Efficient exploration of faulty trees *

Euripides Markou † Andrzej Pelc ‡

Abstract

We consider the problem of exploration of trees, some of whose edges are faulty. A robot, situated in a starting node and unaware of the location of faults, has to explore the connected fault-free component of this node by visiting all its nodes. The *cost* of the exploration is the number of edge traversals. For a given tree and given starting node, the *overhead* of an exploration algorithm is the worst-case ratio (taken over all fault configurations) of its cost to the cost of an optimal algorithm which knows where faults are situated. An algorithm, for a given tree and given starting node, is called *perfectly competitive* if its overhead is the smallest among all exploration algorithms not knowing the location of faults. We design a perfectly competitive exploration algorithm for any line, and an exploration algorithm for any tree, whose overhead is at most 9/8 larger than that of a perfectly competitive algorithm. Both our algorithms are fairly natural and the total time of local computations used during exploration is linear in the size of the explored tree. Our main contribution is the analysis of performance of these algorithms, showing that natural exploration strategies perform well in faulty trees.

1 Introduction.

We consider exploration of faulty trees by a mobile agent, called *robot*. The robot is initially situated in a starting node v of a tree $T = (V, E)$. It has a faithful map of the tree with node and edge labels. Thus the robot knows which port at a node leads to which neighbor. Moreover, it knows the starting node. However, some of the edges of the tree are faulty. The robot does not know *a priori* the location of faults nor their number. When reaching a node for the first time, the robot discovers faulty edges incident to this node. A faulty edge prevents the robot from traversing it. Hence faults disconnect the tree and define a connected fault-free component C containing the starting node. The task of the robot is to *explore* C by visiting all its nodes (and hence traversing all its edges). Exploration is finished when the last node of C is visited. The set of faulty edges $F \subset E$ is called a *fault configuration*. For a given tree T, starting node v, and fault configuration F, the *cost* $\mathcal{C}(A, T, v, F)$ of an exploration algorithm A is the number of edge traversals it performs when exploring the fault-free component C containing v and corresponding to F. A robot that would know F, and hence would know the component C, has a simple exploration algorithm of minimum cost: perform a depth-first search traversal of the subtree C, which

*This work was done during Euripides Markou's stay at the Research Chair in Distributed Computing of the Université du Québec en Outaouais, as a postdoctoral fellow. Andrzej Pelc's research was supported in part by NSERC grant OGP 0008136 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

†Département d'Informatique, Université du Québec en Outaouais, Canada; evripidi@uqo.ca

‡Département d'Informatique, Université du Québec en Outaouais, Canada; pelc@uqo.ca

ends in the node farthest from v. The cost of this optimal exploration is $2n - h$, where n is the number of edges of C and h is the height of C. Call this optimal cost $opt(T, v, F)$.

Now consider an exploration algorithm A for a given tree T and starting node v, that *does not know* F, as supposed in our setting. A natural measure of performance of such an algorithm is the worst-case ratio between its cost and the optimal cost, where the worst case is taken over all fault configurations F. This number $\max_{F \subset E} \frac{\mathcal{C}(A,T,v,F)}{opt(T,v,F)}$ is called the *overhead* of A, and is denoted $\mathcal{O}_{A,T,v}$. This measure has a similar flavor to the competitive ratio of on-line algorithms. It measures the penalty incurred by the algorithm, due to some lack of knowledge. In the case of on-line algorithms, the knowledge concerns future events, which are known to an off-line algorithm (serving as a benchmark) but not to an on-line algorithm. In our case, knowledge concerns the fault configuration, known to an optimal algorithm (serving as a benchmark) but not to the exploration algorithms we want to design.

Given a tree T and a starting node v, an exploration algorithm (not knowing the fault configuration) is called *perfectly competitive*, if it has the smallest overhead among all exploration algorithms working under this scenario. Our aim is to construct exploration algorithms with small overhead: either perfectly competitive algorithms, or ones whose overhead closely approximates the smallest possible overhead.

The following remark will be useful for proving bounds on overhead of exploration algorithms. Suppose that the robot, at some point of the exploration, is at node w, then moves along an already explored edge e incident to w, and immediately returns to w. For any fault configuration, an algorithm causing such a pair of moves has cost strictly larger than the algorithm that skips these two moves. Hence, we restrict attention to exploration algorithms that never perform such returns. We call them *regular*.

1.1 Related work

Our work belongs to a large area of research on exploration and navigation problems for robots in an unknown environment, the unknown ingredient being the fault configuration in our case. Such problems have been extensively studied in the literature (cf. the survey [20]). There are two principal ways of modeling the explored environment. In one of them a geometric setting is assumed, e.g., unknown terrain with convex obstacles [7], or room with polygonal [9] or rectangular [3] obstacles. Another way is to represent the unknown environment as we do, i.e., as a graph, assuming that the robot may only move along its edges. The graph model can be further specified in two different ways. In [1, 4, 5, 11] 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, 16, 18, 19] the explored graph is undirected and the robot can traverse edges in both directions. The efficiency measure adopted in most papers dealing with exploration of graphs is the cost of completing this task, measured by the number of edge traversals by the robot. In some papers, additional restrictions on the moves of the robot are imposed. It is 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 [16]. Another direction of research concerns exploration of anonymous graphs. In this case it is impossible to explore arbitrary graphs and stop after exploration, if no marking of nodes is allowed. Hence the scenario adopted in [4, 5] is to allow *pebbles* which the robot can drop on nodes to recognize already visited ones, and then remove them and drop them in other places. The authors concentrate attention on the minimum number of pebbles allowing efficient exploration of arbitrary directed graphs. Exploring anonymous graphs without

the possibility of marking nodes (and thus possibly without stopping) is investigated, e.g., in [13, 17]. The authors concentrate attention not on the cost of exploration but on the minimum amount of memory sufficient to carry out this task. Exploration of anonymous graphs is also considered in [10, 14, 15].

A measure of performance of exploration algorithms, similar to the competitive ratio, and thus to our notion of overhead, has been used, e.g., in [12], where the authors study exploration of unknown graphs and graphs for which only an unoriented map is available. In their case, the benchmark was the performance of an algorithm having full knowledge of the graph. On the other hand, in [19], the authors studied the problem of fast broadcasting in faulty trees. As in our setting, tree edges were affected by faults with unknown locations but, instead of exploring, the goal was to broadcast information in the fault-free component of the source, as fast as possible, assuming that each informed node can send information only to one neighbor at a time. Again, overhead was used as measure of performance.

1.2 Our results

Our goal is to design exploration algorithms for trees, with small overhead. We have two main results.

- For any line (simple path), and any starting node, we design a perfectly competitive exploration algorithm. It turns out that such an algorithm makes at most two changes of direction during exploration. Its behavior and its overhead depend only on the distance between the starting node and the closer endpoint of the line.
- For an arbitrary tree and arbitrary starting node, we design an exploration algorithm whose overhead is at most 9/8 times larger than the overhead of a perfectly competitive algorithm. This can be considered as an approximation algorithm with ratio 9/8.

Both our algorithms are fairly natural and the total time of local computations used during exploration is linear in the size of the explored tree. Our main contribution is in the analysis of performance of these algorithms, showing that natural exploration strategies perform well in faulty trees.

2 Exploration of lines

In this section we construct a perfectly competitive exploration algorithm for lines. A line is a graph $L = (V, E)$, where $V = \{v_0, ..., v_n\}$ and $E = \{[v_i, v_{i+1}] : i = 0, 1, ..., n-1\}$. v_0 and v_n are called endpoints of the line. The starting node is denoted by v, a and b are distances between v and the endpoints of the line, with $a \leq b$. The endpoint at distance a from v is called the right endpoint and that at distance b – the left endpoint (in case of equality the choice is arbitrary but fixed). Hence the direction right of the starting node means that of the closer endpoint. We assume $b > 0$, otherwise the line consists of one node. Without loss of generality we may restrict attention to fault configurations in which there is at most one fault at each side of the starting node. Throughout the section, the line L and the starting node v are fixed, hence, for any exploration algorithm A and any fault configuration F, we write $\mathcal{C}(A, F)$ instead of $\mathcal{C}(A, L, v, F)$, $opt(F)$ instead of $opt(L, v, F)$, and $\mathcal{O}(A)$ instead of $\mathcal{O}_{A,L,v}$. Since L and v are known to the exploration algorithm, integers a and b can be considered as its input. Let r be the distance between the starting node and the fault in the right direction ($r = a$ if there is no fault to the right of v). Let l be the distance between

the starting node and the fault in the left direction ($l = b$ if there is no fault to the left of v).

In the description of the following procedure GO-LEFT-AND-RETURN we will use the elementary subroutine GO-FIRM in a direction (left or right) which means: go until a fault or an endpoint is met.

Procedure GO-LEFT-AND-RETURN

go left one step
if a fault is met **then**
  GO-FIRM right, STOP
else
  GO-FIRM right, GO-FIRM left, STOP
end if

The following exploration algorithm will be proved perfectly competitive. If there is no fault incident to the starting node (otherwise exploration is trivial) the behavior of the algorithm depends on the distance a between the starting node v and the closer endpoint. If $a \leq 3$, the algorithm performs a depth-first search exploration starting right of v. If $4 \leq a \leq 16$, the algorithm goes one step to the left, returns and performs a depth-first search exploration. Finally, if $a > 16$, the algorithm goes two steps to the left (if possible), returns and performs a depth-first search exploration. The formal description is given below.

Algorithm Line

if there is a fault incident to the starting node **then**
  GO-FIRM in the opposite direction and STOP
else
  if $a \leq 3$ **then**
    GO-FIRM right, GO-FIRM left, STOP
  end if
  if $4 \leq a \leq 16$ **then**
    GO-LEFT-AND-RETURN
  end if
  if $a > 16$ **then**
    go left one step
    if a fault is met **then**
      GO-FIRM right, STOP
    else
      GO-LEFT-AND-RETURN
    end if
  end if
end if

Remark. During the execution of Algorithm Line, local computation time used at each node is constant, hence the running time of the algorithm is linear in the size of the line.

Proposition 1 *The overhead of Algorithm Line is:*

- 1, *when* $a = 0$ *or* $a = 1$

- $\frac{2a+1}{a+2}$, *when* $2 \leq a \leq 3$
- $\frac{2a+4}{a+4}$, *when* $4 \leq a \leq 16$
- $\frac{9}{5}$, *when* $17 \leq a \leq 19$
- $\frac{2a+7}{a+6}$, *when* $a \geq 20$

Proof: Consider a fault configuration F. Denote Algorithm Line by A. If $r = 0$ or $l = 0$ then $\mathcal{C}(A, F) = opt(F)$. Otherwise, consider the following cases:

$1 \leq a \leq 3$. We have $\mathcal{C}(A, F) = 2r + l$ and $opt(F) = \min\{2r + l, 2l + r\}$. If $r \leq l$ then $opt(F) = 2r + l = \mathcal{C}(A, F)$. If $r > l$ then $\frac{\mathcal{C}(A,F)}{opt(F)} = \frac{2r+l}{2l+r}$ which is maximized for $l = 1$ and $r = a$. Thus $\mathcal{O}(A) = \frac{2a+1}{a+2}$ in this case. In particular, $\mathcal{O}(A) = 1$ for $a = 1$.

$4 \leq a \leq 16$. If $l = 1$ then $\mathcal{C}(A, F) = 2 + r = opt(F)$. If $l > 1$ and $r \leq l$ then $\frac{\mathcal{C}(A,F)}{opt(F)} = \frac{2+2r+l}{2r+l}$, which is maximized for $l = 2$, $r = 1$. So $\frac{\mathcal{C}(A,F)}{opt(F)} \leq \frac{3}{2}$. If $l > 1$ and $r > l$ then $\frac{\mathcal{C}(A,F)}{opt(F)} = \frac{2+2r+l}{2l+r}$, which is maximized for $l = 2$, $r = a$. So $\frac{\mathcal{C}(A,F)}{opt(F)} \leq \frac{2a+4}{a+4}$. Since $\frac{2a+4}{a+4} \geq \frac{3}{2}$, when $a \geq 4$, it follows that $\mathcal{O}(A) = \frac{2a+4}{a+4}$ in this case.

$17 \leq a \leq 19$. If $l = 1$ then $\mathcal{C}(A, F) = 2+r = opt(F)$. If $l = 2$ then $\frac{\mathcal{C}(A,F)}{opt(F)} = \frac{4+r}{\min\{4+r,2r+2\}}$ which is maximized for $r = 1$. So $\frac{\mathcal{C}(A,F)}{opt(F)} \leq \frac{5}{4}$. If $l > 2$ and $r \leq l$ then $\frac{\mathcal{C}(A,F)}{opt(F)} = \frac{4+2r+l}{2r+l}$, which is maximized for $r = 1$, $l = 3$. So $\frac{\mathcal{C}(A,F)}{opt(F)} \leq \frac{9}{5}$. If $l > 2$ and $r > l$ then $\frac{\mathcal{C}(A,F)}{opt(F)} = \frac{4+2r+l}{2l+r}$, which is maximized for $l = 3$, $r = a$. So $\frac{\mathcal{C}(A,F)}{opt(F)} \leq \frac{2a+7}{a+6}$. Since $\frac{9}{5} \geq \frac{2a+7}{a+6}$, when $a \leq 19$ and $\frac{9}{5} > \frac{5}{4}$, it follows that $\mathcal{O}(A) = \frac{9}{5}$ in this case.

$a \geq 20$. In this case Algorithm Line behaves as before but now $\frac{2a+7}{a+6} \geq \frac{9}{5}$. Thus $\mathcal{O}(A) = \frac{2a+7}{a+6}$ in this case. □

We now show that Algorithm Line has the smallest overhead among all exploration algorithms for the line, not knowing the fault configuration. We denote by $\mathcal{A}_k$ the class of exploration algorithms for the line which do initially k returns, assuming that no fault or endpoint is encountered before the first k returns, then GO-FIRM, return and GO-FIRM. Any regular exploration algorithm for the line is in one of the classes $\mathcal{A}_k$. An algorithm of class $\mathcal{A}_1$ is called an i-step algorithm, if it goes i steps before the first return, unless a fault or an endpoint is encountered earlier. Notice that algorithm Line is an $\mathcal{A}_0$ class algorithm when $a \leq 3$, an 1-step $\mathcal{A}_1$ class algorithm when $4 \leq a \leq 16$ or a 2-step $\mathcal{A}_1$ class algorithm when $a \geq 17$.

The following lemmas establish bounds on the overhead of algorithms of classes $\mathcal{A}_0$ and $\mathcal{A}_1$. Their proofs are omitted due to lack of space.

Lemma 1 *Any exploration algorithm A of class $\mathcal{A}_0$ has overhead at least $\frac{2a+1}{a+2}$, when $a \geq 2$.*

Lemma 2 *Assume $a \geq 2$. Any x-step algorithm A, of class $\mathcal{A}_1$, which starts exploration by going left has overhead:*

- $\frac{2b+1}{b+2}$, *when* $b \leq x$
- $\max\{\frac{3x+3}{x+3}, \frac{3x+2a+1}{2x+a+2}\}$, *when* $b > x$

Lemma 3 *Any 1-step algorithm A of class $\mathcal{A}_1$ has overhead at least:*

- $\frac{3}{2}$, *when* $2 \leq a \leq 4$
- $\frac{2a+4}{a+4}$, *when* $a > 4$

Lemma 4 *Any 2-step algorithm A of class $\mathcal{A}_1$ has overhead at least:*

- $\frac{5}{4}$, *when* $a = 2$
- $\frac{9}{5}$, *when* $3 \leq a \leq 19$
- $\frac{2a+7}{a+6}$, *when* $a \geq 20$

The following lemma implies that a perfectly competitive algorithm for the line must be either in class $\mathcal{A}_0$ or in class $\mathcal{A}_1$. The proof of the lemma is omitted due to lack of space.

Lemma 5 *Assume $a \geq 2$. For every exploration algorithm A of class $\mathcal{A}_k$ with $k \geq 2$, there exists an algorithm of class $\mathcal{A}_0$ or an i-step algorithm of class $\mathcal{A}_1$, for $i \leq 2$, with smaller or equal overhead.*

Proposition 2 *The overhead of any exploration algorithm for the line is at least:*

- $\frac{2a+1}{a+2}$, *when* $2 \leq a \leq 3$
- $\frac{2a+4}{a+4}$, *when* $4 \leq a \leq 16$
- $\frac{9}{5}$, *when* $17 \leq a \leq 19$
- $\frac{2a+7}{a+6}$, *when* $a \geq 20$

Proof: In view of Lemma 5, we may restrict our considerations to algorithms of class $\mathcal{A}_0$ and i-step algorithms of class $\mathcal{A}_1$, for $i \leq 2$.

Assume $2 \leq a \leq 3$. Any algorithm of class $\mathcal{A}_0$ has overhead at least $\frac{2a+1}{a+2}$ (Lemma 1). Any 1-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{3}{2}$ (Lemma 3). Any 2-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{5}{4}$, when $a = 2$ and at least $\frac{9}{5}$, when $a = 3$ (Lemma 4). Since $\frac{2a+1}{a+2} \leq \frac{5}{4} < \frac{3}{2}$ when $a = 2$, and $\frac{2a+1}{a+2} \leq \frac{3}{2} < \frac{9}{5}$ when $a = 3$, the overhead of any exploration algorithm for the line is at least $\frac{2a+1}{a+2}$.

Assume $4 \leq a \leq 16$. Any algorithm of class $\mathcal{A}_0$ has overhead at least $\frac{2a+1}{a+2}$ (Lemma 1). Any 1-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{2a+4}{a+4}$ (Lemma 3). Any 2-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{9}{5}$ (Lemma 4). Since $\frac{2a+4}{a+4} \leq \frac{2a+1}{a+2}$ and $\frac{2a+4}{a+4} \leq \frac{9}{5}$, the overhead of any exploration algorithm for the line is at least $\frac{2a+4}{a+4}$.

Assume $17 \leq a \leq 19$. Any algorithm of class $\mathcal{A}_0$ has overhead at least $\frac{2a+1}{a+2}$ (Lemma 1). Any 1-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{2a+4}{a+4}$ (Lemma 3). Any 2-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{9}{5}$ (Lemma 4). Since $\frac{9}{5} \leq \frac{2a+1}{a+2}$ and $\frac{9}{5} \leq \frac{2a+4}{a+4}$, the overhead of any exploration algorithm for the line is at least $\frac{9}{5}$.

Assume $a \geq 20$. Any algorithm of class $\mathcal{A}_0$ has overhead at least $\frac{2a+1}{a+2}$ (Lemma 1). Any 1-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{2a+4}{a+4}$ (Lemma 3). Any 2-step algorithm of class $\mathcal{A}_1$ has overhead at least $\frac{2a+7}{a+6}$ (Lemma 4). Since $\frac{2a+7}{a+6} \leq \frac{2a+1}{a+2}$ and $\frac{2a+7}{a+6} \leq \frac{2a+4}{a+4}$, the overhead of any exploration algorithm for the line is at least $\frac{2a+7}{a+6}$. □

Propositions 1 and 2 imply

Theorem 1 *Algorithm Line is a perfectly competitive exploration algorithm for any line.*

3 Exploration of trees

In this section we construct a natural exploration algorithm for an arbitrary tree and arbitrary starting node, and we prove that the ratio between the overhead of this algorithm and the overhead of a perfectly competitive algorithm is at most $\frac{9}{8}$.

The idea of our Algorithm Tree is the following. The algorithm works in phases. The first phase starts at the root v. At the beginning of each phase the robot, having reached a node u for the first time, decides which of the accessible children of u is a root of the deepest subtree. Then the robot explores all subtrees rooted at other accessible children of u, and moves to this last accessible child, thus ending the current phase. The robot stops when there are no accessible children of a node reached at the end of a phase.

In a fault-free tree, this algorithm is clearly optimal, as the robot finishes exploration in the leaf farthest from the starting node. However, due to the existence of faults, the decisions of the robot may be suboptimal: a subtree seeming to be the deepest at some point of the exploration, and thus left to be explored at the end, may turn out later to be quite shallow, due to faults unknown at the decision time. Nevertheless we will show that this exploration strategy is worse than the perfectly competitive one by a factor of at most 9/8.

Let $T = (V, E)$ be the input tree and v the starting node. We consider T as being rooted at v. Let F be a fixed fault configuration. For any $F' \subset F$, we denote by $T[F']$ the connected fault-free component of T containing v and corresponding to the fault configuration F'. Let u be any node of T. Denote by T_u the subtree of T rooted at u, and by $F(u)$ the set of edges in F incident to u. Node u is called *free*, if u is not a leaf in $T[F(u)]$ (i.e. at least one edge going down from u is fault free). Assume that u is free and let $w_1, ..., w_k$ be all children of u in $T[F(u)]$. The *principal child* of u, denoted $pc(u)$, is the child w_i for which the tree T_{w_i} has height greater than or equal to the height of any tree T_{w_j}, $1 \leq j \leq k$, with ties broken arbitrarily. Let $B(F)$ denote the branch of $T[F]$, such that the child of any node u on this branch is $pc(u)$. We call $B(F)$ the *principal* branch.

Let T' be any subtree of the tree T_w, rooted at w. In the description of our algorithm we will use the procedure EXPLORE(T'), which consists in any fixed depth-first-search traversal of T', starting and ending at w.

When the robot arrives at u for the first time, it learns $F(u)$ and hence it can find out if u is free, and if so, it can find $pc(u)$. The following procedure CLEAR(u), called when the robot reaches u for the first time, explores subtrees rooted at all children of u except $pc(u)$ and then proceeds to $pc(u)$.

Procedure CLEAR(u)
for all children w of u in $T[F]$, such that $w \neq pc(u)$ **do**
    EXPLORE($T_w[F]$)
go to $pc(u)$
$current := pc(u)$

Algorithm Tree
$current := v$
while *current* is free **do**
    CLEAR(*current*).

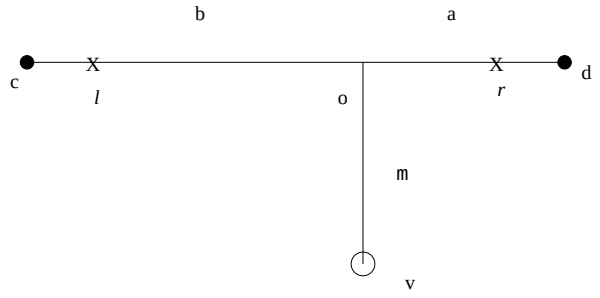

Figure 1: A t-shaped tree (faulty edges marked with X)

Remark. Total local computation time used by Algorithm Tree is linear in the size of the tree.

In order to prove our result, we first compute the overhead of Algorithm Tree, and then establish a lower bound on the overhead of any other exploration algorithm, showing that the ratio between them does not exceed 9/8. Fix a tree T and a starting node v. A subtree T_1 of T is called *t-shaped* (see Figure 1), if it consists of a (possibly empty) simple path between v and a node o, with another (possibly empty) simple path attached to the node o (an empty path is defined as one consisting of a single point). The segment between v and o is called the *stem* of T_1 and the attached simple path is called the *bar* of T_1 .

For any node $w \in V$, let m_w denote the distance between v and w, and let h_w denote the height of the second deepest among all subtrees T_u, where u is a child of w. Let $a_w = h_w + 1$.

Proposition 3 *The overhead of Algorithm Tree, for the tree T and starting node v, is 1 if T is a line with endpoint v, and it is*

$$max_{w \in V} \frac{m_w + 2a_w + 1}{m_w + a_w + 2},$$

otherwise.

Proof: Fix a tree T and a starting node v. Denote Algorithm Tree by A. If T is a line with endpoint v, the proposition clearly holds. Assume that T is not a line with endpoint v. We write $\mathcal{C}(A, F)$ instead of $\mathcal{C}(A, T, v, F)$, $opt(F)$ instead of $opt(T, v, F)$, and $\mathcal{O}(A)$ instead of $\mathcal{O}_{A,T,v}$. Without loss of generality, we may consider only fault configurations that have at most one fault on any branch (more than one fault on a branch does not alter the fault-free component of v). We say that a fault configuration F is *dominated* by a fault configuration F', if $\frac{\mathcal{C}(A,F)}{opt(F)} \leq \frac{\mathcal{C}(A,F')}{opt(F')}$. A fault configuration F is called *special*, if $T[F]$ is a t-shaped tree.

Fix a fault configuration F. Let L be one of the longest branches of $T[F]$. An optimal algorithm traverses all edges twice, except those on L, which it traverses once. On the other hand, algorithm A traverses all edges twice, except those on $B(F)$, which it traverses once. Hence, all edges in $T[F]$ outside of the t-shaped tree T_1 which is the union of L and $B(F)$, are traversed twice by both algorithms. If all such edges are deleted, the ratio between the cost of A and the cost of the optimal algorithm increases. Consider the special configuration $F' \supset F$, for which $T[F'] = T_1$. We have $B(F') = B(F)$ and L is still the longest branch in $T[F']$. Hence, in $T[F']$ (as in $T[F]$), an optimal algorithm traverses all edges twice, except those on L, which it traverses once, and algorithm A traverses all edges twice, except those on $B(F)$, which it traverses once. This implies that F' dominates F.

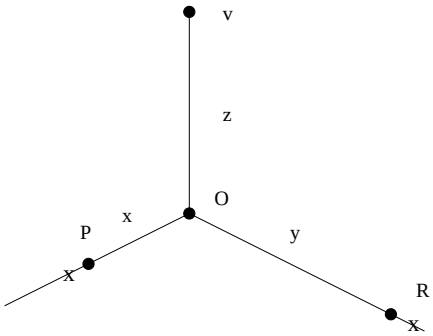

Figure 2: The tree $T[F']$ (faulty edges marked with X)

It follows that $\mathcal{O}(A) = \frac{\mathcal{C}(A,F')}{opt(F')}$, for some special configuration F' obtained as above. If $T[F']$ is a line with endpoint v then $\frac{\mathcal{C}(A,F')}{opt(F')} = 1$, so we can restrict attention to fault configurations for which this is not the case. For such a fault configuration F', let O be the common node of $B(F')$ and L, farthest from v. Let P be the leaf ending $B(F')$ and R the leaf ending L (see Figure 2).

Denote by z the distance between v and O, by x the distance between O and P, and by y the distance between O and R. We have $\frac{\mathcal{C}(A,F')}{opt(F')} = \frac{z+2y+x}{z+2x+y}$. For a given node O (and hence for a given z), this fraction is maximized when y is the largest possible and x is the smallest possible. Since $T[F']$ is not a line with endpoint v, the integer x must be positive, and hence the smallest possible value of x is 1. This corresponds to $P = pc(O)$. The largest possible value of y corresponds to the leaf R not on $B(F')$ and farthest from O. Hence R must be the deapest leaf in T_u, where u is the child of O for which T_u is second deepest.

Hence $\mathcal{O}(A) = \frac{z+2y+1}{z+y+2}$, for some node O, where z and y have the above defined meaning. Notice that $z = m_O$ and $y = a_O$. This concludes the proof of the lemma. □

If T is a line with endpoint v then Algorithm Tree is optimal. Hence from now on we assume that T is not a line with endpoint v. Let o denote the node $w \in V$ for which the maximum from Proposition 3 is taken (if there are many such nodes, o denotes any of them). Let c denote the leaf in T_o farthest from o, and d the second farthest leaf (in the case of a tie, take any farthest, resp. second farthest, leaf). Denote by m the distance between v and o, by a the distance between o and d, and by b the distance between o and c. Thus $a \leq b$. By definition and in view of Proposition 3, the overhead of Algorithm Tree, for the tree T and starting node v, is $\frac{m+2a+1}{m+a+2}$.

Consider the t-shaped subtree T_1 of T whose leaves are v, c and d. We call c the left endpoint of the bar of T_1 and d its right endpoint. A robot starting at v and continuing exploration after node o by going towards c (d) is said to go left (resp. right) on the bar. Since T_1 is a subtree of T, the overhead of a perfectly competitive algorithm for the tree T and starting node v is not smaller than the overhead of a perfectly competitive algorithm for the tree T_1 and starting node v. Hence, in order to establish a lower bound on the first, it is enough to show that it holds for the second. This is the approach we will adopt. Similarly as for the line, we may restrict attention to fault configurations in which there is at most one fault at each side of the point o on the bar. Also, in our lower bound arguments, we consider only fault configurations without faults on the stem of T_1, as such a fault yields ratio $\mathcal{C}(A, T_1, v, F)/opt(T_1, v, F) = 1$. Consider a fault configuration F. Let r

be the distance between the node o and the fault on the bar in the right direction ($r = a$ if there is no fault to the right of o). Let l be the distance between the node o and the fault on the bar in the left direction ($l = b$ if there is no fault to the left of o). See Figure 1.

Until the end of the section, the tree T_1 and the starting node v are fixed, hence, for any exploration algorithm A and any fault configuration F, we write $\mathcal{C}(A, F)$ instead of $\mathcal{C}(A, T_1, v, F)$, $opt(F)$ instead of $opt(T_1, v, F)$, and $\mathcal{O}(A)$ instead of $\mathcal{O}_{A,T_1,v}$.

Similarly as for the line, when the robot reaches node o, GO-FIRM in a direction (left or right) on the bar means: go until a fault or an endpoint is met. We denote by $\mathcal{A}_k$ the class of exploration algorithms for the tree T_1 which, after reaching node o do initially k returns on the bar, assuming that no fault or endpoint is encountered before the first k returns, then GO-FIRM, return and GO-FIRM. Any regular exploration algorithm for the tree T_1 and the starting node v, is in one of the classes $\mathcal{A}_k$. An algorithm of class $\mathcal{A}_1$ is called an i-step algorithm, if it goes i steps on the bar before the first return, unless a fault or an endpoint is encountered earlier.

The proofs of the following lemmas 6, 7, 8 are similar to the proofs of lemmas 1, 2, 5.

Lemma 6 *Any exploration algorithm A of class $\mathcal{A}_0$ has overhead at least $\frac{m+2a+1}{m+a+2}$, when $a \geq 2$.*

Lemma 7 *Assume $a \geq 2$. Any x-step algorithm A, of class $\mathcal{A}_1$, which after first reaching node o, goes left on the bar, has overhead at most:*

- $\frac{m+2b+1}{m+b+2}$, *when* $b \leq x$
- $\max\{\frac{m+3x+3}{m+x+3}, \frac{m+3x+2a+1}{m+2x+a+2}\}$, *when* $x < b$

Lemma 8 *Assume $a \geq 2$. For every exploration algorithm A of class $\mathcal{A}_k$ with $k \geq 2$, there exists an algorithm of class $\mathcal{A}_0$ or an i-step algorithm of class $\mathcal{A}_1$, for $i \leq m + 2$, with smaller or equal overhead.*

The proofs of the following lemmas are omitted due to lack of space.

Lemma 9 *Assume $2 \leq a \leq m + 4$, $m > 0$. Algorithm Tree has overhead less or equal to the overhead of any exploration algorithm A of class $\mathcal{A}_1$.*

Lemma 10 *Assume $a \geq m + 5$, $m > 0$. Any x-step algorithm A, $0 < x \leq m + 2$, of class $\mathcal{A}_1$, has overhead at least $\frac{4m+2a+7}{3m+a+6}$.*

Theorem 2 *For any tree and any starting node, the ratio between the overhead of Algorithm Tree and the overhead of a perfectly competitive algorithm is at most $\frac{9}{8}$.*

Proof: Consider any tree T with starting node v and the corresponding t-shaped tree T_1, previously defined. As mentioned before, lower bounds on the overhead of a perfectly competitive algorithm, proved in the previous lemmas for the tree T_1, carry over to the tree T. Hence we can argue as follows.

If $a \leq 1$ then Algorithm Tree is perfectly competitive. Hence we may assume $a \geq 2$.

- When $m > 0$ and $a \leq m + 4$, Lemmas 6, 8, 9 imply that Algorithm Tree is perfectly competitive.

- When $m > 0$ and $a \geq m + 5$, Lemmas 6, 8, 10 imply that the ratio between the overhead of Algorithm Tree and that of a perfectly competitive algorithm is at most: $\frac{\frac{m+2a+1}{m+a+2}}{\frac{4m+2a+7}{3m+a+6}} \leq \frac{9}{8}$.

- When $m = 0$, the tree T_1 is a line and $v = o$. Hence Proposition 1 and Theorem 1 imply that the ratio between the overhead of Algorithm Tree and that of a perfectly competitive algorithm is at most:

  1, when $a \leq 3$; $\frac{\frac{2a+1}{a+2}}{\frac{2a+4}{a+4}} \leq \frac{9}{8}$, when $4 \leq a \leq 16$; $\frac{\frac{2a+1}{a+2}}{\frac{9}{5}} \leq \frac{9}{8}$, when $17 \leq a \leq 19$; and $\frac{\frac{2a+1}{a+2}}{\frac{2a+7}{a+6}} \leq \frac{9}{8}$, when $a \geq 20$. This concludes the proof.

□

4 Conclusion.

We designed efficient exploration algorithms for faulty trees. For the line we constructed a perfectly competitive algorithm, and for an arbitrary tree we proposed an algorithm whose approximation ratio with respect to a perfectly competitive algorithm is 9/8. Both our algorithms use linear computation time. It remains open if there exists a perfectly competitive algorithm for arbitrary trees, with polynomial computation time. An even more challenging problem is to construct a perfectly competitive exploration algorithm for arbitrary graphs, or – if such polynomial-time algorithms do not exist in general – good approximations thereof.

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 A. Mirzaian, Competitive robot mapping with homogeneous markers, IEEE Transactions on Robotics and Automation 12 (1996), 532-542.

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

[12] A. Dessmark and A. Pelc, Optimal graph exploration without good maps, Proc. 10th European Symposium on Algorithms (ESA 2002), LNCS 2461, 374-386.

[13] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc, Tree exploration with little memory, Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'02), 588-597.

[14] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, Robotic exploration as graph construction, IEEE Transactions on Robotics and Automation 7 (1991), 859-865.

[15] V. Dujmović and Sue Whitesides, On Validating Planar Worlds, Proc. of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2001), Washington DC, U.S.A, January 2001, 791-792.

[16] 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.

[17] P. Fraigniaud and D. Ilcinkas. Directed graphs exploration with little memory, Proc. 21st Symposium on Theoretical Aspects of Computer Science (STACS'04), to appear.

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

[19] P. Panaite, A. Pelc, Optimal broadcasting in faulty trees, Journal of Parallel and Distributed Computing 60 (2000), 566-584.

[20] 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.