pelc was supported in part by NSERC Grant OGP 0008136.

0196-6774/99 $30.00

plored edge (chosen by the adversary). The latter assumption corresponds to the lack of knowledge concerning the unexplored part of the graph.

A natural measure of cost of an exploration algorithm running on a graph $G = (V(G), E(G))$ is the worst-case number of edge traversals it uses, taken over all starting points and all adversary decisions. The obvious lower bound on this cost is $|E(G)|$. This can be achieved for Eulerian graphs by an off-line algorithm provided with a labeled map of the graph, knowing the starting point and other ends of all edges incident to the currently visited node. However, in the unknown environment setting, an exploration algorithm will often have to use substantially more traversals, due to the topology of the graph (which may be non-Eulerian) and to the lack of information indicating an efficient method of exploring. The *penalty* of an exploration algorithm running on a graph $G = (V(G), E(G))$ is the worst-case number of traversals in excess of the lower bound $|E(G)|$. It is important to construct exploration algorithms whose penalty is small for every graph.

1.1. *Related Work*

Exploration and navigation problems for robots in an unknown environment have been previously studied by many researchers (cf. the survey [12]). There are two basic approaches to modeling these problems. In one of them a particular geometric setting is assumed, e.g., an unknown terrain with convex obstacles [5] or a room with polygonal [7] or rectangular [3] obstacles. In the second approach the environment is modeled as a graph and the robot may move only along its edges. This latter setting was further specified in two different ways. In [1, 8] 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, 6] the explored graph is undirected but an additional requirement is imposed that the robot has to come back to the starting point every so often, say for refueling. In all these papers the aim was to explore the environment as efficiently as possible. In particular, under the graph model, this goal translates into constructing exploration algorithms with the smallest possible penalty. Indeed, in [1, 8, 2, 6] such exploration algorithms were constructed and analyzed, and (nearly) matching lower bounds were also given. It turns out that due to the requirement of traversing edges only from tail to head in one case and due to the necessity of periodic returns in the other case, the smallest possibility penalty is relatively high in both situations.

The above work leaves out the natural scenario of exploring an unknown undirected connected graph without the extra requirement of periodic returns to the starting point. It is briefly observed in [6] that the simple depth-first search algorithm gives a penalty of at most $|E(G)|$ in this case

(cf. also the remark in [1] that a penalty of at most $3|E(G)|$ follows immediately from a result in [8]). However, this penalty of order of $|E(G)|$ may become quite large in the case of dense graphs. The aim of the present paper, which is an extended version of [10], is to give an exploration algorithm with the penalty linear in the number of nodes, not only in the number of edges.

1.2. *Our Results*

Our main result is an exploration algorithm working for any (unknown) undirected connected graph $G = (V(G), E(G))$ with penalty $O(|V(G)|)$. More precisely, we show that the penalty of our algorithm never exceeds $3|V(G)|$. We also show that two natural heuristics, GREEDY and depth-first search (DFS), do not achieve penalty $O(|V(G)|)$ for all graphs.

In both GREEDY and DFS (cf. [1], where analogous heuristics were defined in the directed case), the robot uses unexplored edges as long as possible and, when stuck at a node v, it uses a simple strategy to reach a *free* node v' (i.e., incident to yet unexplored edges). In the case of GREEDY, v' is the free node closest to v, while in the case of DFS, v' is the most recently visited free node. It turns out that both these choices are too naive. The vision of GREEDY is too local, while DFS does not make sufficient use of the knowledge of the explored subgraph, basing its decisions only on the order of visits.

As opposed to these simple heuristics, our algorithm explores the graph in the order given by a dynamically constructed tree. As above, the robot uses unexplored edges as long as possible. The place where our algorithm differs from GREEDY or DFS is in the choice of the free node to which the robot relocates after getting stuck. The idea is to bring the robot back to the node of the dynamically constructed tree at which it interrupted the construction of the tree. In doing this we want to control the number of traversals through already explored edges. Following the structure of the dynamic tree prevents the robot from being distracted from systematic exploration by free nodes situated close to it, which is the case for GREEDY. At the same time it excludes "temporal" rather than "geographic" preferences, which cause inefficiency of DFS.

It should also be mentioned that the methods and results from [1, 8] concerning directed graphs cannot be adapted to our scenario. This would require traversing each edge at least once in each direction, thus giving a penalty of at least $|E(G)|$, which we want to avoid. Also methods from [2, 6], based on breadth-first search exploration, natural and useful under the requirement of frequent returns to the starting point, are easily seen to give a penalty of at least $|E(G)|$ and thus cannot be used for our purpose.

The paper is organized as follows. In Section 2 we establish terminology. In Section 3 we describe the above-mentioned natural exploration heuristics and show that they do not achieve penalty $O(|V(G)|)$ for all graphs. In Section 4 we describe our exploration algorithm, prove its correctness, and show that its penalty is linear in the number of nodes of the graph.

2. TERMINOLOGY AND NOTATION

Throughout the paper, *graph* means undirected connected graph. The *order* of a graph is its numbers of nodes. For $G = (V(G), E(G))$ and $H = (V(H), E(H))$, let $G \cup H$ denote the graph $(V(G) \cup V(H), E(G) \cup E(H))$. If H is a subgraph of G, let $G \setminus H$ denote the graph induced in G by $V(G) \setminus V'(H)$, where $V'(H)$ is the set of those nodes in H, all of whose neighbors belong to $V(H)$.

A run of an exploration algorithm is a sequence of *edge traversals*, where consecutive edges form a path in the graph. At any stage of the algorithm execution, already traversed edges are called *explored* and other edges are called *free*. A node is *saturated* if all its incident edges are explored. Otherwise it is *free*. The robot is *stuck* after coming to node v if v becomes saturated after this move of the robot. Let $G = (V(G), E(G))$ be a graph, $v \in V(G)$, and let A be an exploration algorithm in G. A *penalty traversal* is any traversal of an explored edge. $\mathscr{P}_A(G, v)$ denotes the worst-case number of penalty traversals, taken over all possibilities to choose the unexplored edges (i.e., taken over all *adversaries*), when the robot starts at v and moves according to algorithm A. The *penalty of A for G* is $\mathscr{P}_A(G) = \max\{\mathscr{P}_A(G, v) : v \in V(G)\}$. An important property of an exploration algorithm A is that its penalty is *linear in the order of the graph*. By this we mean that there exists a constant c such that, for *every* graph G, $\mathscr{P}_A(G) \leq c \cdot |V(G)|$.

3. NATURAL HEURISTICS

In this section we consider two natural exploration heuristics, the GREEDY algorithm and the depth-first search algorithm, defined similarly as in [1]. Under our scenario these heuristics are, of course, much more efficient than those in [1]. However, our efficiency demand that the penalty be linear in the order of the graph is also much more difficult to satisfy. We show that none of these heuristics meet this requirement. It is worth mentioning that (as opposed to [1]) graphs constructed to prove this are totally different in each case. They are also different from graphs constructed in [1].

The GREEDY exploration algorithm is defined as follows. Whenever the current node is free, the robot takes an unexplored edge incident to the node. When stuck at the current node, the robot relocates to the nearest visited free node via a shortest path, where the *distance d* and the shortest path are defined with respect to the explored part of the graph.

We prove that the penalty of GREEDY is not linear in the order of the graph. To this end, we construct a class of graphs for which this algorithm performs poorly. Intuitively, the way of forcing GREEDY to perform many penalty traversals is the following. At many stages of the exploration, the adversary creates free visited nodes which are sufficiently close to the current node to "attract" the robot far away from a still unexplored part of the graph. In order to come back to this unexplored "hole," the robot performs many penalty traversals. Our graphs contain special devices called *magnets* whose role is to drag the robot away from such holes many times.

Let $g \geq 1$. A *g-magnet* is any graph isomorphic to $M_g = (\{0, \ldots, 4g\}, \{[0,1], [1,2], \ldots, [4g-1, 0], [2g, 4g]\})$. Hence a g-magnet is a cycle of length $4g$ with one extra node of degree 1 attached to it. The node of degree 3 is called the *center* of the magnet and the node of degree 1 is called its *top*. The node at distance $2g$ from the center is called the *root* of the magnet. A 0-*magnet* is any graph isomorphic to $M_0 = (\{0,1\}, \{[0,1]\})$, i.e., a two-node path. One of its nodes is called the root and the other is called the top. For convenience, we say that the root of a 0-magnet is also its center.

A graph S containing exactly one elementary path between two nodes u and v is called a *uv-segment*. This unique path, denoted $R_{uv}(S)$, is called the *uv-road* of S. For $w \in V(R_{uv}(S))$, let $G_{uv}(S, w)$ denote the connected component of $S \setminus R_{uv}(S)$ containing the node w, if such a component exists. Otherwise, let $G_{uv}(S, w) = \emptyset$. Because of the unicity of the path connecting u and v, the graphs $G_{uv}(S, w)$ and $G_{uv}(S, w')$ do not share any node, for distinct nodes w and w' on the uv-road of S.

Let S be a uv-segment and Z be a vw-segment such that $V(S) \cap V(Z) = \{v\}$. The *join* of S and Z, denoted $S \diamond Z$, is the graph $S \cup (Z \setminus G_{vw}(Z, v))$. Notice that $S \diamond Z$ is a uw-segment and, for all $x \in V(R_{uv}(S))$, $y \in V(R_{vw}(Z)) \setminus \{v\}$, we have $G_{uv}(S, x) = G_{uw}(S \diamond Z, x)$ and $G_{vw}(Z, y) = G_{uw}(S \diamond Z, y)$.

Suitably relabeling nodes of S and Z, the join operation can be extended to any pair uv-segment/vw-segment, with $u \neq w$, such that the result is always a uw-segment.

Let M and N be g-magnets, $g \geq 0$, with roots u and v, respectively. If S is a uv-segment such that $V(M) \cap V(S) = \{u\}$ and $V(S) \cap V(N) = \{v\}$ then $M \diamond S = M \cup S$ and $S \diamond N = S \cup N$. Clearly, $M \diamond S$ and $S \diamond N$ are still uv-segments. This operation can be extended to any pair g-magnet/uv-segment such that the result is a uv-segment.

We introduce a particular class of *ab*-segments, for fixed nodes a and b. For $g \geq 0$ and $k \geq 1$, $S_g[k]$ is recursively defined as follows. Let $S_0[k]$ be the path $[x, a, v_1, \ldots, v_{k-1}, b, y]$. Notice that $S = S_0[k]$ is an *ab*-segment with $R_{ab}[S] = [a, v_1, \ldots, v_{k-1}, b]$ and with two 0-magnets, $G_{ab}(S, a)$ and $G_{ab}(S, b)$, rooted at a and b, respectively.

Let $g \geq 1$ and let i be the largest j with $1 + 2^1 + \cdots + 2^j \leq k$; let $k' = k - (2^{i+1} - 1)$. If g is odd and $k' > 0$ then $S_g[k] = M_g \diamond S_{g-1}[1] \diamond S_{g-1}[2^1] \diamond \cdots \diamond S_{g-1}[2^i] \diamond S_{g-1}[k'] \diamond M_g$. For $k' = 0$, $S_g[k] = M_g \diamond S_{g-1}[1] \diamond S_{g-1}[2^1] \diamond \cdots \diamond S_{g-1}[2^i] \diamond M_g$. If g is even and $k' > 0$ then $S_g[k] = M_g \diamond S_{g-1}[k'] \diamond S_{g-1}[2^i] \diamond \cdots \diamond S_{g-1}[2^1] \diamond S_{g-1}[1] \diamond M_g$. For $k' = 0$, $S_g[k] = M_g \diamond S_{g-1}[2^i] \diamond \cdots \diamond S_{g-1}[2^1] \diamond S_{g-1}[1] \diamond M_g$. See the examples given in Fig. 1.

Notice that every graph $S = S_g[k]$ is an *ab*-segment. The *ab*-road $R_{ab}(S)$ is a path of length k. For every $v \in R_{ab}(S)$, either $G_{ab}(S, v) = \varnothing$ or there exists $h \in \{0, \ldots, g\}$ such that $G_{ab}(S, v) = N_0 \cup \cdots \cup N_h$, where N_i is an i-magnet with root v and $V(N_i) \cap V(N_j) = \{v\}$ whenever $i \neq j$. The graph S contains a simple path, connecting b to a, which passes through all the nodes of S apart from the tops of the magnets. Let $P_g[k]$ denote such a path. Intuitively, the adversary first forces the robot to traverse the path $P_g[k]$ and then the magnets force the robot to perform many trips along the *ab*-road. This idea yields the following lemma.

LEMMA 3.1. *For every $g \geq 0$ and $k \geq 1$, $\mathscr{P}_{\text{GREEDY}}(S_g[k], b) \geq (g + 1)k$.*

Proof. Let $S = S_g[k]$. Since the path $P_g[k]$ is simple, the adversary can lead the robot along this path from b to a. At the end of this trip, the only visited free nodes are the centers of the magnets. A magnet is *active* if its center is visited but its top is not. A magnet is *dead* if its top is visited. Otherwise a magnet is *asleep*. Thus, at the beginning all magnets are asleep and after the trip along $P_g[k]$, all magnets are active.

For $g = 0$, the robot moves to the top of the 0-magnet rooted at a, and then relocates to the top of the 0-magnet rooted at b. This involves at least k penalty traversals.

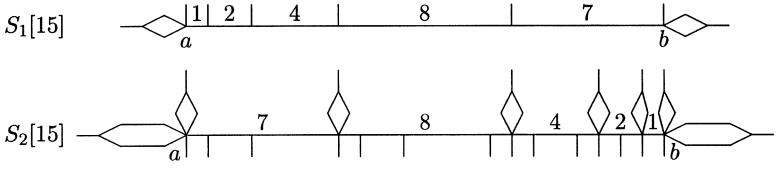

FIGURE 1

For $g = 1$, the robot moves to the top t_0 of the 0-magnet rooted at a and gets stuck. Let $R_{ab}(S) = [a, v_1, \ldots, v_{k-1}, b]$. Active 0-magnets are rooted at $v_1, v_3, \ldots, v_{2^j-1}, \ldots, b$, and active 1-magnets are rooted at a and b. The competitors for the nearest visited free node are the center x of the 1-magnet rooted at a and the node v_1. Since $d(t_0, x) = 3$ and $d(t_0, v_1) = 2$, the robot relocates to v_1, and then moves to the top t_1 of the 0-magnet rooted at v_1 and gets stuck. Since $d(t_1, x) = 4$ and $d(t_1, v_3) = 3$, the robot relocates to v_3. In general, when the robot arrives at v_{2^j-1}, it moves to the top t_{2^j-1} of the 0-magnet rooted at v_{2^j-1} and gets stuck. Since $d(t_{2^j-1}, x) = 1 + (2^j - 1) + 2 = 2^j + 2$ and $d(t_{2^j-1}, v_{2^{j+1}-1}) = 2^j + 1$, the robot relocates to $v_{2^{j+1}-1}$. It follows that the GREEDY strategy leads the robot to the node b, forcing it to perform at least k penalty traversals. Further, the robot visits the tops of the 0-magnet and 1-magnet rooted at b. At this stage, x becomes the unique visited free node. The robot relocates to this node performing at least k more penalty traversals. Therefore, $\mathscr{P}_{\text{GREEDY}}(S, b) \geq 2k$.

Suppose that $g \geq 2$. After traversing $P_g(k)$, the robot moves to the top of the 0-magnet rooted at a and gets stuck. Following the above reasoning, as long as there exist active 0-magnets, the robot relocates to the centers of these magnets and visits their tops. When the robot gets stuck at the top of the 0-magnet rooted at b, all the 0-magnets are dead. The nearest visited free node is now the center of the 1-magnet rooted at b. In general, when the robot gets stuck at the top of the h-magnet rooted at b (h even), all the h-magnets are dead and the nearest visited free node is the center of the $(h + 1)$-magnet rooted at b. The robot relocates to this node, and then moves to the top of the $(h + 1)$-magnet rooted at b and gets stuck. Recall that if a node on the ab-road is the root of an r-magnet then it is also the root of a q-magnet for each $q < r$. Therefore, whenever the robot gets stuck, the competitors for the nearest visited free node are the centers of $(h + 1)$- and $(h + 2)$-magnets. The chosen distance between two consecutive $(h + 1)$-magnets guarantees that the nearest visited free node is always the center of a $(h + 1)$-magnet. Consequently, the robot kills, one by one, all the $(h + 1)$-magnets. This involves a new trip along the ab-road of S, ending at a, and hence at least k more penalty traversals. When the robot gets stuck at the top of the h-magnet rooted at a (h odd), the nearest visited free node is the center of the $(h + 1)$-magnet rooted at a. The robot kills, one by one, all the $(h + 1)$-magnets. This involves a new trip along the ab-road, ending at b, and hence at least k more penalty traversals. We conclude that, in order to explor S, the robot performs $(g + 1)$ trips along the ab-road, after the initial one, and hence performs at least $(g + 1)k$ penalty traversals. ▮

THEOREM 3.1. *The penalty of algorithm* GREEDY *is not linear in the order of the graph.*

Proof. We construct a family of graphs $\{G_g : g \geq 1\}$ such that, for every $g \geq 1$, $\mathscr{P}_{\text{GREEDY}}(G_g) \geq g|V(G_g)|$.

Let $n_g(i)$ denote $|(S_g[2^i])| - (2^i + 1)$, i.e., the number of all nodes of $S_g[2^i]$ apart from those belonging to its *ab*-road. We prove by induction on $g \geq 1$ that

$$(\forall i \geq g + 1) n_g(i) \leq (i + 2)^{g+1}.$$

For every $i \geq 2, n_1(i) = 10 + i \leq (i + 2)^2$. Hence the inequality is true for $g = 1$. We suppose that it is true for g and prove it for $g + 1$. Suppose that $g + 1$ is odd (for $g + 1$ even, the argument is similar). By definition, $S_{g+1}[2^i] = M_{g+1} \diamond S_g[2^0] \diamond \ldots S_g[2^{i-1}] \diamond S_g[1] \diamond M_{g+1}$. It follows that

$$\begin{aligned} n_{g+1}(i) &\leq 8(g + 1) + n_g(0) + \cdots + n_g(i - 1) + n_g(0) \\ &\leq 8(g + 1) + (i + 1)n_g(i - 1). \end{aligned}$$

By induction hypothesis, for $i \geq g + 2, n_{g+1}(i) \leq 8(g + 1) + (i + 1)(i + 1)^{g+1} \leq (i + 2)^{g+1} + (i + 1)(i + 2)^{g+1} = (i + 2)^{g+2}$. Hence, the inequality is true for $g + 1$.

Consequently, for every $g \geq 1$ and $i \geq g + 1$, $|V(S_g[2^i])| \leq 2^i + 1 + (i + 2)^{g+1}$. By Lemma 3.1, $\mathscr{P}_{\text{GREEDY}}(S_g[2^i]) \geq (g + 1)2^i \geq (g + 1)|V(S_g[2^i])| - (g + 1)[1 + (i + 2)^{g+1}]$. Let $i_0 \geq g + 1$ be any integer for which $2^{i_0} \geq (g + 1)[1 + (i_0 + 2)^{g+1}]$. Let $G_g = S_g[2^{i_0}]$. It follows that $\mathscr{P}_{\text{GREEDY}}(G_g) \geq g|V(G_g)|$. ∎

Another natural way to explore a graph is given by the *depth-first search* strategy. The DFS exploration algorithm is defined as follows. Every time the current node is free, the robot takes an unexplored edge incident to the node. When stuck at a node, the robot relocates to the most recently visited free node, following a shortest path in the explored part of the graph.

The reason why DFS may be inefficient is, in some sense, opposite to that for the GREEDY strategy. The DFS exploration algorithm "blindly" looks at its history, always choosing the recently visited free node, even when it is far away from the current node. Hence the robot will postpone exploration of regions close to the current node, in order to relocate to the recently visited node. The difficulty in constructing graphs for which DFS performs poorly lies in the fact that adding devices similar to magnets cannot help. Such cycle-like structures with pending unexplored edges are a natural way to create free visited nodes (and hence to "drag" the robot

away) but graphs obtained in this way are necessarily sparse; in particular their number of edges is linear in their order. Since, as we know, the penalty of DFS does not exceed the number of edges, this construction method cannot show that the penalty of DFS is not linear in the order of the graph. To prove the inefficiency of DFS we need dense graphs.

For every positive even integer $x = 2k$, let $R[x] = (V(R[x]), E(R[x]))$, where, $V(R[x]) = \{a_0, \ldots, a_{x+1}, d_1, \ldots, d_k, e_1, \ldots, e_k\} \cup \{b_{i,j}, c_{i,j} : 0 \le i \le x, 1 \le j \le k\}$ and $E(R[x]) = \{[b_{i,j}, a_i], [b_{i,j}, a_{i+1}], [c_{i,j}, a_i], [c_{i,j}, a_{i+1}] : 0 \le i \le x, 1 \le j \le k\} \cup \{[b_{0,1}, d_1], \ldots, [b_{0,k}, d_k], [c_{x,1}, e_1], \ldots, [c_{x,k}, e_k]\}$. See Fig. 2 for the graph $R[4]$.

Let $C[x]$ denote the following cycle in $R[x]$: $[a_0, b_{0,1}, a_1, b_{1,1}, a_2, \ldots, a_x, b_{x,1}, a_{x+1}, c_{x,1}, a_x, \ldots, a_1, c_{0,1}, a_0, \ldots, a_0, b_{0,k}, a_1, \ldots, a_x, b_{x,k} a_{x+1}, c_{x,k}, a_x, \ldots, a_1, c_{0,k}, a_0]$. Notice that $C[x]$ passes exactly once through all edges of $R[x]$ apart from those of type $[b_{0,*}, d_*]$ and $[c_{x,*}, e_*]$.

Let G be a graph of order $x \ge 2$, $x = 2k$, with a_0 as one of its nodes. $R[G]$ denotes the graph defined by $V(R[G]) = V(R[x])$ and $E(R[G]) = E(R[x]) \cup E(G_1) \cup \cdots \cup E(G_{x-1})$, where, for every $i \in \{1, \ldots, x-1\}$, G_i is any graph with $V(G_i) = \{b_{i,1}, \ldots, b_{i,k}, c_{i,1}, \ldots, c_{i,k}\}$ for which there exists an isomorphism $f: G \to G_i$ such that $f(a_0) = c_{i,k}$. Intuitively, $R[G]$ is the graph $R[x]$ with isomorphic copies G_i of G inserted on sets of nodes $\{b_{i,1}, \ldots, b_{i,k}, c_{i,1}, \ldots, c_{i,k}\}$. Notice that $R[x] = R[N_x]$, where $N_x = (\{a_0, 1, 2, \ldots, x-1\}, \emptyset)$.

For every connected graph G with $a_0 \in V(G)$, let $p(G)$ denote $\mathscr{P}_{\mathrm{DFS}}(G, a_0)$. For convenience, let additionally $p(N_x) = 0$.

LEMMA 3.2. *Let $x = 2k \ge 2$. If G is N_x or any connected graph of order x, with $a_0 \in V(G)$, then*

1. $|V(R[G])| = x^2 + 3x + 2$;
2. $p(R[G]) \ge 2x^2 + (x-1)p(G)$.

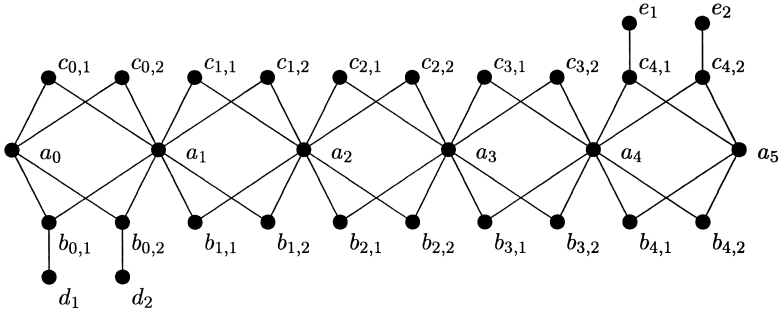

FIG. 2. The graph $R[4]$.

Proof. 1. Straightforward from the definition of $V(R[G])$.

2. Suppose that the robot starts at node a_0 and moves according to the DFS exploration algorithms. Since $R[x]$ is a subgraph of $R[G]$, $C[x]$ is also a cycle in $R[G]$. Since $C[x]$ is a simple cycle, there exists an adversary that, first, leads the robot along $C[x]$. At the end of this trip, the robot gets stuck at a_0.

If $G = N_x$, the most recently visited free node is $c_{x,k}$. By definition of DFS, the robot relocates to $c_{x,k}$. This requires $2x + 1$ penalty traversals.

If $G \neq N_x$ then $G_1, \ldots, G_{x-1}$ are connected components in the graph induced by the unexplored edges of $R[G]$. In this case, the most recently visited free node is $c_{1,k}$. Therefore, the robot relocates to $c_{1,k}$. This node belongs to $V(G_1)$. By definition of DFS, the robot does not leave the nodes of G_1 before exploring all of its edges. Moreover, when this happens, $c_{2,k}$ becomes the most recently visited free node. The robot relocates to this node, which belongs to $V(G_2)$. Iterating the above reasoning, it follows that the robot can be led to the node $c_{x,k}$ by saturating all nodes of $G_1, \ldots, G_{x-1}$. Notice that this can be done so that the robot performs at least $(x-1)p(G)$ penalty traversals. Supplementary $2x + 1$ penalty traversals are performed in order to reach nodes $c_{1,k}, \ldots, c_{x,k}$.

We conclude that there exists an adversary that leads the robot from a_0 to $c_{x,k}$ such that all edges, apart from those of the form $[b_{0,*}, d_*]$ and $[c_{x,*}, e_*]$, are explored and the number of penalty traversals is at least $2x + 1 + (x-1)p(G)$. At this stage of exploration, the free visited nodes (in the reverse order of visiting) are $c_{x,k}, b_{0,k}, c_{x,k-1}, \ldots, x_{x,1}, b_{0,1}$. Therefore, the robot has to relocate to these nodes, in the above order, and to explore the remaining edges. This requires $(2k-1)(2x+1) = 2x^2 - x - 1$ penalty traversals. Consequently, $p(R[G]) \geq 2x^2 + x + (x-1)p(G)$. ∎

THEOREM 3.2. *The penalty of algorithm DFS is not linear in the order of the graph.*

Proof. We construct a family of graphs $\{G_i : i \geq 1\}$ such that, for every $i \geq 1$, $\mathscr{P}_{\mathrm{DFS}}(G_i) \geq i|V(G_i)|$.

For every positive even integer x, let $H_1[x] = R[x]$ and $H_i[x] = R[H_{i-1}[x]]$, for $i \geq 2$. Notice that these graphs are well defined since, for every $i \geq 1$, $|V(H_i)|$ is even and $a_0 \in V(H_i)$.

For $k \geq 0$, let Π_k denote the set of all nonzero polynomials of degree at most k, with nonnegative integer coefficients. We prove by induction on $i \geq 1$ that there exists $P_i \in \Pi_{2^i-1}$ such that

$$\left|V(H_i[x])\right| = x^{2^i} + P_i(x) \qquad \text{and} \qquad p(H_i[x]) \geq 2ix^{2^i}.$$

By Lemma 3.2, $|V(H_1[x])| = x^2 + 3x + 2$ and $p(H_1[x]) \geq 2x^2$; hence the above property is true for $i = 1$. Supposing that the property is valid for $i \geq 1$, we prove its validity for $i + 1$.

Let $y = |V(H_i[x])|$; by induction hypothesis, there exists $P_i \in \Pi_{2^i-1}$ such that $y = x^{2^i} + P_i(x)$. By definition, $|V(H_{i+1}[x])| = |V(R[y])|$. By Lemma 3.2(1), $|V(H_{i+1}[x])| = y^2 + 3y + 2 = x^{2^{i+1}} + P_{i+1}(x)$, where $P_{i+1}(x) = 2x^{2^i}P_i(x) + P_i^2(x) + 3[x^{2^i} + P_i(x)] + 2$. Clearly, $P_{i+1} \in \Pi_{2^{i+1}-1}$. By Lemma 3.2(2), $p(H_{i+1}[x]) \geq 2y^2 + (y-1)p(H_i[x])$. By induction hypothesis, $p(H_i[x]) \geq 2ix^{2^i}$. It follows that $p(H_{i+1}[x]) \geq 2x^{2^{i+1}} + x^{2^i}(2i)x^{2^i} + [P_i(x) - 1]p(H_i[x]) \geq 2(i+1)x^{2^{i+1}}$. This ends the proof by induction.

Let $i \geq 1$. Since the degree of P_i is at most $2^i - 1$, there exists $x_i \geq 2$ such that $x_i^{2^i} \geq P_i(x_i)$. For $G_i = H_i[x_i]$ we have $|V(G_i)| \leq 2x_i^{2^i}$ and hence $\mathscr{P}_{\mathrm{DFS}}(G_i) \geq p(G_i) \geq i|V(G_i)|$. ∎

4. AN EXPLORATION ALGORITHM WITH LINEAR PENALTY

We present an algorithm which performs no more than $3|V(G)|$ penalty traversals for every connected graph G. The main idea is to maintain most of the robot relocations along a (dynamically constructed) spanning tree of G.

The algorithm EXPLORE(r), given below, controls the movements of the robot, where the node $r \in V(G)$ is its initial position. Two arrays are used in the algorithm description: $parent : V(G) \to V(G) \cup \{null, unassigned\}$ and $color : V(G) \to \{white, blue, red\}$. For every $v \in V(G)\backslash\{r\}$, $parent[v]$ will represent the *parent* of v in the spanning tree T of G that EXPLORE(r) will build. More exactly, $parent(v)$ is the neighbor of v on the path joining v with r in T.

```
EXPLORE(r): parent[r] := null;
            parent[u] := unassigned for every u ∈ V(G)\{r};
            color[u] := white for every u ∈ V(G);
            v := r;
            while v ≠ null do
                  SATURATE(v);
                  if v has a neighbor u such that parent[u] = unassigned
                  then
                     parent[u] := v;
                     v := u;
                  else
                     v := parent[v];
                  Move to v;
```

Procedure SATURATE(v) is the crucial part of the algorithm. We describe it later in detail. The aim of this procedure is to perform a travel which starts and ends at v and saturates v. As a side effect of SATURATE(v), some *white* nodes turn *red*.

The correctness of EXPLORE is straightforward. By the definition of SATURATE, every time the **if** instruction is reached, the robot knows all neighbors of v. Hence, all steps are well defined. Since every node is visited and saturated, the whole graph G is explored when the end of algorithm EXPLORE is reached.

During the execution of EXPLORE, we can distinguish two types of penalty traversals: those required by the procedure SATURATE and those dictated directly by EXPLORE in its last line. We call the traversals of the second type *external penalty traversals*. It is clear that the external penalty traversals are along the edges of a spanning tree T of G, in depth-first order. At this level of the description of EXPLORE, we can state the following complexity result that points out the role of the dynamically constructed tree T.

LEMMA 4.1. *For every connected graph G, $\mathscr{P}_{\text{EXPLORE}}(G) \leq 2|V(G)| + p$, where p is the number of penalty traversals performed during the executions of procedure* SATURATE.

In the case where the robot has to deal only with Eulerian graphs, our exploration strategy, based on depth-first search in an unknown tree, leads to a simple algorithm with linear penalty. Indeed, in this case, the procedure SATURATE(v) can simply tell the robot to take an unexplored edge as long as the current visited node is free. Since the graph is Eulerian, every node has an even degree and, therefore, the travel started at v ends also at v. Consequently, no penalty traversal is performed during any execution of SATURATE. Let EXPLORE′ denote the algorithm EXPLORE working only for Eulerian graphs, with procedure SATURATE defined as above. Clearly, for every Eulerian graph, $\mathscr{P}_{\text{EXPLORE}'}(G) \leq 2|V(G)|$.

Consider the general case. The travel, obtained by taking an unexplored edge as long as the currently visited node is free, does not necessarily end at the starting node. In order to bring the robot back to the starting node v, the procedure SATURATE(v) performs a sequence of *free trips* and maintains a *bridge* between the currently visited node and node v.

A free trip is a maximal sequence of consecutive unexplored edge traversals. During the execution of SATURATE, some *white* nodes are colored *blue*. By the end of SATURATE, every *blue* node will become either *white* or *red*. A *red* node will never change its color. A bridge is any path (not necessarily elementary) in the explored part of G, all of whose nodes are *blue*. The only penalty traversals are involved when the robot relocates

between two consecutive free trips. The number of these penalty traversals is kept small by using bridges.

Procedure SATURATE is defined as follows.

```
SATURATE(v): if v is saturated then return;
             color[v] := blue;
             u := v;
             while not (u = v and v saturated) do
               if u is incident to a free edge e then
                 u := the other end of e;
                 color[u] := blue;
               else
                 Let [u_0, u_1, ..., u_k] be a path from u to v where
                   all nodes are blue and all edges are explored;
                 color[u] := red;
                 u := u_1;
               Move to v;
             All blue nodes become white;
```

LEMMA 4.2. 1. *Procedure* SATURATE(v) *is well defined.*

2. *Procedure* SATURATE(v) *stops and, at its end, the node* v *is saturated and the robot is at* v.

3. *For every execution of* SATURATE, *the number of penalty traversals is equal to the number of blue nodes turned red.*

4. *A red node does not change its color.*

Proof. 1. It is sufficient to prove the following property, for every $i \geq 1$:

$P(i)$: *If* SATURATE(v) *performs the ith iteration of the while loop then, before this iteration, there is a bridge connecting u to* v.

Obviously, $P(1)$ is true. Suppose that $P(i)$ is true, for some $i \geq 1$, and suppose that SATURATE(v) performs the $(i+1)$th iteration of the while loop. Let u be the node where the robot is before the ith iteration. By induction hypothesis, there is a path B in the explored part of graph G, all its nodes being colored *blue*, which connects u to v. If u is not saturated, let u' be the node chosen by procedure SATURATE. Since this node will be colored *blue*, it follows that, before the $(i+1)$th iteration, the edge $[u', u]$ followed by the path B is a bridge connecting u' to v. If u is saturated, let $[u_0, u_1, \ldots, u_k]$ be the path defined by procedure SATURATE. Notice that such a path is the bridge B. Clearly, before the $(i+1)$th iteration, the path $[u_1, \ldots, u_k]$ is a bridge connecting u_1 to v. Hence, in both cases, $P(i+1)$ is true.

2. After each iteration of the while loop, either a new edge is explored or a new mode is colored *red*. Notice that outside procedure SATURATE nodes do not change their color. By definition, SATURATE colors a node *red* only once. Since the number of edges to be explored and the number of nodes to be colored *red* are finite, there is an iteration where vertex v is reached again and no unexplored edge is incident to v.

3. A maximal sequence of iterations, with u not saturated, defines a free trip for the robot. The only penalty traversal in SATURATE are involved when the robot moves from u to u_1. Each such move corresponds to a *red* coloring.

4. See (2). ∎

THEOREM 4.1. *The penalty of algorithm* EXPLORE *is linear in the order of the graph.*

Proof. We prove that, for every connected graph G, $\mathscr{P}_{\text{EXPLORE}}(G) \leq 3|V(G)|$. By Lemma 4.1, it is sufficient to prove that the number of penalty traversals, performed during all executions of SATURATE, is at most $|V(G)|$. This follows immediately from Lemma 4.2(3, 4). ∎

ACKNOWLEDGMENT

Thanks are due to the anonymous referee who suggested a simpler version of algorithm EXPLORE.

REFERENCES

1. S. Albers and M. R. Henzinger, Exploring unknown environments, *SIAM J. Comput.*, to appear.
2. B. Awerbuch, M. Betke, R. Rivest, and M. Singh, Piecemeal graph learning by a mobile robot, *in* "Proceedings 8th Conf. on Comput. Learning Theory, 1995," pp. 321–328.
3. E. Bar-Eli, P. Berman, A. Fiat, and R. Yan, On-line navigation in a room, *J. Algorithms* **17** (1994), 319–341.
4. P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen, and M. Saks, Randomized robot navigation algorithms, *in* "Proceedings 7th ACM-SIAM Symp. on Discrete Algorithms, 1996," pp. 74–84.
5. A. Blum, P. Raghavan, and B. Schieber, Navigating in unfamiliar geometric terrain, *SIAM J. Comput.* **26** (1997), 110–137.
6. M. Betke, R. Rivest, and M. Singh, Piecemeal learning of an unknown environment, *Mach. Learning* **18** (1995), 231–254.
7. X. Deng, T. Kameda, and C. H. Papadimitriou, How to learn an unknown environment. I. The rectilinear case, *J. ACM* **45** (1998), 215–245.
8. X. Deng and C. H. Papadimitriou, Exploring an unknown graph, *J. Graph Theory*, to appear.

9. F. Hoffmann, C. Icking, R. Klein, and R. Kriegel, A competitive strategy for learning a polygon, *in* "Proceedings, 8th ACM-SIAM Symp. on Discrete Algorithms, 1997," pp. 166–174.
10. P. Panaite and A. Pelc, Exploring unknown undirected graphs, *in* "Proceedings, 9th Ann. ACM-SIAM Symposium on Discrete Algorithms, 1998," pp. 316–322.
11. C. H. Papadimitriou and M. Yannakakis, Shortest paths without a map, *Theoret. Comput. Sci.* **84** (1991), 127–150.
12. N. S. V. Rao, S. Hareti, W. Shi, and S. S. Iyengar, "Robot Navigation in Unknown Terrains: Introductory Survey of Non-heuristic Algorithms," Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, July 1993.