Searching with Uncertainty
(Extended Abstract)

Evangelos Kranakis*   Danny Krizanc*

Abstract

We consider the problem of searching for an item in a distributed network in the presence of uncertainty. Although the location of the item in the network is 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. This information is correct with some bounded probability. In this paper we give algorithms for searching in a distributed network that has the topology of a ring or torus under various models.

Key Words and Phrases: Ring, Torus, Routing Tables, Search, Uncertainty.

1 Introduction

A friend tells you about a great restaurant in mid-town Manhattan. She remembers the name but she doesn't remember where it is located. At every intersection, there is a policeman with extensive but imprecise information about the location of restaurants in New York. You approach the policeman at the nearest intersection and ask him where the restaurant is. He tells you "I'm ninety per cent sure it is north of here." You travel one block north and

*Carleton University, School of Computer Science, Ottawa, ON, K1S 5B6, Canada, email: {kranakis,krizanc}@scs.carleton.ca. 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).

the policeman there tells you he remembers it to be east of that intersection. The next policeman tells you to go south. What should you do now? It appears at least one of the policeman is wrong.

This example suggests the problem of searching under uncertainty in distributed networks. We assume that the users of a network are interested in locating items available at different nodes in the network. (The items might be, for example, information, services, etc.) They do this by sending agents out to search for the desired item. We assume further that each of the nodes of the network maintains a database indicating the first edge on a shortest path to the items of interest. I.e., the database is routing database in which nodes may have more the one name or no name corresponding to the items available at the node. Finally, we assume that inaccuracies occur in the database for reasons such as the movement of items, out-of-date information, etc. The problem is how to use the inaccurate databases to find the desired item.

In order to make the problem more precise we must be more precise about how we model the uncertainty. We consider two possible models of uncertainty below. Other possible models are discussed in the conclusions.

A network of n nodes is represented as a connected undirected graph $G = (V, E)$ with V its set of nodes and E its set of links. The universe of items of interest is S. Let $Adj(x)$ be the set of links adjacent to node x in the network. For $x \in V$ and $v \in S$ define $S_v(x)$ to be the set of links adjacent to x which are on a shortest path from x to the node containing v. Clearly, $\emptyset \subset S_v(x) \subseteq Adj(x)$, for all x, v.

Let $1 \geq p > \frac{1}{2}$ be constant. We consider two models of uncertainty.

Single Shortest Path (SSP) model: Assume for each item $v \in S$ a tree representing a single shortest path from each node in V to the node containing v has been defined. If $e \in S_v(x)$ is the edge chosen in this shortest path tree then e appears in the entry for v in the database of x with probability greater than or equal to p. The probability the entry is some other $e' \in Adj(x) \setminus \{e\}$ is less than or equal to $1 - p$. Furthermore, for each v and for each x and x' these events are independent, i.e., an error occurs in x's entry for v with probability at most $1 - p$ independent of errors occuring in the v entry of any other nodes, where by an error we mean that the node reports an edge other than the shortest path tree edge when queried about v. Note the "error" may in fact be an edge on a different shortest path to v and therefore may not be an error at all.

We further distinguish a submodel of the SSP model which we call SSP with global information. In this model, some global information about how the shortest path trees were originally constructed is available to the search algorithm. For example, in our discussion of the torus below, we assume that the original databases, before the introduction of errors, were constructed using the standard row column shortest path trees and this is used in the construction of the searching algorithm.

All Shortest Paths (ASP) model: In this model, the probability that the entry for v in the routing table of x is some $e \in S_v(x)$ is greater than or equal to p and the probability it is some $e \in Adj(x) \setminus S_v(x)$ is less or equal to $1-p$. Again errors at distinct nodes (in this case reporting an edge in $Adj(x) \setminus S_v(x)$) occur independently. We note that for any geodesic (unique shortest path) network the two models are the same.

In this paper we consider searching with uncertainty under the above models for the ring and the torus network. In both cases we assume that the topology of the network as well as the standard global orientation (or sense of direction) is available for constructing our searching algorithms. Furthermore, we assume the bound p is known. The goal is to design a searching algorithm that an agent can run in order to find a desired item. The cost of the algorithm is the number of nodes visited by the agent. In our algorithms we assume that agents can store a constant number of words of information and besides making queries can make minimal use of the computational resources of the nodes it visits.

For the n node ring we give a searching algorithm in the SSP or ASP model which with high probability finds a given item in $d + O(\log n)$ steps where d is the distance from the starting node to the node containing the desired item. (By "with high probability" we mean with probability greater than $1 - n^{-O(1)}$.) Note that for $d = \omega(\log n)$ this is optimal to within an additive term. For the $n \times n$ torus we give a searching algorithm in the SSP (with global information) model which with high probability finds a given item in $d + O(\log^2 n)$ steps. Clearly this is optimal to within an additive term for $d = \omega(\log^2 n)$. Finally, for the ASP model we give an algorithm which with high probability finds an item in $O(d \log n)$ steps.

To the best of our knowledge these are the first results to be reported in a model of searching as described above. We note that a significant amount of research has been done on routing in networks with random or malicious faults [1, 3, 6, 8, 14, 18] but this work does not appear to capture the search problem considered here. In terms of the restaurant finding problem, routing under faults corresponds to the problem where the exact address of the restaurant is known and one wishes avoid roadblocks or traffic jams. Other related lines of research including searching for a fugitive in a graph [4, 12, 15] or exploring an unknown graph [2, 7, 16]. In neither case do the nodes contain information of the type considered here that might direct the search. In the restaurant analogy, there are no informed policeman. We feel our work is closer in spirit to that of computing with uncertainty or with noisy computing elements [9, 10, 13, 17]. Seen in this light the results are relatively simple to derive (with the possible exception of the last result). In the conclusions, we discuss a number of variations on the models developed in the paper which we feel should lead to many interesting avenues of research.

2 Ring

In this section we consider search algorithms on the ring. Note that the odd ring is geodesic and that in the even ring the shortest paths are unique except for nodes at distance $\frac{n}{2}$. For this reason the searching problem is essentially equivalent in the two models so we consider only the ASP model below. We assume that the ring is oriented in a consistent manner and this orientation is known to all processors. In this orientation each node has its left (respectively, right) link labeled Left (respectively, Right). Furthermore, we assume the size of the ring is known.

Clearly, on the ring a searching algorithm could completely ignore the advice of the databases and arbitrarily choose a direction (say Left) and move around the ring in that direction until the item is found. In the worst case, this algorithm requires $n-d$ steps where $d \leq \frac{n}{2}$ is the actual distance to the item. Slightly better might be to consult the database of the initial node and use that direction to search for the item (ignoring all later advice). In this case, for any item the expected number of steps is $pd+(1-p)(n-d) = (1-p)n$. Better still: take the majority of the advice of three nodes. This leads directly to the following algorithm parameterized by k: Search in a given direction for k steps. Maintain a count of the advice given by the routing tables of the k processors along the way and make a decision based on the principle of maximum likelihood either to stay the course or reverse direction until the desired item is found. More precisely, we have the following algorithm.

SearchRing(k) :

1. Let dir be the direction given by the initial node.
2. **for** $j = 1$ **to** k or until item is found **do**
3. move in direction dir
4. **if** majority of processors agree with dir,
   continue until desired item is found
   else reverse direction and continue until item is found

Thus, the search proceeds for k steps and then the direction is chosen according to the principle of maximum likelihood. This means that it selects the direction agreed by the majority of nodes along this path of length k. The probability that this direction is correct is equal to p_k, where

$$p_k = \sum_{i \geq \lceil k/2 \rceil} \binom{k}{i} p^i (1-p)^{k-i}.$$

Let d be the distance of the initial node to the node containing the desired item, and let X the number of steps traversed by the algorithm. It is clear from the algorithm that if the direction decision made by the algorithm is correct then $X = d$ or $X = d+2k$ depending on whether or not the algorithm reversed direction after k steps. Similarly, if the decision made was incorrect then $X = n-d$ or $X = n-d+2k$ depending on whether or not the algorithm reversed direction after k steps. It follows that that with probability at least p_k the number of steps traversed by the algorithm does not exceed $d + 2k$.

Using Chernoff-Hoeffding bounds [5, 11] setting $\mu = kp$ and $\delta = 1 - \frac{1}{2p}$ we observe that

$$\begin{aligned} p_k &= \Pr[S_k \geq k/2] \\ &= \Pr[S_k \geq (1-\delta)\mu] \\ &= 1 - \Pr[S_k < (1-\delta)\mu] \\ &> 1 - e^{-\mu\delta^2/2}, \end{aligned}$$

where S_k is the sum of k independent and identically distributed random variables $X_1, \ldots, X_k$ such that $\Pr[X_i = 1] = p$.

Choosing

$$k \geq c \frac{2p}{(p-1/2)^2} \ln n,$$

we conclude the probability that the algorithm requires less than $d+2k$ steps is $\geq 1 - n^{-c}$ for $c > 0$. Therefore we have proved the following theorem.

Theorem 2.1 *If d is the distance of the initial node to the node containing the item of interest then, for $k \in \Theta(\log n)$, with probability at least $1 - n^{-O(1)}$ the algorithm SearchRing(k) finds the item in $d + O(\log n)$ steps.* ∎

We observe that the above algorithm is optimal to within an additive term for $d = \omega(\log n)$. Further we note that for smaller d it is possible to make the algorithm optimal to within a multiplicative term by making the initial search in both directions out to distances $1, 2, 4, \ldots, 2^i$, where $i = \lceil \log k \rceil$.

3 Torus

In this section we consider search algorithms on the torus. First we observe that by ignoring the databases of the nodes and searching in a spiral pattern around the start node an item will be found in $O(d^2)$ steps where d is the actual distance from the start node to the node containing the desired item. Note that this algorithm works in either of the models described above.

We again assume the size of the torus is known and a global orientation of the edges is given, i.e., each node of the torus has its four links labeled by Up, Down, Left, Right (abbreviated by U, D, L, R, respectively). We also use the notation $\rightarrow \leftarrow, \uparrow, \downarrow$ for the edges reported in the routing tables.

3.1 SSP with global information model

In this section we further assume that the initial values of the databases (before errors were introduced) were derived using the row then column routing rule, i.e., the shortest path tree used in the definition of the SSP model consists of row edges directed to the column followed by column edges directed to the root node.

Algorithm SearchTorus[SSP]

The main steps of the algorithm are as follows.

1. **Deciding Direction:** For $i = 1$ to $O(\log n)$ move right along row of the initial node. Select direction using principle of maximum likelihood, i.e. the majority among directions $\rightarrow, \leftarrow$ (ignoring $\uparrow, \downarrow$ edges).

2. **Search along chosen direction:** Move in the direction chosen in step 1 above. For each successive row block B of $O(\log n)$ nodes, count the number of nodes pointing $\rightarrow$ and $\leftarrow$, until two adjacent blocks, say B, B' are found such that the majority of nodes in B point $\rightarrow$ and the majority of nodes in B' point $\leftarrow$.

3. **Deciding on a vertical column:** Let the blocks B, B' lie between nodes $(i,j), (i',j)$ such that $k := |i - i'| = O(\log n)$. Consider the square centered at the point $((i+i')/2, j)$ with side length equal to $2k$. Search all the nodes of this square and select a column such that a majority of the nodes point $\uparrow$ or point $\downarrow$.

4. **Search along a column:** Move along the selected column in any direction for $k = O(\log n)$ steps. Select direction according to the principle of maximum likelihood. Traverse the ring, corresponding to this column of the torus, moving in the chosen direction until information is found.

5. **Search along a spiral:** If information has not been found after the execution of steps 1 through 4 then search for the information by tracing counter-clockwise a spiral around the node.

The analysis of this algorithm is similar to that of algorithm SearchRing for the ring. Using Chernoff bounds we can show that the decision taken by the algorithm to search in the proposed direction in any of steps 1 through 4 is wrong with probability at most $n^{-O(1)}$. Moreover, with probability at least $1 - n^{-O(1)}$ the distance traveled in steps 1 through 4 does not exceed $d + O(\log^2 n)$, where d is the distance of the initial node to the item. The item may not be located in the first four steps of the algorithm, in which case it will be found in step 5 by travelling a distance $O(d^2)$. However, the probability that step 5 will be executed does not exceed $n^{-O(1)}$. Therefore we have outlined the proof of the following theorem.

Theorem 3.1 *In the SSP model with global information, with probability at least $1 - n^{-O(1)}$, algorithm SerachTorus[SSP] finds any item in $d + O(\log^2 n)$*

steps, where d is the distance of the starting node to the node containing the desired item. ■

3.2 ASP model

In the ASP model at each node the routing table indicates the correct direction is Up (respectively, Down, Left, Right) with (some unknown) probability p_U (repsectively, p_D, p_L, p_R) such that $p_U + p_D + p_L + p_R = 1$. There are four possible pairs of adjacent links, namely $(U, R), (R, D), (D, L), (L, U)$. Clearly, there is such a pair (A, B) and there can exist at most two pairs (A, B) such that the sum $p_A + p_B$ of their respective probabilities is $> 1/2$. Moreover, it is easy to see that in the second case these two pairs must be adjacent (if not, then we would have that two non-adjacent pairs, say $(U, R), (R, D)$, such that $p_U + p_R > 1/2$ and $p_R + p_D > 1/2$ which contradicts the fact that $p_U + p_D + p_L + p_R = 1$).

To sum up, we have that exactly one of the following two possibilities occurs:

1. there is a unique pair, say (A, B), such that $p_A + p_B > 1/2$ where $(A, B) = (R, U)$),
2. there are exactly two pairs, $(A, B), (B, C)$ such that $p_A + p_B > 1/2$ and $p_B + p_C > 1/2$ where $(A, B) = (R, U)$ and $(B, C) = (R, D)$).

Algorithm SearchTorus[ASP]

We are now in a position to state the main algorithm under the ASP model. We use the same conventions on notation as in algorithm SearchTorus[SSP]. First we describe the following subroutine which makes a search within a subtorus centered at u.

> **Search (u):** Processor u searches for the desired item by testing all nodes within a square centered at node u with side length equal to $2k$, where $k := O(\log n)$. Compute for each adjacent pair of directions the sum of those nodes recommending either of the directions. If exactly one pair represents the majority of all recommendations then travel in the direction of one of these directions (say, the one with the most recommendations) until the boundary of the square is reached. If there

are two pairs that represent the majority then travel in the direction of the directions shared by the pairs until the boundary of the square is reached.

Using the subroutine **Search** (u) the main algorithm can be described as follows.

1. Use subroutine **Search (u)** to search within a square centered at u and select a new node u'. We replace u with u' and iterate.

2. **Stopping the iterartions:** We continue iterating until either the item is found or we find two adjacent squares, say S, S', such that searching in S yields a recommendation $\rightarrow$ and searching in S' yields $\leftarrow$ (or $\uparrow, \downarrow$ respectively).

3. **Search along a block of columns (rows):** In squares of size k compute the majority of nodes in the up/down (left/right) direction, move in that direction to the edge of the square and iterate.

4. **Search along a spiral:** If item has not been found after the execution of steps 1 through 3 then search for the item by tracing counter-clockwise a spiral around the initial node.

Using similar arguments to the above we can show that each of iterations of the algorithm is correct with probability at least $1 - n^{O(1)}$. Since each iteration takes $O(\log^2 n)$ steps and if correct moves you $\Omega(\log n)$ steps closer to the item and the number of iterations is bounded by $O(n)$ we can show the following theorem.

Theorem 3.2 *In the ASP model, with probability at least $1 - n^{-O(1)}$, the algorithm $SearchTorus[ASP]$, does not exceed $O(d \log n)$ steps when searching for any item, where d is the distance of the initial node to the node containing the desired item.* ∎

4 Conclusions and open problems

We have introduced a new model for the problem of searching under uncertainty in networks and provided solutions to the problem for the case of the ring and the torus network. A number of possible lines of research are suggested by this work.

Directly arising from the results described here is the question of whether the algorithm $SearchTorus[ASP]$ is optimal for the ASP model. It may be possible to use lower bound techniques for problems of computing with uncertainty to derive a lower bound for this problem. Another obvious question is to consider bounds for other networks or for general networks in the models suggested in this paper. The algorithms described here are deterministic. What about randomized algorithms? For example, one might consider an algorithm that follows a random walk that is biased in some way by the suggestions given by the node databases.

Other models suggest themselves. The networks we considered were node symmetric and our algorithms in the ASP model used only information about the size of the network and the topology. In other networks, knowing the topology can make a big difference. For example, in a chain network, if we assume the nodes know their position, then nodes near the ends have an advantage when searching. The impact of this knowledge on the model might be interesting to explore. Our algorithms implicitly assume that besides querying a nodes database, agents can store $O(1)$ counters and can use a small amount of the computational resources of a node. One might consider the impact of restricting the power of searching algorithms. The above model assumes the errors are probabilistic. One could consider a liars model where an upper bound on the number of malicious nodes is given. For example, it is easy to see that $d + \Theta(k)$ steps are sufficient (and necessary) in such a model on the ring. What about the torus? The above assumes that items occur once in the network. In many cases the items may be replicated and one is interested in finding the closest incarnation of the item. This might be called the MacDonald's problem. You are looking for a MacDonald's in midtown Manhattan but don't care which one, just the closest. In our model, the independence assumption seems pretty unrealistic. One could consider models where errors had some limited dependence and/or had some dependence on the distance to the item, e.g., databases close to an item are more likely to be correct.

References

[1] Y. Afek, E. Gafni, and M. Ricklin, “Upper and lower bounds for routing schemes in dynamic networks,” FOCS 89, 370-375.

[2] S. Albers and M. Henzinger, “Exploring unknown enviroments,” STOC 97, 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] D. Bienstock and P. Seymour, “Monotonicity in graph searching,” *J. of Alg.*, **12** (1991), 239-245.

[5] H. Chernoff, “A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations,” *Annals of Mathematical Statistics*, **23** (1952), 493-509.

[6] R. Cole, B. Maggs and R. Sitaraman, “Routing on butterfly networks with random faults,” FOCS 95, 558-570.

[7] X. Deng and C. Papadimitriou, “Exploring an unknown graph,” FOCS 90, 356-361.

[8] D. Dolev, E. Kranakis, D. Krizanc and D. Peleg, “Bubbles: Adaptive routing scheme for high-speed dynamic networks,” STOC 95, 528-537.

[9] W. Evans and N. Pippenger, “Lower bounds for noisy boolean decision trees,” STOC 96, 620-628.

[10] U. Fiege, D. Peleg, P. Raghavan and E. Upfal, “Computing with unreliable information,” STOC 90, 128-137.

[11] W. Hoeffding, “On the distribution of the number of successes in independent trials,” *Annals of Mathematical Statistics*, **27** (1956), 713-721.

[12] L. Kirousis and C. Papadimitriou, “Interval graphs and searching,” *Discrete Mathematics*, **55** (1985), 181-184.

[13] E. Kushilevitz and Y. Mansour, "Computation in noisy radio networks," SODA 98, 236-243.

[14] T. Leighton and B. Maggs, "Expanders might be practical: Fast algorithms for routing around faults in multibutterflies," FOCS 89, 384-389.

[15] N. Megiddo, S. Hakimi, M. Garey, D. Johnson and C. Papadimitriou, "The complexity of searching a graph," *JACM*, **35** (1988), 18-44.

[16] P. Panaite and A. Pelc, "Exploring unknown undirected graphs," SODA 98, 316-322.

[17] N. Pippenger, "On networks of noisy gates," FOCS 85, 30-36.

[18] P. Raghavan, "Robust algorithms for packet routing in a mesh," SPAA 89, 344-350.