Searching with Mobile Agents in Networks with Liars [*]

Nicolas Hanusse[§] Evangelos Kranakis[*‡] Danny Krizanc[†‡]

October 28, 2002

Abstract

We present deterministic algorithms to search for an item s contained in a node of a network, without prior knowledge of its exact location. Each node of the network has a database that will answer queries of the form "how do I get to s?" by responding with the first edge on a shortest path to the node containing s. It may happen that some nodes, called *liars*, give bad advice. If the number of liars k is bounded, we show different strategies to find the item depending on the topology of the network. In particular we consider the complete graph, ring, torus, hypercube and bounded degree trees.

1 Introduction

The current information explosion on the Internet makes appealing the idea of having a "personal explorer" chasing down information on the web. These personal explorers can be thought of as mobile programs that traverse the network and have the ability to focus their efforts and perform certain predetermined tasks. They have already found numerous applications, like reminding us of appointments, periodically dialing phone numbers, or even pointing out spelling errors.

Mobile agents can perform very complex information gathering, like assembling and digesting "related" topics of interest. Depending on their "behavior" mobile agents can be classified as reactive (responding to changes in their environment) or pro-active (seeking to fulfill certain goals). Moreover agents may choose to remain stationary (filtering incoming information) or become mobile (searching for specific information across the Internet and retrieving it) [17]. There are numerous examples of such agents in use today, including the Internet search engines, like Yahoo, Lycos, etc.

In this paper we consider the problem of searching for an item in a distributed network in the presence of "liars." The objective is to design a mobile agent that travels along the network links in order to locate the item. Although the location of the item in the network is initially unknown, information about its whereabouts can be obtained by querying the nodes of the network. The nodes have databases providing the first edge on a shortest path to the item sought. The agent

[§]LaBRI-CNRS, Universite Bordeaux I, 33405 Talence, France, Email: hanusse@labri.u-bordeaux.fr. Research supported by CNRS-Jemstic "Mobicoop"

[*]School of Computer Science, Carleton University, Canada. Email: kranakis@scs.carleton.ca. Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada)

[†]Department of Mathematics, Wesleyan University, Middletown, Connecticut 06459, USA, Email: dkrizanc@caucus.cs.wesleyan.edu

[‡]Research supported in part by MITACS (Mathematics of Information Technology and Complex Systems) grants.

[*]A preliminary version of this paper has appeared in [9]

queries the nodes; the queried nodes respond either by providing a link adjacent to them that is on a shortest path to the node that holds the item or if the desired item is at the node itself then the node answers by providing it to the agent. However certain nodes in the network may be liars, e.g., due to out-of-date network information in their databases. The liars are unknown to the mobile agent that must still find the item despite the fact that responses to queries may be wrong. In this paper we give deterministic algorithms for searching in a distributed network with a bounded number of liars that has the topology of a complete network, ring, torus, hypercube, or trees under three models of liars.

A variant of the above *searching* model, was introduced in [10], where the network topologies considered were the *ring* and the *torus* and the nodes respond to queries with a bounded probability of being incorrect. Additional investigations under the same model of "searching with uncertainty" were carried out for fully interconnected networks in [11]. Models with faulty information in the nodes have been considered before for the problem of routing (see [1, 3, 7, 8, 15]). However, in this problem it is assumed that the identity of the node that contains the information is known, and what is required is to reach this node following the best possible route. Search problems in graphs, where the identity of the node that contains the information sought is not known, have been considered before. These include *deterministic search games*, where a fugitive that possesses some properties hides in the nodes or edges of a graph [5, 13, 14]), and the problem of exploring an *unknown* graph [2, 12, 16]. Our model is similar in spirit to the model in [4] where the authors propose algorithms to search for a point on a line or on a lattice drawn on the plane. However, in that model, the nodes have limited if any knowledge of where the point lies and they do not provide new location information at each step as in our case.

1.1 Preliminaries and Definitions

In order to present the problem more precisely, we must define the search model in a given network.

The mobile agent is basically a software program running an algorithm that requires a certain amount of memory, storing relevant information about its current position in the network, e.g. in a binary tree the distance to the root, in a ring the distance from the starting node, etc. We assume that the agent knows the size, n, and topology of the network it is on and that it has an upper bound k on the number of liars it may encounter. We do not assume it has a bound on the actual distance d from its starting point to the item it is searching for but we are generally interested in the case where k is small with respect to d. The complexity measures we are interested in are the number of steps used by the agent as a function of d and k (and possiblely n) and the amount of memory required by the algorithm. We will see later that the algorithm depends on the topology of the network and we will consider different trade-offs between the amount of memory required by the mobile agent and the number of steps, i.e the number of moves of the mobile agent.

A network of n nodes is represented as a connected undirected graph $G = (V, E)$ where V is the set of vertices or nodes and E the set of edges or links. Let s denote the item the mobile agent is searching for and assume there is a unique node in G containing s. A *query* $Q_u(s)$ returns either s if the node u contains s or a subset of edges, incident to u, belonging to a shortest path leading to the item s. If $Q_u(s)$ returns an edge that does not belong to a shortest path to s, the node u is called a *liar*, otherwise a *truthteller*. The path $p = u_0 u_1 \cdots u_\alpha$ is a sequence of nodes followed by the mobile agent until item s is found. The number of edges followed by the path p is called the *number of steps* of the mobile agent, which is denoted by α. If there are no liars we expect that the mobile agent will follow an optimal path, i.e. if $k = 0$, it is obvious that $\alpha = d$ where d is the distance between the starting node of the mobile agent and the node containing the item. By convention, we assume that the nodes are labelled by the set $\{1, 2, \ldots, n\}$.

1.2 Response Models

We consider three models of responses to queries:

One advice per node with co-ordination (CO) Model: In this case, a query returns a unique edge. We assume some preprocessing was done when building the databases stored in each node u of V. Let v be the node containing s and choose a fixed shortest path tree with destination v. For a given node u, $Q_u(s) = e$ where e is the (unique) outgoing edge incident to u chosen in this shortest path tree. If a node indicates an edge on another shortest path this node is considered to be a liar. The mobile agent is assumed to have knowledge as to how the shortest path trees were originally constructed. For example, we assume they always report first a row and then a column in the case of the torus. The truthtellers are *co-ordinated* in that the set of edges they report leads to the construction of a particular shortest path spanning tree. An adversary may decide which nodes are liars but has no influence over which edges are to be reported by the truthtellers.

One advice per node without co-ordination (NCO) Model: In this model an adversary decides which nodes are liars and also decides which correct edge the truthtellers will report whenever there is a choice among shortest path edges.

One advice per edge (ECO) Model: In this model a truthteller returns $Q_u(s)$ equal to the set of all incident edges to u belonging to a shortest path tree. A liar may return any (presumably non-empty) subset of the edges incident to u. Again the adversary has no input as to what is returned by a truthteller.

In fact, all three models are the same if we consider a network in which each shortest path is unique. In this case, ECO only reports a single edge and there is no point in a liar reporting more than one edge since you would immediately know the node was lying.

1.3 Results

In this paper, we consider searching for an item under the above models and for different topologies: complete graph, ring, torus, hypercube, trees. In each case, we assume that the mobile agent knows the topology of the network and suspects a bounded number k of liars. We assume that the responses of the nodes are set before the start of the algorithm according to the model considered and that they do not change throughout the running of the algorithm. The cost measures we consider for a given algorithm are the number of steps (i.e, edges traversed) and the amount of memory required by the mobile agent. Our results are presented in Table 1.

Although we defined three models, we can remark that any algorithm in the NCO model will work in the CO Model. In the same way, if you have an NCO algorithm then it will work in the ECO model. The mobile just chooses an arbitrary edge in the set reported in the ECO Model and runs the NCO algorithm. For the algorithms we present the ECO and CO algorithms behave the same. (Whether this holds in general is an open question as well as any proof that the models are in fact distinct.) For these reasons, in the table any line that has NCO (resp. CO) does not need CO and ECO (resp. ECO) to be listed. All lower bounds, except the reference of [11] for the complete graph, are independent of the amount of memory the algorithm uses. Nevertheless, the results indicated in [11] imply that no fixed memory deterministic algorithm can locate an item in

Topology	Lower bound	Upper bound	Memory (in bits)	Model
	$k+1$	$k+1$	$k \log n$	NCO
Complete		$2k+3$	$\log k$	NCO
	∞	∞	$O(1)$	[11]
Ring	$d+\min\{2d, 2k\}$	$d+4k+2$	$O(\log k)$	NCO
		$O(d)$	$O(\log d)$	NCO
	$\max\{\min\{d^2, k+1\}, d+\min\{2d, k\}\}$	$O(d^2)$	$O(\log d)$	CO, NCO
		$d+O(k)$	$O(k \log k)$	CO
Torus		$O(d\sqrt{k})$	$O(\log k)$	NCO
		$d+k\log k$, for $d > \sqrt{k}\log k$	$O(\log k)$	NCO
Hypercube	$\max\{\min\{\sum_{i=1}^{d} \binom{\log n}{i}, k+1\}, d+\min\{2d, k\}\}$	$O(dk)$	$O(\log n \log k)$	CO
		$d+O(k)$	$O(\log n \log k)$	ECO
Tree (δ, Δ)	$\Omega(d+(\delta-1)^{\min\{k,d\}})$	$d+O((\Delta-1)^{2k})$	$O(k \log(\Delta-1))$	CO

Table 1: Upper and lower bounds of the number of steps

a finite number of steps. We establish two general lower bounds that hold for networks satisfying certain conditions. Lower bounds for the complete graph, torus and hypercube are derived from these results.

The case of the torus is particularly interesting since it illustrates the possibility of running simultaneously different mobile agents: for small d and large k, $O(d\sqrt{k})$ is better than $d+O(k \log k)$. In the ECO model, the databases stored in each node require more memory than for the CO and the NCO Model. However, the ECO Model appears to be more efficient at least in the case of the hypercube. We are also interested in finding trade-offs between the memory required by the mobile agent and the number of steps it uses. The study of the complete graph is a perfect example to show, for the same model, we have two algorithms with a comparative number of steps but an exponential ratio in the amount of memory required.

2 Lower Bounds

In this section we establish two general theorems that will be useful in deriving lower bounds for the time required by deterministic agents on the particular graphs we consider below. Both of the bounds hold in all three models of responses to queries.

Theorem 1 *Let G be a graph with $k \leqslant 2d$ liars. Assume there exist three nodes, u, v and w such that the distance between u and v and between u and w is d and the distance between v and w is $2d$. Then there exists a distribution of liars on G such that any deterministic agent requires at least $d+k$ steps to find an item at distance d starting at u.*

Proof. Form a shortest path tree emanating from u to all nodes of G. Set the advice of all nodes to point towards u. Given any deterministic agent, run it for k steps. It is easy to see that either the agent visited neither of v and w and it is at least d steps away from one of them or the agent visited one of v or w and it is at least d steps from the other. In either case, we can place the item at one of v or w and correct the advice of all nodes not visited to obtain a distribution of liars that requires at least $d+k$ steps. ■

Theorem 2 *Let G be a graph with k liars. Let D be the number of nodes within distance d of some node u of G. Then there exists a distribution of liars on G such that any deterministic agent requires at least* $\min\{D, k+1\}$ *steps to find an item at distance at most d starting at u.*

Proof. Form a shortest path tree emanating from u to all nodes of G. Set the advice of all nodes to point towards u. Given any deterministic agent, run it for $\min\{D-1, k\}$ steps. At this point there exists at least one node at distance at most d from u that the agent has not visited. Place that item at such a node and correct the advice of all nodes not visited to obtain a distribution of liars that requires at least $\min\{D, k+1\}$ steps. ■

3 Complete Graph

From theorem 2 it is easy to see that for $k < n$ the number of steps required by an agent to search on the complete graph is at least $k+1$ (in all models). In this section, we present two algorithms for agents. The first one uses the optimal number of steps but a significant amount of memory for large values of k. The second one reduces the amount of memory required at a slight cost in time. We note that all shortest paths are of length one so that all response models are equivalent.

Algorithm SEARCHCOMPLETE(S) works as follow: starting from a node u , we follow its advice to node u' unless we have already visited u' in which case we select any node not previously visited and go there.

Theorem 3 *In any complete graph of n vertices with k liars, a mobile agent can find an item in at most $k+1$ steps with $k \log n$ bits of memory .*

Proof. Using the algorithm SEARCHCOMPLETE, the item s is found in a single step as long as the mobile agent is in a truthteller node. Since the network has at most k liars, a path p can be of length at most $k+1$, the worst being when all the k first nodes are liars. The mobile agent stores the labeling of each node visited (at most k) and each node belongs to the set $\{1, \ldots, n\}$. In the worst case, the amount of memory will be $k \log n$. ■

We are interested in a trade-off between the memory and the number of steps required by a mobile agent to find an item. Algorithm SEARCHCOMPLETEII illustrates this idea: follow the advice of nodes labeled $1, 2, \ldots, k+1$ until you find the item, i.e. if node labeled i gives bad advice and sends you to a node not containing the item then go to node labeled $i+1$. More formally, the algorithm is as follows:

SEARCHCOMPLETEII(S):

1. let $u = u_0$ and $v = 1$;
2. if $s \in u$, mobile agent stops;
3. go to node v;

4. if $s \in v$, mobile agent stops;
5. mobile agent queries v: $u = Q_v(s)$;
6. mobile agent stores $v = v + 1$ in its memory;
7. go to node u and continue in step 2.

For this algorithm we have

Theorem 4 *In any complete graph of n vertices and k liars, a mobile agent can find an item in at most $2k + 3$ steps with $\log k$ bits of memory.*

Proof. Using SearchCompleteII, one step is required to go to node 1 and then the path is of length at most $2k + 2$. Indeed, in the worst case, nodes $1, 2, \ldots, k$ are liars and their advice is to go to a node $u < k + 1$. Note that $\log k$ bits are required to maintain a counter. ■

4 Ring

For the ring, each vertex is of degree two and we may consider a global orientation known by each processor. Each node has a left and a right edge labelled respectively $\leftarrow$ and $\rightarrow$, i.e. the query $Q_u(s)$ returns $\leftarrow$ or $\rightarrow$.

Clearly, ignoring the advice of all nodes and moving in an arbitrary direction, the item can be found in at most $n - d$ steps. Another strategy consists in reaching at round i the node at distance 2^i on the left and then the node at distance 2^{i+1} on the right, for $i = 0, 2, 4, \ldots, \log d$. In total, $O(d)$ steps are sufficient. For small values of k, a tighter analysis is possible as we see below.

The following lemma is straightforward:

Lemma 1 *Let $B = u_1 u_2 \ldots u_l$ be a set of l consecutive nodes in a ring along the direction $\rightarrow$. If the number of $\rightarrow$ (resp. $\leftarrow$) advices is more than $k + 1$, then s is located on the right (resp. left) side of u_1 (resp. u_l).*

Algorithm SearchRing(s, k) is an implementation of Lemma 1.

SearchRing(s, k)

1. Let *dir* be the direction indicated by the starting node
2. **repeat**
3. move in direction *dir*
4. **until** an advice a is given more than k times or s is found.
5. move in direction a until s is found.

For this algorithm we can prove

Theorem 5 *In a ring of n vertices with k liars, a mobile agent can find an item in at most $d+4k+2$ steps with $O(\log k)$ bits of memory.*

Proof. Suppose the first node u_0 is a truthteller. In this case, the number of steps will be d, since the mobile agent keeps going in a unique direction. If u_0 is a liar, we start to go in the "wrong direction". By using Lemma 1, we know that we change direction as soon as we find $k+1$ truthtellers. Since the number of liars is bounded by k, a set of $2k+1$ consecutive nodes is always sufficient to know the best direction. Then, the mobile agent moves in the opposite direction and takes $2k+1$ steps to go back to the initial node. In total $\alpha = 2(2k+1)+d$. A counter of size $O(\log k)$ bits is sufficient to run the algorithm. ■

From theorem 1 it is easy to derive a lower bound of $d+\min\{2d, k\}$ for the problem of searching on a ring with k liars (assuming $n > 4d$). A slight improvement to theorem 1 is possible for the special case of a ring.

Theorem 6 *Assume $n > 4d$. There exists a distribution of k liars in the ring of n vertices for which the number of steps required by a deterministic agent to search for an item at distance d is at least $d+\min\{2d, 2k\}$.*

Proof. Let u, v and w satisfy the requirements of theorem 1. Given any agent starting at u, run it for $\min\{2d, 2k\}$ steps. At this point, the agent is either closer to v and has never visited w or closer to w and never visited v. It is easy to see that in either case we can place the item at one of the nodes in such a way that the agent is required to perform at least d more steps in its search. We further note that the agent has encountered at most k liars (only those on the same side of u as we have placed the item).

■

5 Torus

Let T_{n_1,n_2} be a torus corresponding to the cartesian product of $C_{n_1} \times C_{n_2}$ with $Card(V) = n = n_1 n_2$. For sufficiently large n, using Theorem 1 and Theorem 2 it is straight forward to derive a lower bound of $\max\{\min\{d^2, k+1\}, d+\min\{2d, k\}\}$ steps for any deterministic agent searching for an item at most d away from the origin on a torus with at most k liars. Further, we observe that by ignoring the databases of the nodes and searching in a spiral pattern around the start node, an item can always be found in $O(d^2)$ steps. For small values of k with respect to d and for different response models, better upper bounds are possible.

We present three algorithms to find the item on a torus of n nodes. We assume there exists a global orientation of the edges known by each node and its four incident edges are labelled L, R, U, D for the left, right, up and down direction. We also use the notation $\leftarrow, \rightarrow, \uparrow, \downarrow$ for the edges. In the discussion below, u represents the current location of the agent. If dir is a direction, $\overline{dir}$ indicates the opposite direction: $\leftarrow = \overline{\rightarrow}$ and $\uparrow = \overline{\downarrow}$. The set of *advices of a block* (or of a rectangle) consists of the set of directions $\{a_{\leftarrow}, a_{\rightarrow}, a_{\downarrow}, a_{\uparrow}\}$ so that each a_{dir} corresponds to the number of responses in the direction dir. The *advice a_{dir} of a block B* is the direction indicated by the majority of B.

5.1 CO Model

For this model, we assume truthtellers always report a row direction when possible and only report a column direction when necessary. The algorithm SEARCHRINGII(s, dir, l) travels in a set, called *block*, of l consecutive nodes along the direction dir and returns the number of query responses for each direction $\leftarrow, \rightarrow, \uparrow, \downarrow$.

SEARCHRINGII(s, dir, l):

1. let $a_{\leftarrow}, a_{\rightarrow}, a_{\uparrow}, a_{\downarrow} = 0$;
2. for $i = 1$ to l, move along the direction dir. Each time an advice is dir, $a_{dir} = a_{dir} + 1$;
3. return $\{a_{\leftarrow}, a_{\rightarrow}, a_{\uparrow}, a_{\downarrow}\}$.

SEARCHTORUS(s, k)

1. **Decide an horizontal direction:**
   (a) $dir = Q_u(s)$, $a =$ SEARCHRINGII$(s, dir, 2k+1)$;
   (b) if $a_{\overline{dir}} > k$ then $dir = \overline{dir}$ else if $a_{dir} \leqslant k$ then go to step 3.
2. **Search approximately the column of s along an horizontal direction:** Move in the direction indicated by the majority of nodes in step 1. For each successive block B of $2k+1$ nodes, compute the majority of horizontal responses with SEARCHRINGII and move along the direction dir until two adjacent blocks, say B and B', are found with different majority.
3. **Search the column of s:**
   (a) let $R = \{c_1, c_2, \ldots, c_{4k+2}\}$ be the set of columns of the two last blocks B and B' (if we come from Step 1, $R = \{c_1, c_2, \ldots, c_{2k+1}\}$ and $B' = \{\}$), and u an arbitrary node of $B \cup B'$;
   (b) move following the horizontal advice of the current node until we find a node $u' \in c_i$ with another advice or until u' is on the boundary of R;
   (c) store i;
   (d) if i has been stored $k+1$ times, then $c = i$ else go to the adjacent node in the upward direction, repeat from step b.
4. **Search the row of s:** do SEARCHRING(s, k') in the column c where k' is the number of remaining edges.

Steps 1 and 2 of SEARCHTORUS(s, k) use the two following lemmas:

Lemma 2 *Let $B = u_1u_2 \ldots u_{2k+1}$ and $B' = u_{2k+2}u_{2k+2} \ldots u_{4k+2}$ be two sets of $4k+2$ consecutive nodes in a ring. If the advice of B is horizontal and the advice of B' is different then the column of s intersects a node u_i belonging to B or B'.*

Lemma 3 *Let B be a block of $2k+1$ consecutive nodes along an horizontal (resp. vertical) direction. Let m be the advice of B. If B has more than $k+1$ nodes indicating a direction different from m, then the column (resp. row) of s intersects a node of B.*

Theorem 7 *In any torus of n vertices and k liars, a mobile agent of $O(k \log k)$ bits of memory can find an item in at most $d + O(k)$ steps.*

Proof. To choose an horizontal direction, the number of extra steps is bounded by $4k + 2$. The next extra steps are counted in Step 3. If we consider a set of $2k + 1$ consecutive rows, at least $k + 1$ rows contain only truthtellers. Let us suppose column j contains s. In a row of truthtellers, the sequence of advices is $a_1 \dots a_{j-1} a_j a_{j+1} \dots a_{4k+2}$ with $a_i = \rightarrow$ (resp. $\leftarrow$) if $i < j$ (resp. $i > j$) and $a_j = \uparrow$ or $\downarrow$. If we move along the advice of a_i, step b stops at a_j, corresponding to the column of s. If the mobile agent is in a row containing a liar, an adversary can only shift the mobile agent once per liar in the wrong horizontal direction. In total, the mobile agent can be shifted "badly" k times. Then, a row of truthteller requires at most k steps to move the mobile agent in the correct column. If we start from c_1, s may be located on the opposite boundary of R, c_{4k+2}.

Let us consider the vertical extra steps: the number of rows is at most $2k+1$ and it may happen that s is below the first row. This gives $2(2k + 1) = 4k + 2$ extra steps.

In total, for step 3, the number of extra steps is $2k + 2(4k + 2) = 10k + 4$. When we add the number of extra steps of Step 1 and Step 4, we obtain at most $10k + 4 + 2(4k + 2) = 18k + 8$.

All variables are of size $O(k)$ and so, are coded with $O(\log k)$ bits. Step 3 requires $4k + 2$ variables of size $O(\log k)$ bits. In total, the mobile agent uses $O(k \log k)$ bits of memory. ∎

Remark. No attempt is made to optimize constants. We may consider a strategy by using a counter of the maximum liars remaining in the network. In this case, later blocks will be of a smaller size. However, the lower bound $d + k$ steps indicates that another strategy would not improve significantly our upper bound.

5.2 NCO and ECO Models

In the NCO Model, the walk of SearchTorus in Step 3 does not work. Indeed, a row of truthtellers may indicate different columns for the item. We propose a new strategy to choose a starting direction in SearchTorusII. We make a search within a square of area $O(k)$ instead of a segment of $O(k)$ nodes along a given direction. We propose a variant of an algorithm which can be found in [10] to choose a starting direction in a square:

SearchSquare(s, u, l): (a) For each direction dir, $a_{dir} = 0$; (b) the mobile agent searches for the desired item s by testing all nodes in a square B of area l centered at node u; for each node of advice dir, $a_{dir} = a_{dir} + 1$; (c) return $\{a_{\leftarrow}, a_{\rightarrow}, a_{\uparrow}, a_{\downarrow}\}$;

The following lemma is a stronger version of Lemma 1:

Lemma 4 *Let $B = u_1 u_2 \dots u_l$ be a set of l nodes such that u_1 is the leftmost and u_i the rightmost node. If the number of $\rightarrow$ (resp. $\leftarrow$) advices is more than $k+1$ then s is located on the right (resp. left) side of u_1 (resp. u_i).*

A similar lemma is valid for the vertical direction.

The idea of SearchTorusII is the following: (1) we first locate s in a band of columns (or rows) $c_1, \dots, c_w$ of width $w = O(\sqrt{k})$ finding squares B, B' of area $4k + 1$ with different horizontal (or vertical) advice, (2) we find the vertical (or horizontal) direction to follow by a walk in a rectangle R of size $O(\sqrt{k}) * (2k+1)$ containing B, B' (3) we search for s in the direction given by the majority of the nodes of R in consecutive squares of width $O(\sqrt{k})$.

SEARCHTORUSII(s):

1. **Move along an horizontal and/or vertical direction:**

   Let us iterate a=SEARCHSQUARE($s, u, 4k+1$) by moving u, following the advice of a, to the boundary of the square. Continue iterations until two squares not necessarily consecutive, say B and B', such that B and B' yield different recommendations for the horizontal or vertical direction are found. If the advice of B and B' are respectively (a_1, a_2) and $(\overline{a_1}, \overline{a_2})$ with $a_1 \in \{\rightarrow, \leftarrow\}$ and $a_2 \in \{\uparrow, \downarrow\}$, go to Step 4.

2. **Deciding a vertical (horizontal) direction:** Let c_1 and c_w be two columns (resp. rows) representing the horizontal (resp. vertical) boundary of the location of the item s obtained in Step 1. Let l be the current row (resp. column). Let R be a rectangle of $2k+1$ rows (resp. columns) $l_1, l_2, \ldots, l_{2k+1}$ centered in $l_{k+1} = l$ bounded by c_1 and c_w. Search for the item in rectangle R by tracing a spiral around l, i.e following rows

   $$l_{k+1}, l_{k+2}, l_k, \ldots, l_{k+1+i}, l_{k+1-i}, \ldots, l_{2k+1}, l_1.$$

   Let *dir* be the vertical (resp. horizontal) direction given by the majority of nodes. Go to the column (resp. row) $c = (c_1 + c_w)/2$ and move in the direction *dir* until the boundary of R is reached.

3. **Search for s along a vertical (horizontal) direction:** Visit between the columns (resp. rows) c_1 and c_w the nodes of the current row (resp. column). If the item has not been found, go to the next row (resp. column) in the direction *dir* and repeat from Step 3.

4. **Search for s in a square:** Search for s in the square of area $16k + 4$ containing B and B'.

Using Algorithm SEARCHTORUSII, we have

Theorem 8 *In any torus of n vertices and k liars, a mobile agent can find an item in at most $O(d\sqrt{k})$ steps with $O(\log k)$ bits of memory.*

Proof. Step 1 and each iteration of Step 3 takes $O(k)$ steps and moves the mobile agent $\Omega(\sqrt{k})$ closer to the item. It may happen that we find s in Step 2 but the walk in the rectangle R is a spiral to obtain the same result. In Step 2, $O(k^{3/2})$ nodes are visited but the mobile agent goes $\Omega(k)$ closer to the destination. ■

If $d = \Omega(\sqrt{k}\log k)$, another strategy illustrated by SEARCHTORUSIII may be interesting: (1) As for Steps 1 and 2 of SearchTorus, we first locate s in a band of columns (or rows) $c_1, \ldots, c_w$ with $w \leqslant 8k+2$ finding two blocks B, B' of size $4k+1$ with different horizontal or vertical advice, (2) The next steps consist of applying a variant of the dichotomy principle in rectangles of size $O(k) \times O(k)$ to find the column of s.

More precisely, SEARCHTORUSIII(s, k) works as follow:

1. **Move along an horizontal and/or vertical direction:** (a) Let $dir = Q_u(s)$; (b) Iterate SEARCHRINGII($s, dir, 4k+1$) (with *dir* becoming the advice of the current block) until two adjacent blocks, say B and B', such that B and B' yield different recommendations for the horizontal or vertical direction, are found.

2. **Search for s in a set of rows (columns):** Assume the direction indicated in Step 1 is horizontal. A similar step will be done otherwise.

   - (a) Let $p, p+1, \ldots, q$ be the set of consecutive columns where p (resp. q) is the leftmost (resp. rightmost) column of B and B', let $m = \lceil (p+q)/2 \rceil$, go to node u belonging to column c_m and assign to dir a vertical direction (ex: $dir = \uparrow$);
   - (b) Let R be a rectangle, above node u (below if $dir = \downarrow$), of $3k+1$ consecutive rows starting from B and delimited by columns p and q
   - (c) Let $B_m = R \cap c_m$ be a set of nodes;
   - (d) In B_m, for each direction dir, compute the number of advices a_{dir}:
     - $\underline{\text{case } a_{\leftarrow} > k}$ (resp. $a_{\rightarrow} > k$), the column of s is between p and m (resp. m and q) : $q = m$ (resp. $p = m$), $m = \lceil (p+q)/2 \rceil$, if $p = q$ do SEARCHRING(s, k) in column c_m else move u to c_m, repeat from Step c;
     - $\underline{\text{case } a_{\uparrow} \text{ and } a_{\downarrow} > k}$: the row of s belongs to R, go to Step 3
     - $\underline{\text{case } a_t > k \text{ with } t \in \{\uparrow, \downarrow\}}$: $dir = t$, move u along the direction dir for $3k+1$ steps.
   - (e) repeat from Substep b until two adjacent rectangles R and R' of different vertical advices are found.

3. **Search for s in a rectangle:** The item s is located within the rectangle R and eventually R' of Step 2; (a) let u be the center of the rectangle (b) let dir be the advice of SEARCH-SQUARE$(s, u, 4k+1)$; (c) move u in the middle of its previous location and the boundary of the rectangle; (d) repeat from Substep b until the token is found.

SEARCHTORUSIII leads to the following result:

Theorem 9 *In any torus of n vertices and k liars, a mobile agent of $O(\log k)$ bits of memory can find an item in at most $O(d + k \log k)$ steps.*

Proof. Each iteration of SearchRingII takes the mobile agent $\Omega(k)$ steps closer to the destination. During Step 2, either the column of s is found in $O(\log k)$ search in blocks of $3k+1$ nodes or the approximate location of the token is computed in a rectangle of size $O(k) \times O(k)$. In the first case, using SEARCHRING, the token is found with $O(k)$ extra steps. In the second case, each time we do a vertical move of $3k+1$ steps, we go $\Theta(k)$ steps closer to the destination. In the last step, we only use the dichotomy principle. An examination of $4k+1$ nodes gives the vertical or horizontal move to follow. This step will be iterated at most $O(\log k)$ times since we know the boundary of the token location. ■

Figure 1 gives an illustration of SEARCHTORUSIII. Blocks 1,2,...,7 of $4k+1$ nodes corresponds to the iteration of Step 1. Then, the vertical advice of Block 7 leads to the iteration of Step 2 until Block 10. The location of the token is approximately known and the iteration of Step 3 in Squares of area $4k+1$ find the token in Square 14.

5.3 ECO Model

In the ECO Model, the mobile agent may use the same algorithms as the CO Model. Indeed, the mobile agent can do the co-ordination itself choosing one edge per node. Since we have a lower

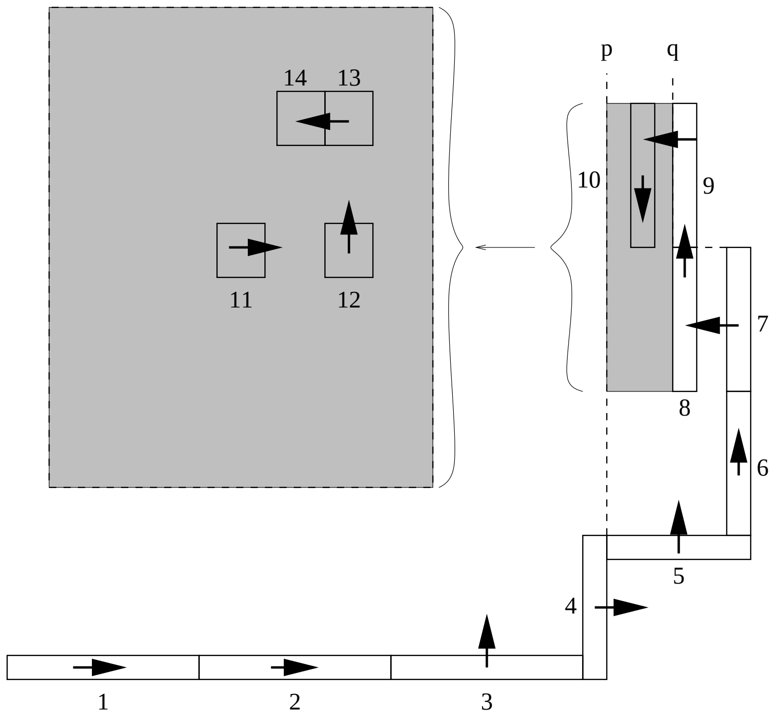

Figure 1: Example of an execution of SEARCHTORUSIII

bound of $d+\Omega(k)$ steps for any model, the upper bounds in ECO Model does not change a lot if we do not pay particular attention to the constants.

6 Hypercube

In this section, we show that the ECO model has an advantage over the CO model. We present one algorithm per model. We show that a mobile agent can find a token in at most $d+O(k \log n)$ steps in the CO model and in at most $d+O(k)$ steps in the ECO model.

Let C_l be an hypercube of $n=2^l$ vertices. Each node u is coded by $(x_l,\ldots,x_1)$ with $x_i \in \{0,1\}$. We assume there exists a global orientation of edges known by each node such that two nodes are adjacent along the direction i, labelled $\rightarrow_i$, if they agree in all but the position i. For $l \geqslant 2$, C_l is hamiltonian (see [6]) and it follows that any subgraph of C_l isomorphic to $C_{l'}$ with $l' < l$ is hamiltonian.

If we do not follow the advice of nodes, a naive algorithm to find an item at a distance d is to visit all nodes at progressively larger distance from the starting node. In the worst case this strategy will require at least $D=\sum_{i=1}^{d}\binom{\log n}{i}$ steps since there exist D nodes of at most distance d of a given node. Using theorems 1 and 2 one gets a lower bound (assuming $l \geqslant 2d$) of $\max\{\min\{D,k+1\},d+\min\{2d,k\}\}$ for any deterministic agent searching for an item at most d away from the origin on a hypercube with at most k liars.

6.1 CO Model

In this model, the co-ordination works in the following way : each node always reports first the direction 1, then direction 2, ..., direction l. In other words, if the advice of a node $u = (x_l, \ldots, x_1)$ is $\rightarrow_i$, it indicates that the destination v should have at least the $i-1$ last coordinates $x_{i-1} \ldots x_1$ identical.

Let us consider the starting node $u = (x_l, \ldots, x_1)$ and the node $v = (y_l, \ldots, y_1)$ is the node containing s. The idea of the algorithm to find the coordinates of v is the following :

SEARCHHYPERCUBE(s, k)

1. let $i = 0$;
2. **Select** $2k+1$ **nodes to visit**: we choose a subgraph $Q' = C_{\lceil \log 2k+1 \rceil}$ of C_l such that all nodes of Q' have same coordinates $x_{l'}, \ldots, , x_{i+1} y_i, \ldots, y_1$;
3. **Compute the number of advice for any direction**: we follow an hamiltonian path in Q' and compute, for the first $2k+1$ nodes, the number a_j of responses $\rightarrow_j$ of Q';
4. **Determine the coordinate to change to go closer**: if $a_i > k$ then $y_i = 1 - x_i$ and $\forall j < i, y_j = x_j$;
5. i:=i+1;
6. repeat from Step 2 until the item is found.

We obtain immediately

Theorem 10 *In an hypercube C_l of $n = 2^l$ nodes with k liars, a mobile agent using $O(\log n \log k)$ bits of memory can find an item in at most $d(2k+1)$ steps.*

6.2 ECO Model

In the ECO Model, a node u gives a response $Q_u = (a_{l-1}, \ldots, a_0)$. The position of s is given using the majority among $2k+1$ responses for each co-ordinate.

Lemma 5 *Let v be the node containing s. Let U be a set of $2k+1$ arbitrary nodes. If we query all nodes of U then $v = (\lfloor \sum_{u \in U} a_{l-1}/k \rfloor, \ldots, \lfloor \sum_{u \in U} a_0/k \rfloor)$.*

An easy upper bound of $d + 4k + 2$ steps can be obtained by following $2k+1$ nodes in a hamiltonian path in C_l. This result can be improved if we consider only a hamiltonian path in a subgraph of C_l isomorphic to $C_{\lceil \log (2k+1) \rceil}$.

Theorem 11 *In a hypercube C_l of $n = 2^l$ vertices with k liars, a mobile agent can find an item in at most $d + 2k + 1 + \lceil \log (2k+1) \rceil$ steps with $O(\log n \log k)$ bits of memory.*

7 Trees

For trees, the shortest path between two nodes is unique and so all three response models are equivalent. We present one algorithm for bounded degree trees (δ, Δ) where the degree of each internal node is bounded from below by δ and from above by Δ.

The naive algorithm of using breadth-first search yields an upper bound of at least $(\delta - 1)^d$ steps since at least that many nodes may have to be searched in the worst-case. Theorems 1 and 2 suggest a lower bound of $\max\{\min\{(\delta-1)^d, k+1\}, d+\min\{2d, k\}\}$ assuming the tree has diameter at least $2d$.

By using the advice of the nodes, some improvement over breadth-first search is possible for small k, but we shall see that an exponential penalty is necessary. We suppose that we are starting from a node u_1, considered as a root of the tree. Node u_1 gives an orientation of the edges. Each node, except the root, has $\Delta - 1$ incident edges, corresponding to the directions upward, downward 1, downward 2, etc. and labelled $\uparrow$, $\downarrow_1, \ldots, \downarrow_{\Delta-1}$. By convention, the edge pointing upward is the edge leading to the root. A node u is a *suspect* if its response is upward and if its parent's response is downward. SearchTree(s, k) works as follows:

SearchTree(s, k)

1. **Detection of a suspect**: follow the downward advice until either a suspect u_l (at distance l from the root) or s is found
2. **Choose a path in a subtree**: traverse the all subtree rooted in u_{l-k} (u_0 if $l < k$) of depth $2k$ and choose to follow the k first edges belonging to the path to leaves with the maximum of downward responses,
3. iterate from first step.

Analyzing SearchTree, we obtain :

Theorem 12 *In a tree of bounded degree Δ of n vertices and k liars, a mobile agent can find an item in at most $d + O((\Delta-1)^{2k+1})$ steps with $O(k \log \Delta)$ bits of memory.*

Proof. For any path of length $2k$ pointing "downward", we know at most k nodes are liars. In Algorithm SearchTree, if the node u_l is reached after such a path, we are sure the destination belongs to the subtree T' rooted in u_{l-k}. If the token is at distance at most $2k$ from u_{l-k}, it will be found in at most $O((\Delta-1)^{2k})$ in Step 2. In the other case, there exists a path from u_{l-k} to a node u_{l+k} containing at least k advices pointing downward. It follows the token should belong to the subtree rooted in the node of this path at depth l.

Each time a suspect is found, the number of remaining liars decreases. It follows that each iteration of Step 2 leads to a visit of a smaller subtree each time. In total, the number of visits is bounded from above by $\sum_{i=k}^{1} O((\Delta-1)^{2i}) = O((\Delta-1)^{2k})$.

To execute the algorithm, the mobile agent must know the number of remaining liars ($O(\log k)$ bits are sufficient), the maximal number of downward advices (bounded by $2k$ and coded with $O(\log k)$ bits) and its position in the subtree of depth at most $2k$. The path from the root u_{l-k} and its current location is of length at most $2k$ and for each node on the path, it only has to store the index of the traversed edge. ∎

This is the first example where the number of steps is exponential in k. However, the next result indicates that the gap between the upper bound and lower bound is not so large when $\delta = \Delta$:

Theorem 13 *Let T be an n node tree with minimal internal node degree $\delta > 2$. If $k < \log_{\delta-1} n$, there exists a distribution of k liars on T such that the number of steps required to find an item at distance d from the origin is $\Omega((\delta-1)^{\min\{k,d\}})$. If $k \geqslant \log_{\delta-1} n$, the minimal number of steps is n.*

Proof. Take the distribution where all nodes at depth less or equal to k point upward toward the root or origin node, i.e., the first k nodes on the correct path are liars. Any deterministic algorithm can be forced to check all paths of length k starting from the root until a truthteller on the path is found or the item is found (on the last try in the case $k \geqslant d$). If $k > \log_{\delta-1} n$ then all nodes point to the root and the advice of the nodes is useless. The item may be placed at the last node searched. ■

8 Conclusion and open problems

We have presented in this paper several models of searching in a network containing liars. A comparison of the models shows that different trade-offs between number of steps and amount of memory (of the mobile agent and at each node) may be considered. For the most part the bounds we present are not tight and improvements are possible.

In our model we assume an upper bound on the number of liars is known. What happens if this assumption is eliminated or weakened? All algorithms in this paper are deterministic. It might be interesting to consider randomized algorithms for these problems.

In some situations the item we search for (either information of service), may be replicated in different nodes of the network. Our algorithms may be adapted to this situation but it becomes more difficult to determine which nodes are lying. In this case, two responses might seem to be locally contradictory but might be in fact correct. Problems in this setting remain open.

References

[1] Y. Afek, E. Gafni, and M. Ricklin, Upper and lower bounds for routing schemes in dynamic networks, in: Proc. 30th Symposium on Foundations of Computer Science, (1989), 370–375.

[2] S. Albers and M. Henzinger, Exploring unknown environments, *in Proc. 29th Symposium on Theory of Computing*, (1999), 416–425.

[3] B. Awerbuch, B. Patt-Shamir, and G. Varghese, Self-stabilizing end-to-end communication, *Journal of High Speed Networks* **5** (1996), 365–381.

[4] R.A. Baeza-Yates, J.C. Culberson and G.J.E Rawlins, Searching in the plane, *Information and Computation* **106(2)** (1993), 234–252.

[5] D. Bienstock and P. Seymour, Monotonicity in graph searching, *Journal of Algorithms* **12** (1991), 239–245.

[6] P.J. Cameron. *Topics, Techniques, Algorithms.* Cambridge University Press, 1994.

[7] R. Cole, B. Maggs and R. Sitaraman, Routing on butterfly networks with random faults, *in: Proc. 36th Symposium on Foundations of Computer Science*, (1995), 558–570.

[8] S. Dolev, E. Kranakis, D. Krizanc and D. Peleg, Bubbles: Adaptive routing scheme for high-speed networks, *SIAM Journal on Computing*, to appear.

[9] N. Hanusse, E. Kranakis, D. Krizanc, Searching with Mobile Agents in Networks with Liars, in the Proceedings of Europar 2000, Arndt Bode, Thomas Ludwig, Wolfgang Karl, Roland Wissmuller, editors, LNCS 1900, pp. 583-590, 2000.

[10] E. Kranakis and D Krizanc, Searching with uncertainty, *in: Proc. 6th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, (1999), C. Gavoille, J.-C, Bermond, and A. Raspaud, eds., pp, 194-203, Carleton Scientific, 1999.

[11] L.M. Kirousis, E. Kranakis, D. Krizanc, and Y.C. Stamatiou. Locating Information with Uncertainty in Fully Interconnected Networks, In proceedings of 14th International Conference, DISC 2000, Toledo Spain, October 2000, SVLNCS, M. Herlihy, ed., pp. 283-296, Vol 1914, 2000.

[12] E. Kushilevitz and Y. Mansour, Computation in noisy radio networks, *in Proc. 9th Symposium on Discrete Algorithms*, 1998, 236–243.

[13] L. Kirousis and C. Papadimitriou, Interval graphs and searching, *Discrete Mathematics* **55** (1985), 181–184.

[14] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou, The complexity of searching a graph, *Journal of the ACM* **35** (1988), 18–44.

[15] T. Leighton and B. Maggs, Expanders might be practical, *in: 30th Proc. Symposium on Foundations of Computer Science*, (1989), 384–389.

[16] P. Panaite and A. Pelc, Exploring unknown undirected graphs, *in: Proc. 9th Symposium on Discrete Algorithms*, (1998), 316–322.

[17] Mobile Agents, W. R. Cockayne and M. Zyda, editors, Manning Publications Co., Greenwich, Connecticut, 1997.
http://www.manning.com/Cockayne/Contents.html