Locating Information with Uncertainty in Fully Interconnected Networks: The Case of Non-Distributed Memory

Lefteris M. Kirousis[1⋆] Evangelos Kranakis[2⋆⋆] Danny Krizanc[3⋆⋆⋆] and Yannis C. Stamatiou[4†]

[1] University of Patras, Department of Computer Engineering and Informatics, Rio 26500, Patras, Greece. e-mail: kirousis@ceid.upatras.gr

[2] Carleton University School of Computer Science, Ottawa, ON, K1S 5B6, Canada. email: kranakis@scs.carleton.ca

[3] Wesleyan University, Department of Mathematics, Middletown, CT 06459. e-mail: dkrizanc@wesleyan.edu

[4] Department of Mathematics,University of the Aegean, Karlovassi, 83200, Samos, Greece. e-mail: stamatiu@aegean.gr. e-mail: stamatiu@aegean.gr

June 18, 2003

Abstract. We consider the problem of searching for a piece of information in a fully interconnected computer network (also called complete network or *clique*) by exploiting advice about its location from the network nodes. Each node contains a database that "knows" what kind of documents or information are stored in other nodes (e.g. a node could be a Web server that answers queries about documents stored on the Web). The databases in each node, when queried, provide a pointer that leads to the node that contains the information. However, this information is up-to-date (or correct) with some bounded probability. While, in principle, one may always locate the information by simply visiting the network nodes in some prescribed ordering, this requires a time complexity in the order of the number of nodes of the network. In this paper, we provide algorithms for locating an information node in the complete communication network, that take advantage of *advice* given from network nodes. The nodes may either give correct advice, by pointing directly to the information node, or give wrong advice by pointing elsewhere. On the lower bounds side, we show that no fixed memory (i.e. with memory independent on the network size) deterministic algorithm may locate the information node in a constant (independent on the network size) expected number of steps. Moreover, if $p = \omega(1/n)$ is the probability that a node of an n-node clique gives correct advice, we show that no algorithm may locate the information node in expected number of steps less than $1/p - o(1)$. In order to study how the expected number of steps is affected by the amount of memory allowed to the algorithms, we give a memoryless randomized algorithm with expected number of steps $4/p + o(1/p) + o(1)$ and a 1-bit randomized algorithm requiring on the average at most $2/p + o(1)$ steps. In addition, in the memoryless case, we also prove a $4/p$ lower bound for the expected number of steps in the case where the nodes giving faulty advice may decide on the content of this advice in any possible way and not merely at random (*adversarial* fault model). Finally, for the case where faulty nodes behave randomly, we give an optimal, unlimited memory deterministic algorithm with expected number of steps bounded from above by $1/p + o(1/p) + 1$.

Key Words and Phrases: Search, Information Retrieval, Complete Network, Uncertainty, Random Walks.

⋆ Partially supported by the Computer Technology Institute.

⋆⋆ Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant and by MITACS project CANCCOM (Complex Adaptive Networks for Computing and Communication).

⋆⋆⋆ Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant and by MITACS project CANCCOM (Complex Adaptive Networks for Computing and Communication).

† Supported by the Greek Ministry of National Economy through a NATO scholarship (contract number 106384/ΔOO 1222/2-7-98) and by the MITACS project CANCCOM. Also partially supported by the Computer Technology Institute.

1 INTRODUCTION

Suppose that we have a network of computers interconnected in some particular topology (e.g. ring, mesh, clique etc.) and one of the computers possesses a piece of information. The objective is to design a *software agent* that is able to travel along the communication links of the network and locate the information as fast as possible. An additional element to this picture, that differentiates the search from the usual random search, is that some of the computers, when queried, will respond with the name of the computer that holds the information. However, some other faulty computers, when queried, will give consistently wrong advice as to which computer has the information. (We do not consider intermittent faults that may appear and then disappear in an unpredictable fashion.) These nodes may choose their advice either in a totally random fashion, which we call the *random model* or in any conceivable way, a variant we call the *adversarial* model.

In this paper we address the problem of locating a piece of information in the *complete network* using, possibly incorrect, advice from the nodes, where each computer may communicate directly with any other computer. This variant of *searching with uncertainty*, was introduced in [15], where the network topologies were the *ring* and the *torus*. Models with faulty information in the nodes have been considered before for the problem of routing (see [1, 3, 6, 8, 16]). 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 in the past. We have the *deterministic search games*, where a fugitive that possesses some properties (it is agile or inert) hides in the nodes or edges of a graph and the aim is to locate it using as few searchers as possible (for definitions and relevant theory see [5, 14, 18]). Also, the problem of exploring an *unknown* graph has been considered (see [2, 19, 20], for example). Closer to the spirit of our work are the search problems that are defined and studied in [4] where the authors consider problems of locating points in the plane using incomplete knowledge about their position. On another front, a number of algorithms have appeared that search a graph relying on some sort of *random walk* along its edges (for a comprehensive and very accessible survey, see [17]). We have a very interesting class of games called *stochastic games* in which opponents' strategies incorporate some sort of probability transition matrix (see [23]). Also, in [24] one may find a detailed treatment of *geometric games* where the aim is to locate some element of a hidden random set.

In this work, we show how to locate a piece of information on the complete network of n nodes using the *advice* that the nodes give when queried. The advice is correct, i.e. it directs the agent to the information node, with some bounded probability $p = \omega(1/n)$. This problem was studied in [12] in the context of World Wide Web searching for search algorithms that are only allowed to use a single bit per network node to remember whether they have encountered the node before. Such algorithms impose minimal overhead on communication links as they visit the network nodes and they are well suited to information retrieval in communication topologies resembling large cliques. In contrast, in this paper we consider algorithms that are not allowed to use *any* storage local to each node that they visit in order to study how variations in their properties (e.g. amount of storage local to them, use of randomization) affect their search efficiency.

The class of algorithms we will consider can execute one of the following operations: (i) query a node of the complete network about the location of the information node (ii) follow the advice given by a queried node and (iii) select the next node to visit using some probability distribution function on the network nodes (that may be, for example, a function of the number of previously seen nodes or the number of steps up to now). A *step* for the algorithm is its movement from one node to another. Throughout the paper, the time complexity measures concern the number of steps (i.e. number of network node visits), considering negligible the computations done in order for the algorithm to decide on the next node to visit.

The algorithms we will consider may be either *memoryless*, have *limited memory* or have *unlimited memory*. An algorithm is memoryless, if it remains in a single state as it moves from a node to the next one. It is of limited memory, if it can be in one of several *fixed*, i.e. independent of the size of the complete network, states. This means that the algorithm can store internally only a *fixed* amount of information which is independent of the size of the complete network. Finally, an algorithm is of unlimited memory if it may assume an unbounded number of states (e.g. the identities of the nodes encountered in the past) that may depend of the size of the complete network. In addition, an algorithm may use randomization or be deterministic.

In this paper, we first prove a lower bound for the expected number of steps needed by any algorithm in order to locate the information node for both the random and the adversarial model of faulty nodes. This bound is equal to $\frac{1}{p} - o(1)$ and it is proved by a reduction to our problem of the problem of hitting a random set in a complete network by an adaptive random walk (whose average number of steps we bound from below). We then show that, for both faulty node models, no bounded memory deterministic algorithm can locate the information node in a constant expected number of steps. The intuitive reason for this is that the algorithm may fall into cycles in the space of its states, for sufficiently large networks. However, we give an optimal deterministic algorithm with unlimited memory (for the random model only) that achieves an expected number of steps bounded from above by $\frac{1}{p}+o(\frac{1}{p})+1$. When randomization is allowed, we give a memoryless randomized algorithm that achieves an expected number of steps bounded from above by $\frac{4}{p} + o(\frac{1}{p}) + o(1)$ while if one bit of memory is allowed we give a randomized algorithm that needs at most $\frac{2}{p} + o(1)$ steps on the average. The last two bounds hold for both the random and the adversarial model of faulty nodes. Furthermore, we prove a $4/p$ lower bound for the expected number of steps required by any memoryless algorithm when the faulty nodes of the network have the ability to determine on the identity of the node they point to not only at random but according to any possible strategy, leaving the case of random determination as an interesting open problem. In the random case, we believe that memoryless algorithms require, asymptotically, less than $4/p$ steps on the average.

2 THE COMPUTER NETWORK MODEL

In order to present and analyze the various algorithmic approaches to the problem of searching for a piece of information in the complete network, we will define a model of the computer network that captures the fact that some computers give valid advice (i.e. they point to the node containing the information) while some others give faulty advice by pointing elsewhere.

The computer network is modeled as a clique on n nodes where there is a direct link between any pair of nodes. Each node also contains the following information:

- *Node identity number.* This is a number within the range $1 \ldots n$ that uniquely identifies each node.
- *Advice.* This is also a number within the range $1 \ldots n$ and it is interpreted as the advice of the node as to which node of the network contains the information.

In order to model the fact that the advice of a node may be incorrect with some probability as well as the fact that the information may reside in any node in a random fashion, we introduce randomness to the model in the following way:

- A node is randomly and uniformly selected from among the n nodes that contain the information and its identity, say s, becomes known to the other nodes. Node s sets its *advice* to be equal to itself, i.e. to contain s. In this way, any algorithm may distinguish a node that contains the information from one that does not.
- Each node, except s, flips a coin that shows heads with probability p and if head shows-up, then the node sets its *advice* equal to s. These nodes give correct advice about the location of the information. The node for which tails appeared, the *faulty* nodes, must point to some node other than the information node by choosing one of the node identities in the set $\{1 \dots n\} - \{s, \mathrm{id}\}$.

After this procedure is carried out, *no changes* are permitted to any piece of information that was determined by the procedure. That is, the sets of faulty and non-faulty nodes remain the same and the wrong advice given by the faulty nodes never changes.

An important issue related to the faulty nodes is how they choose their wrong advice. In relation to this issue, we may define the following two models:

- *Model A or random model.* In this model, the faulty nodes make their choices simply at random.
- *Model B or adversarial model.* This model, allows the faulty nodes to decide their wrong advice in any possible way. They may even "conspire" in order to confuse the search algorithm by, say, pointing (all of them) to the same node. This could possibly confuse, for example, algorithms that make their moves based on the advice given by the majority of encountered nodes.

The usefulness of defining both models, from a theoretical perspective, is that we can examine how the various lower or upper bounds we derive for our search problem depend on the behaviour of the faulty nodes. For example, lower bounds that are proved for the random model, also hold for the adversarial model while upper bounds proved in the adversarial model, hold for the random model. Also, interestingly, for the derivation of the upper bound stated in Theorem 7, it seems that we need to assume that the faulty nodes behave in a random fashion.

From a practical perspective, the adversarial model would be useful in modeling situations where a set of nodes point to the same wrong node because the information item has moved to another node and the nodes that pointed to it have not updated their links yet.

3 SOME LOWER BOUNDS FOR SEARCHING IN THE CLIQUE

Let $K_n = (V_n, E_n)$ be the clique graph on n vertices. In this section, we will prove that if $p = \omega(1/n)$, then no algorithm that searches for a piece of information in K_n (under the model we have defined) can do, asymptotically, better than executing $\frac{1}{p} - o(1)$ steps on the average. To this end, we will use the fact that given a random set of node $S \subset V_n$ of cardinality k an *adaptive* random walk that starts from a node in $V_n - S$ will hit some node in S in expected time that is equal to $\frac{n+1}{k+1}$. An adaptive random walk is simply a random walk in which after each step, the currently visited node, along with its edges, is deleted from the clique.

Our aim is to show that the problem of locating the information node is at least as difficult as the problem of locating one of k non faulty nodes using a random walk that starts from a randomly chosen initial node. The following proposition holds:

Proposition 1. *For a randomly selected set S of non-faulty nodes, the average number of steps for locating the information node $t \in S$ is no less than the average number of steps required to hit the random set S.*

It is easy to see why this proposition is valid by observing that *any* algorithm that locates the *randomly chosen* information node in some number of steps may also be used to hit a random set to which the information node belongs, in the same number of steps. Also note that this proposition holds for both the random and the adversarial model for the faulty nodes, thus making the lower bound given in Theorem 2 below valid for both models.

Now given a node v in V_n, we denote by $r_v(u)$ the probability of the adaptive random walk choosing vertex $u \neq v$ when it leaves from node v. In the uniform adaptive random walk on K_n, $r_v(u) = \frac{1}{n-1}$ for every v, u. The following can be proved:

Lemma 1. *The uniform adaptive random walk is an optimal algorithm for hitting a randomly selected set of k clique nodes.*

Proof. Suppose that an algorithm uses a non-uniform distribution function $r_v(u)$ (that may also be a function of the current step) and that it is about to select the next vertex to visit. Then the probability of failure to select at step i a node from the random set of size k is

$$\begin{aligned} P_v &= \sum_{u \text{ adjacent to } v} r_v(u)\left(1 - \frac{k}{n-i}\right) \\ &= \left(1 - \frac{k}{n-i}\right) \sum_{u \text{ adjacent to } v} r_v(u) = \left(1 - \frac{k}{n-i}\right), \end{aligned}$$

since nothing has been revealed about the initial random set of k nodes (it is not affected by the deletion of the i clique nodes as this set does not have common elements with the random set). Therefore, we see that the probability of failure from any node is independent of the probability distribution chosen for the node by the algorithm. Consequently, the uniform distribution is an optimal one.

Now no algorithm can have at the ith step a probability of failure less than the probability of failure of the adaptive random walk. This is because for such an algorithm the set of the k selected nodes is still unknown and, after the deletion of i nodes, the remaining nodes can be regarded to form the clique K_{n-i}. Therefore, the probability of not hitting this set, using *any* distribution on the outgoing edges of the currently visited node, is $\frac{k}{n-i}$. ■

According to the lemma above, the number of steps taken by the adaptive random walk is a lower bound for the number of steps required by any algorithm to locate one of the randomly chosen nodes. The expected number of steps, $E(n,k)$, required by the uniform adaptive random walk is given by the following theorem.

Theorem 1. *Let $E(n,k)$ be the expected number of steps required by the uniform, adaptive random walk to hit a set of k randomly selected nodes of a clique of n nodes. Then $E(n,k)$ is given by the following expressions:*

$$E(n,k) = \sum_{i=1}^{n-k+1} i \prod_{l=1}^{i-1} \left(1 - \frac{k}{n-l+1}\right) \frac{k}{n-i+1} = \sum_{i=1}^{n-k+1} i \frac{\binom{n-i}{k-1}}{\binom{n}{k}} = \frac{n+1}{k+1}. \tag{1}$$

Proof. The first equality in (1) follows from the definition of the expectation of the number of steps (note that the random walk we examine terminates in at most $n-k+1$ steps) as the expression

$$\prod_{l=1}^{i-1} \left(1 - \frac{k}{n-l+1}\right) \frac{k}{n-i+1}.$$

is simply the probability that the uniform adaptive random walk terminates after exactly i steps: the product from $l = 1$ to $l = i - 1$ gives the probability of failure at steps 1 up to $i - 1$ while the last product gives the probability of success at the ith step.

In order to see why the second equality is true, we will show that the probability that the uniform, adaptive random walk terminates after exactly i steps is also given by $\frac{\binom{n-i}{k-1}}{\binom{n}{k}}$. Since the algorithm terminates after i steps, i.e. at the ith steps it hits on of the k randomly selected non-faulty nodes, then $i - 1$ nodes are faulty and 1 is non-faulty. Then any subset of $k - 1$ out of the remaining $n - i$ nodes can be the rest of the $k - 1$ non-faulty nodes. Thus, success at i steps can occur in $\binom{n-i}{k-1}$ possible placements of the k non-faulty nodes. Dividing with the number of all possible placements of these nodes gives us the equality.

Finally, to show the last equality, we will show that

$$\sum_{i=1}^{n-k+1} i\binom{n-i}{k-1} = \binom{n+1}{k+1} \tag{2}$$

from which the equality easily follows. The right hand-side of (2) is the number of $k + 1$ element subsets of a given set of $n + 1$ elements. Let, without loss of generality, this set be the set $\{1, 2, \ldots, n + 1\}$. Then we can partition the $k + 1$ element subsets of this set into classes S_i as follows: in the ith class we place these subsets whose second largest element is equal to $n - i + 1$ and the largest element is larger than $n - i + 1$ (note that since $k > 0$ that there are at least 2 elements in each subset of $k + 1$ elements and, thus, our definition of the ith class is meaningful). In addition, i can only run from $i = 1$ to $n - k + 1$. Clearly, these classes form a partition of the set of subsets with $k + 1$ elements from the set $\{1, 2, \ldots, n + 1\}$. Since for each value of i, there are $i\binom{n-i}{k-1}$ subsets in class i, the last equality of (1) follows. ∎

Now we can prove the following:

Theorem 2. *Suppose that each node in the complete network on n nodes decides to be non-faulty independently of the others with probability $p = \omega(1/n)$. Then the expected number of steps required to locate the information node is bounded from below by an expression equal to $(1/p) - o(1)$.*

Proof. Let S denote the expected number of steps before locating the information node in a clique of n nodes, where each node decides decides independently of the others to be non-faulty with probability $p = \omega(1/n)$. Let k denote the random variable that counts the non-faulty nodes. According to the *law of total probabilities for expectations* (see, for example, [13]):

$$\mathbf{E}[S] = \mathbf{E}[S \mid k = 1] \cdot \mathbf{Pr}[k = 1] + \mathbf{E}[S \mid k = 2] \cdot \mathbf{Pr}[k = 2] + \cdots + \mathbf{E}[S \mid k = n] \cdot \mathbf{Pr}[k = n]. \tag{3}$$

From Equation (1), $\mathbf{E}[S \mid k = j] \geq E(n, j)$ (as the adaptive random walk is optimal). Also, $\mathbf{Pr}[k = j] = \binom{n}{j}p^j(1 - p)^{n-j}$. Then it follows from (3) that

$$\begin{aligned}\mathbf{E}[S] &\geq \sum_{k=1}^{n}\binom{n}{k}E(n, k)p^k(1 - p)^{n-k} = \sum_{k=1}^{n}\binom{n}{k}\frac{n+1}{k+1}p^k(1 - p)^{n-k} \\ &= \frac{1}{p} - \frac{(1-p)^{n+1}}{p} - (n + 1)(1 - p)^n. \end{aligned} \tag{4}$$

Since $p = \omega(1/n)$, the last two terms are $o(1)$ each. Thus

$$\mathbf{E}[S] \geq \frac{1}{p} - o(1)$$

which proves the required. ■

The theorem above shows that the expected number of steps of any algorithm that locates the information node is bounded from below by an expression whose dominating term is $\frac{1}{p}$. Thus no algorithm may achieve an expected number of steps given by an expression with dominating term $\alpha\frac{1}{p}$, for $\alpha < 1$.

4 DETERMINISTIC SEARCH ALGORITHMS

We will now give an argument for showing that no fixed memory deterministic algorithm exists that can achieve a constant expected number of steps. In other words, randomization is *necessary* if the algorithm's memory is a constant quantity, independent of the size of the complete network.

Theorem 3. *No fixed memory deterministic algorithm can locate the information node in the complete network of any size in a constant, independent on the network size, expected number of steps.*

Proof. Let A be a deterministic algorithm with a fixed amount of memory, that locates the information node in cliques of any size. A deterministic algorithm that performs one of the operations we described, can be thought of as a function $f : (S, V) \mapsto S$, where S is the fixed set of different states A may assume and $V = \{v_1, \ldots, v_n\}$ is the set of network nodes. The effect of the transformation of a state $j \in S$ to a state $j' \in S$ is that the algorithm moves from the current node to the next one. A natural, but crucial to our argument, assumption that we make is that the algorithm's decision should somehow be encoded in the new state i'. In this way, the computation of the algorithm may be viewed as a sequence $(j_0, v_{j_0}), (j_1, v_{j_1}), \ldots, (j_t, v_{j_t})$ of steps with the last step being the one that verifies that the node v_t has the information.

We will first show that no fixed memory, deterministic algorithm exists that locates the information node in complete networks of any size and does not follow the advice of at least one of the nodes it encounters. This is easy to see as the set of different states of the algorithm is fixed and, thus, only finitely many decisions to move to new nodes can be encoded in its states. Therefore, for sufficiently large complete networks, the algorithm will fail as it will have to repeat some state, entering a cycle in its state space and repeating the state loop for ever. Following the advice at some point, effectively gives to the algorithm the ability to visit another node "for free". Following the advice simply once, however, leads again to the same problem with the algorithm entering a cycle in its state space for a sufficiently large complete network. This is true since, after following the advice, the algorithm will again act relying only on its fixed set of states in order to make its next decision. Repeating this reasoning, we conclude that the algorithm will have to follow the advice of nodes it encounters an unbounded number of times, as the size of the complete network increases. However, this will eventually cover all the possible different configurations of the algorithm and the algorithm will find itself in an infinite loop. ■

The essence of the above argument is that a fixed memory deterministic algorithm should follow the advice unboundedly often as the size of the network increases. However, in doing this, it will eventually enter a cycle in its state space with fixed and nonzero probability.

It can also be proved that if we alter our model so that when a faulty node is queried again it determines its wrong advice at random and does not necessarily give the same wrong advice, then the "always follow the advice" policy results in an algorithm with expected number of steps equal to $\frac{1}{p}$.

5 RANDOMIZED ALGORITHMS

In this section, we will examine the memoryless and fixed memory cases for algorithms that use randomization. In view of Theorem 3, the use of randomization is indispensable.

Before continuing with the statement of the algorithms and their analysis, we will mention two formulas that will be useful in evaluating certain sums that will arise in our calculations (see, e.g., [11] for details on these, as well as, similar summation formulas):

$$\sum_{l=2}^{\infty}(l-1)la^l = \frac{2a^2}{(1-a)^3} \tag{5}$$

and

$$\sum_{l=0}^{\infty}(l+1)a^l = \frac{1}{(1-a)^2}. \tag{6}$$

5.1 A memoryless randomized algorithm

The algorithm we will describe is memoryless and in order to decide at each step whether it will follow the advice of the current node or it will choose the next node to visit at random, it flips a fair coin.

Algorithm: Memoryless Search
Input : A clique (V, E) with a node designated as the information holder
Aim: Find the node containing the information

1. **begin**
2. current = RANDOM(V) // Pick a node uniformly at random from the set of nodes V.
3. **while** current(information) $\neq$ **true** // While current node does not have the information
4. c = FLIP(1/2) // Flip a fair coin.
5. **if** $c =$ HEADS
6. current = RANDOM($V -$ current) // Choose uniformly at random from remaining nodes
7. **else**
8. current $\leftarrow$ advice(current) // Follow node's advice
9. **end while**
10. **end**

Theorem 4. *For the adversarial model (and, thus, the random model) if $p = \omega(1/n)$, the algorithm* Memoryless Search *locates the information node in expected number of steps at most equal to $4/p + o(1/p) + o(1)$.*

Proof. The algorithm succeeds if at the last decision it makes (random selection in line 6 or follow advice in line 8) it lands on the information node. For the analysis, let l count the number of steps, initially set to 1. Then it is increased each time the loop that starts at line 3 is repeated.

In attempting to analyze probabilistically the complexity of the algorithm we are faced with the difficulty that some of its steps *expose* information about the initially random set of faulty and non-faulty nodes. For example, consider the last two flips of a sequence of coin flips and assume that they are both tails. Assuming, further, that the first flip leads the algorithm to a faulty node so that the algorithm does not terminate at the second flip, what is the probability of success at the second flip? It is equal to the probability that the node reached after the first flip is non-faulty. One would say that this probability is p but this is not correct. If the node was encountered by the algorithm at some previous step, then the node is certainly faulty, otherwise the algorithm would have terminated at that step. Thus, now that the node is encountered a second time, the probability that it is non-faulty is equal to 0 implying that at the second T flip of our scenario the probability of success is also 0. The situation changes if the node reached after the first flip was never met before. Then the probability of success after the second flip results in tails would indeed be equal to p. For these reasons, in the analysis that follows we will concentrate on the realization of the success event from sequences of coin flips ending with HT (HEADS (H), TAILS (T)) as it is only for such sequences that we may claim that the probability of success at the T step is equal to p since the previous step essentially "randomizes" the T step that follows. Of course this means that we will only obtain an upper bound to the expected number of steps of the algorithm. In addition, we will ignore sequences ending in HH and TH as they have negligible effect on the expectation (as n tends to infinity, success in such sequences is becoming increasingly rare).

In view of the above discussion, then, our strategy is to study the expected length of an H, T sequence that ends up with a successful T, while the Ts in the middle in the sequence are all unsuccessful (i.e. the information node was not encountered). Depending on the positions of the Ts, given a sequence of coin flips ending with HT (where T is followed by success) we can divide it into *segments* or *epochs* as follows: The first epoch, e_1, starts from step $l = 1$ and ends at the step where T appeared right after H. Let X_1 be the number of coin flips of epoch e_1. The ith epoch, $i > 1$, is defined similarly starting at the next flip after the last one (which is a T by definition) of epoch $i-1$. Also, let X_i be the number of coin flips in epoch e_i. Each time an epoch ends, we consider that we perform an experiment with probability of success k/n, with k the number of non-faulty nodes, which in the context of our algorithm is the probability of having reached a non-faulty node at the H step before the T (the two flips that mark the end of epoch). Let S be the number of epochs formed at the end of the algorithm. Then the expected number of coin flips X required for success in the experiment, given that the number of epochs is $S = s$, is the following:

$$\mathbf{E}[X/s] = \mathbf{E}[\sum_{i=1}^{s} X_i] = \sum_{i=1}^{s} \mathbf{E}[X_i] = s\mathbf{E}[X_1], \tag{7}$$

as the random variables X_i are independent and identically distributed and their sum is equal to X. It is not hard to see that the expectation of X_1 is the following:

$$\mathbf{E}[X_1] = \sum_{l=1}^{\infty} l \frac{\text{number of possible H/T strings with } l \text{ letters, ending in HT}}{\text{number of possible H/T strings with } l \text{ letters}}. \tag{8}$$

In order to compute the numerator in the above expression, called *success strings* for brevity, we work as follows. For $l = 2$, there is only one success string: the string HT. In order to construct the success strings for any $l > 2$, fix the last $l-2$ letters to be HT. Then the preceding $l-2$ positions can be filled in with Hs and Ts so that no H appears before a T. But this leaves only one possibility to fill in the remaining $l-2$ positions: we start with r_T, $r_T \geq 0$ Ts, and continue with r_H, $r_H \geq 0$ Hs, so that $r_T + r_H = l - 2$. This can be done in exactly $l-1$ ways.

Continuing from (8) and using (5), we obtain the following estimate:

$$\mathbf{E}[X_1] = \sum_{l=2}^{\infty} l \frac{l-1}{2^l} = 4. \tag{9}$$

From (7) and (9) it follows that $\mathbf{E}[X/s] = 4s$.

According to the law of total probabilities for expectations (see, e.g., [13], page 8), the following holds:

$$\mathbf{E}[X] = \sum_{s=1}^{\infty} \mathbf{E}[X/s] \Pr[S = s].$$

Thus:

$$\mathbf{E}[X] = \sum_{s=1}^{\infty} 4s \left(1 - \frac{k}{n}\right)^{s-1} \frac{k}{n} = \frac{4n}{k}. \tag{10}$$

In order to find an upper bound to the expected value of the number of steps in our model where each network node decides to be non-faulty with probability p, we work as follows (note that the first equality uses, again, the law of total probability for expectations):

$$\begin{aligned}\mathbf{E}[\#\text{ of steps}] &\leq \sum_{k=1}^{n} \mathbf{E}[X] \binom{n}{k} p^k (1-p)^{n-k} = 4n \sum_{k=1}^{n} \frac{1}{k} \binom{n}{k} p^k (1-p)^{n-k} \\ &= 4n \sum_{k=1}^{n} \left(\frac{1}{k+1}\right) \binom{n}{k} p^k (1-p)^{n-k} + 4n \sum_{k=1}^{n} \left(\frac{1}{k(k+1)}\right) \binom{n}{k} p^k (1-p)^{n-k} \end{aligned} \tag{11}$$

Will we be, first, concerned with the first sum. Let

$$A(z) = 4n \sum_{k=1}^{n} \left(\frac{1}{k+1}\right) \binom{n}{k} p^k (1-p)^{n-k} z^k \text{ and } B(t) = \sum_{k=1}^{n} \binom{n}{k} p^k (1-p)^{n-k} t^k.$$

Then the first sum in (11) is equal to A(1).

Using ordinary generation function properties (the "right shift" and "index divide" properties – see [11], Section 3.1) or simply by inspection, we can see that the following holds:

$$A(z) = 4n \sum_{k=1}^{n} \left(\frac{1}{k+1}\right) \binom{n}{k} p^k (1-p)^{n-k} z^k = 4n \frac{1}{z} \int_0^z B(t) dt.$$

The integral above can be computed easily and it gives

$$\begin{aligned} A(z) &= 4n \left(\frac{1}{zp(n+1)} - \frac{1}{z(n+1)} + \frac{1}{n+1}\right) [1 - p(1-z)]^n \\ &\quad - 4n \left(\frac{1}{zp(n+1)} - \frac{1}{z(n+1)} + \frac{1}{n+1} + \frac{n}{n+1}\right) (1-p)^n \end{aligned} \tag{12}$$

from which we conclude, setting $z = 1$, that

$$4n \sum_{k=1}^{n} \left(\frac{1}{k+1}\right) \binom{n}{k} p^k (1-p)^{n-k} z^k = 4n \frac{1}{p(n+1)} - 4n \left(\frac{n}{n+1} (1-p)^n\right) - 4n \frac{(1-p)^n}{p(n+1)},$$

which is equal to $\frac{4}{p} - o(1)$ due to the fact that $p = \omega(1/n)$.

We will now show that the second summand is $o(1/p) + o(1)$, as n tends to infinity. To this end, set $\alpha = \frac{k}{n}$, for the various values of k. Using Stirling's approximation formula, each of the n terms of the second sum in (11) is, asymptotically, of the form (see, e.g., [11])

$$4n \frac{1}{\alpha n(\alpha n + 1)} \frac{1}{\sqrt{2\pi n \alpha (1-\alpha)}} \left[\left(\frac{p}{\alpha}\right)^{\alpha} \left(\frac{1-p}{1-\alpha}\right)^{1-\alpha} \right]^n$$

for various values of α in the interval $(0,1]$ (there can only exist n values for α, since $\alpha = k/n$, for $k = 1 \dots n$). The expression raised to n, is strictly less than 1, for values of $\alpha \neq p$ while it is equal to 1, when $\alpha = p$. In the former case, the sum of the corresponding terms vanish as there are polynomially many of them and each of them is exponentially small. In the latter case, i.e. $\alpha = p$, the terms are of the following form:

$$4n \frac{1}{pn(pn+1)} \frac{1}{\sqrt{2\pi np(1-p)}}.$$

As $p = \omega(1/n)$, we can easily see that the above term is $o(1/p)$ (we simply check that the ratio of this term with $1/p$ vanishes). Thus the second sum is equal to $o(1/p) + o(1)$.

Gathering the estimates for the two sums, we conclude that

$$\mathbf{E}[\# \text{ of steps}] = \frac{4}{p} + o\left(\frac{1}{p}\right) + o(1)$$

completing the proof of the theorem. ∎

5.2 Generalizing the memoryless randomized approach

In this section we will analyze a generalization of the memoryless algorithm described in the previous section in which the probability of obtaining heads is h and the probability of obtaining tails is t. To this end, we will derive the exact expectation for the number of steps required by the algorithm in the general t, h case. Success at a specific step s of the execution of the algorithm may occur either as a result of two consecutive coin tossess equal to HT respectively (the case analyzed in the previous section) or equal to TT, TH or, finally, HH all these realizations of success being mutually exclusive.

Before going into the analysis details recall that the case with two consecutive tails, or TT-case as we will call it for brevity, was special in the sense that the probability of success of a coins sequence ending in TT depends on whether the node on which the algorithm lands following immediately its advice (second of the two Ts) as this node may have been visited in the past *again* with its adviced followed. Note that if the same node was encountered before only during H steps, i.e. the algorithm did not follow the advice but alwayes chose the next node at random, the situation would be much different as the TT-case would succeed with probability p. If the node was encountered in the past, however, with its advice taken then this probability is 0 as the node is certainly faulty. If by d_i we denote the probability that the first T occuring at step i of a TT-case leads to a previously encountered node whose advice the algorithm followed, then with the use of conditional probabilities the probability of success at step i, denoted by x_i, of a TT-case starting at step $i-1$ of the execution of the algorithm is equal to

$$x_i = (1 - d_{i-1}) \cdot p + d_{i-1} \cdot 0 = (1 - d_{i-1})p$$

with $i \geq 2$ and $d_1 = 0$.

Therefore, we may write the expectation of the number of steps required by the algorithm as follows:

$$\begin{aligned}\mathbf{E}[\# \text{ of steps}] &= \sum_{s=1}^{\infty} s \Pr[R = s] \\ &= \sum_{s=1}^{\infty} s \prod_{i=1}^{s-1} \left[1 - \left(ttx_i + htp + hh\frac{1}{n} + th\frac{1}{n}\right)\right] \left(ttx_s + htp + hh\frac{1}{n} + th\frac{1}{n}\right).\end{aligned} \tag{13}$$

Theorem 5. *For the adversarial model, if $p = \omega(1/p)$, no memoryless randomized algorithm may locate the information node in less than $4/p$ expected number of steps.*

Proof. In view of Equation (13), an adversary wishing to maximize the expected number of steps that the algorithm spends can affect no other parameter except for the d_i's as it is not allowed to decrease p or change t in order, for example, to favour steps having the least probability of success.

By observing the role of the d_i's in the expression, we deduce that the expectation acquires its maximum value (keeping n, p, t constant) if $d_i = 0$, for all $i > 2$. In this way, no TT-case may be advantageous for the memoryless algorithm but the algorithm can never know as it cannot remember previous TT-case outcomes. Setting $d_i = 0$ we obtain

$$\begin{aligned}\mathbf{E}[\#\text{ of steps}] &= \sum_{s=1}^{\infty} s \left[1 - \left(htp + hh\frac{1}{n} + th\frac{1}{n}\right)\right]^{s-1} \left(htp + hh\frac{1}{n} + th\frac{1}{n}\right) \\ &= \frac{n}{(1-t)(tpn+1)}. \qquad (14)\end{aligned}$$

Optimizing for t, we find that (14) attains its minimum (i.e. the best possible for the algorithm) for

$$t_{\text{opt}} = \frac{1}{2}\frac{pn-1}{pn}$$

which, as $p = \omega(1/n)$, is asymptotically equal to 1/2. Substituting this value for t into (14) we see that the algorithm's best possible performance, when TT-cases are useless due to the adversary's actions, is equal to $4/p$, asymptotically. ■

We will now present an adversarial scenario that effectively nullifies any advantage of TT-cases for a memoryless algorithm that blindly flips a coin to decide on its next action.

Definition 1 (Cycle adversary). *A cycle adversary is an adversary who, after the faulty nodes have been selected and before they attempt to choose at random the node to point to (except the information node), it forces them to form a maximal cycle.*

Then the following Corollary easily follows from Theorem 5:

Corollary 1. *The cycle adversary forces the memoryless algorithm to take on the average 4/p steps, at best, before it locates the information node.*

Proof. We simply observe that no TT-case may lead to success as either the algorithm moves on the area of non-faulty nodes, in which case the first T will lead it to success or, if the first T does not lead the algorithm to success, then it certainly moves within the set of faulty nodes and the second T of the TT-case is always bound to fail at any step of the execution of the algorithm. Thus $d_i = 0$ for all steps $i \geq 2$ of the algorithm. The 4/p bound follows by setting $t = 1/2$, which reflects the best possible strategy for the memoryless algorithm. ■

One may now justifiably inquire into what happens against a *random* adversary. This (weaker) adversary is not able to *directly* manipulate the d_i's in (13) and we should expect the algorithm to do better than $4/p$, hopefully achieving the optimal $1/p$. Unfortunately, giving a good estimate for the d_i's seems a very difficult task and our efforts, so far, to analyze them were not fruitful. What is obvious is that, since $x_i \leq p$ for all steps of the algorithm (and it can be equal to p only for a fixed number of initial steps), the algorithm can do no better than $1/(tp)$, which is the expected number of steps that results if we replace all x_i's in (13) with p. Thus, if x_i was equal to p for all i then the algorithm could approach arbitrarily close to the optimal expectation without, of course, achieving it as $t = 1$ implies that the algorithm gets trapped within a cycle. However, this situation is overoptimistic.

5.3 One bit of memory helps

In this section, we will show that one bit of memory suffices in order to reduce the dominant term in the complexity figure for the expected number of steps from $\frac{4}{p}$ (memoryless algorithm) to $\frac{2}{p}$, when $p = \omega(1/n)$. In view of our analysis in the previous subsection, the intuition behind this improvement is the ability of the algorithm to eliminate TT steps (and also HH steps, incidentally) and, thus, eliminate their contribution to the probability of failure at each step. The algorithm is always on the safe side with a probability of success equal to p during HT transitions and $1/n$ during TH transitions.

More precisely, the upper bound for the expected number of steps will be reduced to $\frac{2}{p} + o(1)$. The algorithm given below, simply alternates between following the advice of the currently visited node and selecting a random node as the next node to visit. It only needs to remember if at the last step it followed the advice or not. Although one bit suffices, the algorithm is stated as if it knew the number of steps it has taken. Then it simply checks if this number is odd or even. However, one bit would be sufficient. The only reason that we state the algorithm so that is uses the variable l is to facilitate the analysis of its expected number of steps.

Algorithm: Fixed Memory Search
Input : A clique (V, E) with a node designated as the information holder
Aim: Find the node containing the information

```
begin
  current = RANDOM(V) // Pick a node uniformly at random from the set of nodes V.
  l ← 1 // The variable l is not required. It is used only to simplify subsequent analysis.
  while current(information) ≠ true // While current node does not have the information
     l ← l + 1
     if l mod 2 = 0
        current = current(advice) // Follow node's advice
     else
        current ← RANDOM(V − current) // Pick one of the remaining nodes at random
  end while
end
```

Theorem 6. *For the adversarial model (and, thus, the random model) if $p = \omega(1/n)$, the algorithm* Fixed Memory Search *locates the information node in expected number of steps at most equal to $2/p + o(1)$.*

Proof. The intuition behind the existence of the factor 2 at the time complexity of this algorithm is that the steps where the algorithm follows the advice have a probability of success p, i.e. the probability that the preceeding randomized step led the algorithm to a node that points to the information node. Thus, the expected number of trials until such a node is encountered is $1/p$. On the other hand, the expected number of random steps the preceed advice-following steps until a node that points to the information is found is, again, $1/p$. Assuming that no success occurs at the random steps, we obtain an upper bound for the expectation of the algorithm equal to $2/p$.

For the detailed analysis, at step l, failure can occur in one of the following ways:

– If $l = 1$, the algorithm fails if it randomly selects one node other than the node containing the information. The probability for this happening is $q_1 = (1 - \frac{1}{n})$.

– If l is even, the algorithm fails if the currently visited node is faulty. The probability of this event is $q_l = q = 1 - p$.
– If l is odd, the algorithm fails if it randomly selects a node other than the currently visited one, that does not contain the information. The probability of this event is $q_l = (1 - \frac{1}{n-1})$.

Then the expected number of steps (expectation of the random variable l) is the following (where $q_l = 1 - p_l$):

$$
\begin{aligned}
\mathbf{E}[\text{\# of steps}] &= \sum_{l=1}^{\infty} l \cdot \Pr[\text{first success in } l\text{-th step}] = \sum_{l=1}^{\infty} l \cdot q_1 \cdots q_{l-1} p_l \\
&= \left(1 - \frac{1}{n}\right) \sum_{l \geq 2,\ \text{even}} l q^{\frac{l}{2}-1} \left(1 - \frac{1}{n-1}\right)^{\frac{l}{2}-1} (1-q) \\
&+ (1 - \frac{1}{n}) \sum_{l \geq 3,\ \text{odd}} l q^{\frac{l+1}{2}-1} \left(1 - \frac{1}{n-1}\right)^{\frac{l-1}{2}-1} \frac{1}{n-1} + \frac{1}{n} \\
&= \left(1 - \frac{1}{n}\right) \sum_{l \geq 0} 2(l+1) q^l \left(1 - \frac{1}{n-1}\right)^l (1-q) \\
&+ \left(1 - \frac{1}{n}\right) \sum_{l \geq 0} [2(l+1)+1] q^{l+1} \left(1 - \frac{1}{n-1}\right)^l \frac{1}{n-1} + \frac{1}{n}.
\end{aligned}
$$

After some algebraic manipulations that rely on (5) and (6), we obtain the following (note that the inequality holds for $n \geq 2$):

$$
\begin{aligned}
\mathbf{E}[\text{\# of steps}] &= \left(1 - \frac{1}{n}\right) \frac{2n - p - 1}{1 + pn - 2p} + \frac{1}{n} \leq \frac{n-1}{n} \frac{2n}{p(n-2)} + \frac{1}{n} \leq \left(\frac{n-1}{n-2}\right) \frac{2}{p} + \frac{1}{n} \\
&= \left(1 + \frac{1}{n-2}\right) \frac{2}{p} + \frac{1}{n} = \frac{2}{p} + \frac{2}{p(n-2)} + \frac{1}{n}.
\end{aligned}
$$

Since $p = \omega(1/n)$, from the above expression it follows that $\mathbf{E}[\text{\# of steps}] \leq \frac{2}{p} + o(1)$. ■

6 UNLIMITED MEMORY: RANDOMIZED AND DETERMINISTIC ALGORITHMS

We will now give an unlimited memory randomized algorithm for locating the information node in a clique. More specifically, the algorithm can remember the identities of previously visited nodes using in a array of $O(n)$ bits and use this knowledge in order to decide its next move. In this way, the algorithm avoids visiting again previously visited nodes. Such an algorithm always terminates within $n - 1$ steps in the worst case.

Algorithm: Unlimited Memory Randomized Search
Input : A clique (V, E) with a node designated as the information holder
Aim: Find the node containing the information
1. **begin**
2. $l \leftarrow 0$
3. current = RANDOM(V) // Pick a node uniformly at random from the set of nodes V.
4. **while** current(information) $\neq$ **true** // While current node does not have the information

```
5.          l ← l + 1
6.          M ← M ∪ {current} // M holds the up to now visited nodes.
7.          read(current(advice)) // Read the advice of the visited (current) node.
8.          if advice ∉ M
9.             M ← M ∪ {advice}
10.            current = advice
11.         end if
12.         else
13.            current ← RANDOM(V − M) // Pick one of the remaining nodes at random
14.     end while
15. end
```

We can now prove the following:

Theorem 7. *In the random model, if $p = \omega(\frac{1}{n})$, then the expected number of steps required by the algorithm* Unlimited Memory Randomized Search *in order to locate the information node is, asymptotically, bounded from above by $\frac{1}{p} + o(\frac{1}{p} + 1$.*

Proof. The algorithm fails at step $l \geq 1$ if the *while* test returns false. This can occur if either of the following two (mutually exclusive) events had occurred just before the execution of the test:

1. Event A_1: the node *current* in line 6 gives wrong advice *and* the node *advice* (i.e. the answer to the query in line 7) is one of the previously visited values (line 13 is executed) *and* the random choice in line 13 does not lead to the information node. The probabilities of these three events are q (i.e. the probability that a node is faulty), $\frac{l-1}{n-2}$ and $\frac{n-(l-1)-2}{n-(l-1)-1}$ respectively. Note that we have used here the fact that we work in the random model since we compute the probability that the answer to the query leads to one of the previously visited nodes as $(l-1)/(n-2)$. If the faulty nodes had decided their wrong advice in ways other than random, this probability could depend on *which* are the nodes in the set of the previously visited nodes and not simply on the cardinality of the set (which is equal to $l-1$).
2. Event A_2: the node *current* in line 6 gives wrong advice *and* the answer v is a node that lies outside the previously visited. The probabilities of these two events are q and $\frac{n-(l-1)-2}{n-2}$ respectively.

Let E_l be the event that at steps $1 \ldots l$ the algorithm failed to find the information. Then the probability of failure at the lth step, given that the algorithm has failed at steps $1, \ldots, l-1$, is equal to the following (remember that $q = 1 - p$ is the probability that a node is faulty):

$$\begin{aligned} q_l &= \Pr[\text{failure in } l\text{-th step}|E_{l-1}] = \Pr[A_1|E_{l-1}] + \Pr[A_2|E_{l-1}] \\ &= q\frac{l-1}{n-2} \cdot \frac{n-(l-1)-2}{n-(l-1)-1} + q\frac{n-(l-1)-2}{n-2} \\ &= q\frac{n-(l-1)-2}{n-2}\left(1 + \frac{(l-1)}{n-(l-1)-1}\right) = q\frac{n-l-1}{n-l} \cdot \frac{n-1}{n-2}. \end{aligned} \tag{15}$$

Obviously, when $l = 0$, $q_0 = 1 - 1/n$ since the algorithm fails when the random choice does not lead to the unique node that carries the information.

It follows that the expected number of steps of the above algorithm is given by the following formula, in which $p_l = 1 - q_l$ and for each q_l, $l \geq 1$, we used the upper bound $q_l \leq q\frac{n-1}{n-2}$ (from (15)):

$$\begin{aligned}
\mathbf{E}[\#\text{ of steps}] &= \sum_{l=0}^{n-1}(l+1)\cdot \Pr[\text{first success in } l\text{-th step}] \\
&= \sum_{l=0}^{n-1} l\cdot \Pr[\text{first success in } l\text{-th step}] + \sum_{l=0}^{n-1}\Pr[\text{first success in } l\text{-th step}] \\
&= \sum_{l=1}^{n-1} l\cdot q_0q_1\cdots q_{l-1}p_l + 1 \\
&= \sum_{l=1}^{n-1} l\cdot q_0q_1\cdots q_{l-1} - \sum_{l=1}^{n-1} l\cdot q_0q_1\cdots q_{l-1}q_l + 1 \\
&= \sum_{l=1}^{n-1} q_0q_1\cdots q_{l-1} - (n-1)q_0q_1\cdots q_{n-1} + 1 \\
&\leq \sum_{l=1}^{n-1} q_0q_1\cdots q_{l-1} + 1 \leq \frac{n-1}{n}\sum_{l=1}^{n-1} q^{l-1}\left(\frac{n-1}{n-2}\right)^{l-1} + 1 \\
&\leq \frac{n-1}{n}\sum_{l=1}^{\infty} q^{l-1}\left(\frac{n-1}{n-2}\right)^{l-1} + 1 = \frac{(n-1)(n-2)}{n(np-p-1)} + 1 < \frac{1}{p - p\frac{1}{n} - \frac{1}{n}} + 1 \\
&= \frac{1}{p} + \frac{1}{p}\frac{p+1}{pn-(p+1)} + 1.
\end{aligned}$$

As $p = \omega(1/n)$ the second term is $o(\frac{1}{p})$, from which the theorem follows. ■

Observe that if $p = \omega(\frac{1}{\sqrt{n}})$ then $\mathbf{E}[\#\text{ of steps}] = \frac{1}{p} + o(1) + 1$. Also, it would be interesting to consider whether this bound also holds for the adversarial model of faulty nodes.

As we have proved, no fixed memory deterministic algorithm has a constant expectation in the number of steps required to locate the information node. However, the unlimited memory randomized algorithm we described in this section can be converted easily into an unlimited memory deterministic one by only changing line 12 to be as follows:

current = the lexicographically smallest node in $V - M$.

This change does not affect the analysis of the randomized algorithm because the probability that the chosen node is faulty or not, or that it is the information node, is the same as if the next node had been chosen at random.

7 CONCLUSIONS

In this paper we have considered the problem of searching for an information node in the complete network of n nodes using information about its location obtained from the nodes of the network, where the information is correct with some bounded probability $p = \omega(1/n)$. This search problem was introduced in [15] for the ring and toroidal interconnection networks. One possible direction for future research is to consider searching with uncertainty on classes of graphs more general than the cliques. One complication that immediately arises is that a node giving correct advice does not point to the information node directly, since it may not be adjacent to it, but it points to a node that lies on a shortest path to that node (see [15]). On another front, it would be interesting to derive lower bounds on the expected number of steps for more general graphs. However, perhaps it is now not possible to apply the random walk argument that we applied to the cliques. Recall that the proof of the $\frac{1}{p}$ lower bound was based on the analysis of the expected number of steps required by a random walk on the complete network before *hitting* a randomly chosen set of nodes that give *correct* advice. Following a similar line of thought for other classes of graphs seems to be difficult in view of the discussion in [17] about the analysis of the expected hitting times in random walks on graphs. In [17] it is stated that hitting (or access) times can remain bounded, i.e. independent of the number of nodes of the graph, even for regular graphs. In the same paper, it is also stated that the only graphs for which a nonconstant lower bound on the expected hitting time can be proved are the graphs with *transitive* automorphism group. Therefore, deriving lower bounds for the problem of searching with uncertainty on more general classes of graphs may require more complex arguments. Also, proving a lower bound with dominating term less than $\frac{4}{p}$ for all memoryless, randomized algorithms or finding one with a better complexity figure is an interesting question that we believe merits further investigation. Finally, it would also be worth considering whether Theorem 7 also holds for the adversarial model.

Acknowledgments

We would like to acknowledge the helpful comments and suggestions of the anonymous referees that helped us improve the presentation of our work substantially and simplify many of our derivations.

REFERENCES

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

[2] S. Albers and M. Henzinger, Exploring unknown environments, Proc 29th Symposium on Theory of Computing, 1999, pp. 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] R. Cole, B. Maggs, and R. Sitaraman, Routing on butterfly networks with random faults, Proc 36th Symposium on Foundations of Computer Science, 1995, pp. 558–570.

[7] X. Deng and C. Papadimitriou, Exploring an unknown graph, Proc 31st Symposium on Foundations of Computer Science, 1990, pp. 356–361.

[8] S. Dolev, E. Kranakis, D. Krizanc, and D. Peleg, Bubbles: Adaptive routing scheme for high-speed networks, SIAM Journal on Computing 29(3) (1999), 804–833.

[9] W. Evans and N. Pippenger, Lower bounds for noisy boolean decision trees, Proc 28th Symposium on Theory of Computing, 1996, pp. 620–628.

[10] U. Fiege, D. Peleg, P. Raghavan, and E. Upfal. Computing with, uncertainty, Proc 22nd Symposium on Theory of Computing, 1990, pp. 128–137.

[11] R. Sedgewick and P. Flajolet, An Introduction to the Analysis of Algorithms, Addison-Wesley, 1996.

[12] A.C Kaporis, L.M. Kirousis, E. Kranakis, D. Krizanc, Y.C. Stamatiou, and E.C. Stavropoulos, Locating information with uncertainty in fully interconnected networks with applications to World Wide Web information retrieval, The Computer Journal 44(4) (2001), 221–229.

[13] S. Karlin and H.T. Taylor, A first course in Stochastic Processes, Second Edition, Academic Press, 1975.

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

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

[16] T. Leighton and B. Maggs, Expanders might be practical, Proc 30th Symposium on Foundations of Computer Science, 1989, pp. 384–389.

[17] L. Lovász, Random Walks on Graphs: A Survey, Combinatorics 2 (1993), 1–46.

[18] 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.

[19] P. Panaite and A. Pelc, Exploring unknown undirected graphs, Proc 9th Symposium on Discrete Algorithms, 1998, pp. 316–322.

[20] C.H. Papadimitriou and M. Yannakakis, Shortest paths without a map, Theoretical Computer Science 84(1) (1991), 127–150.

[21] N. Pippenger, On networks of noisy gates. Proc 26th Symposium on Foundations of Computer Science, 1985, pp. 30–36.

[22] P. Raghavan, Robust algorithms for packet routing in a mesh, Proc 1st Symposium on Parallel Algorithms and Architectures, 1989, pp. 344–350.

[23] T.E.S. Raghavan, T.S. Ferguson, T. Parthasapathy, and O.J. Vrieze (Editors), Stochastic games and related topics, Kluwer Academic Publishers, 1991.

[24] W.H. Ruckle, Geometric games and their applications, Research Notes in Mathematics, Pitman Publishing Inc., 1983.