

Detecting and cleaning intruders in sensor networks

Poonna Yospanya, Bundit Laekhanukit, Danupon Nanongkai, Jittat Fakcharoenphol
Department of Computer Engineering,
Kasetsart University,
Bangkok, 10900 Thailand.

E-mails: {poonna@eng.src,b4205092@,danupon@eng.src,jtf@}ku.ac.th

Abstract

We view the problem of detecting and cleaning intruders in sensor networks using mobile agents as a version of the graph searching problem. The goal is to minimize the number of agents running at the same time. Three scenarios are considered, each differs in the relative power of the agents and the intruders. Our main idea is to use breadth-first-search (BFS) trees to organize the search. In the case where the intruder is most powerful, we search the graph by levels of the nodes on the BFS tree. However, in the second case where the network is configurable, the number of agents could be improved significantly if a good sub-graph can be found. This motivates us to define the Minimum Search Number Spanning Tree problem, of which we also prove its hardness. We however show that one can still use a BFS tree to get a good result. In the last scenario where the intruder has no information on the status of the agents, random walks are used. In each case, we prove upper bounds on the number of agents and provide experiment results.

1 Introduction

The networks of small but numerous sensors with wireless communication prove to be very useful in remote sensing tasks (see, for example, [17, 13, 6]). Their unique characters are the sources of many challenges [5, 7]. Very often, as in military examples, they are placed in hostile environment, a result of both physical factors, such as heat, wind, and radio activity, and also logical factors, i.e., software intruders, which are the focus of this paper.

If the sensor nodes are powerful enough, it is possible to equip them with management software, e.g., system diagnosing or intrusion detection system. However, the resource, especially the memory resource, is very limited in each node. Storing these huge, infrequently used programs might cause the nodes to be unable to perform their main sensing task due to memory shortage. Thus, this crucial

managing capability is provided through mobile software *agents*. The agent is a program that can migrate itself on the network. When residing on some sensor node, it takes care of various system management tasks. However, it also takes large amount of computational resource of the node, interrupting the node's primary work. Thus, simply flooding the entire network with agents would stop the whole network from functioning.

The problem addressed in this paper is how to organize a small group of agents to track down an *intruder*, which is a kind of malicious software agent¹. We discuss the problem of *detecting* and *cleaning* the intruder. We note that one way to detect the intruder is by actually cleaning the network. This problem is essentially a well-studied problem, called *graph searching* [3, 16], in graph theory. Furthermore, the requirement that the agents only move along the edge of the communication graph results in the *contagious* version of the problem, defined by Barrière, Floccini, Fraigniaud, and Santoro [1] who also give a distributed algorithm that computes the search strategy when the graph is a tree. They leave the cases of other families of graphs as the open problem. This paper can be considered as an attempt to give partial answers to the problem for communication graphs induced by typical sensor networks.

We study three versions of the problem, each with different agent-intruder relative power. In all cases, the intruder is infinitely faster than the agents. The first case is when the intruder is adversarial, and the agents have no control over the network. The second is when the network is dynamically configurable, so that it is possible to “constrain” the intruder (and agents) to move only on selected edges. In these first two cases, the fast intruder knows the status of the network and the agents, and utilizes this knowledge to move, or evade the agents accordingly. Therefore, to detect the intruder, one essentially has to clean the entire network, and the problem of detecting and clean-

¹From here on, we will refer to the agents doing the detecting and cleaning tasks simply as agents, and the intruding agent as intruder.

ing become the same problem. The third case is when the intruder does not know the states of the agents. It can move very fast, but cannot plan its action based on the agents’s choices. Only in this case, we provide another detection algorithm based on random walks on graphs.

1.1 The problem

The problem addressed in this paper is how to organize a small group of agents to track down a faster intruder. We discuss the problem of *detecting* and *cleaning* the intruder. We show that one way to detect the intruder is by actually cleaning the network. This problem is essentially a well-studied problem, called *graph searching* in graph theory.

In the graph-searching problem, introduced by Breisch [3] and by Parson [16], we consider the following situation. A group of agents move along the edges of the graph to find an intruder, who moves along the edges of the graph infinitely faster and also has a complete knowledge of the status of the agents. Given a graph G , the *graph-searching problem* is to determine the minimum number k such that there is a search strategy for k searchers that guarantees the capture of the intruder. The minimum number k is called the *search number* of the graph G .

The variant of the graph-searching problem we consider in this paper is the *contagious graph search*, defined by [1]. The problem can be formulated as follows.

We are given a graph G whose edges are “contaminated.” A group of searchers are placed on the nodes of G . At any step, we can move a searcher along an edge. An edge $\{u, v\}$ becomes temporarily “cleaned” if a searcher moves from u to v . However it only remains cleaned if and only if (1) there is another searcher at u or (2) all edges adjacent to u are cleaned. Another requirement for the contagious graph search is that the set of cleaned edges must form a connected subgraph.

1.2 Related results

Megiddo, Hakimi, Garey, Johnson, and Papadimitriou [14] prove that determining the search number is NP-Hard. They also give a linear time algorithm to find the search number on trees. The problem has many variants which are related to concepts in graph theory (see, for examples, [10, 9, 2, 19]).

The problem has been formulated as a pursuer-evader problem in Demirbas, Arora, and Gouda [4] where a single faster agent tries to track a single slower adversarial intruder. We, however, focus on the opposite case, i.e., when the intruder is infinitely faster, but many agents are allowed.

1.3 Organization

We give a formal description of the model and a summary of results in Section 2. This paper focuses on three scenarios. The first case when the intruder is adversarial and the agents have no control over the network is described in Section 3. Section 4 deals with the second where the network is configurable. Finally, when the intruder is oblivious, Section 5 gives a brief discussion on how to use random walks to find the intruders. The experiment results are presented in Section 6. We conclude and list a few interesting open problems in Section 7.

2 Preliminaries

2.1 The model

2.1.1 The network

We model the situation as follows. There are n *sensor nodes*, each with a unique id, locating in a circular area of diameter D . The locations of the nodes are uniformly distributed over the area². Among the nodes, a special node s is chosen to be the base station, the only node that the intruder cannot reside. We define the density of the nodes to be $\rho = \frac{n}{\pi D^2/4}$. This parameter is not entirely independent of D and r if we want to ensure connectivity, as will be discussed later on.

Each sensor node has a two-way broadcast communication capability with *maximum radius* r . The locations of the sensor nodes, together with the parameter r , induce a possible communication graph. Now we formalize this. We let $V = \{v_1, v_2, \dots, v_n\}$ denote the set of n sensor nodes. For each node v_i , denote its location with (x_i, y_i) . There is an edge (v_i, v_j) in the communication graph $G = (V, E)$ iff $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq r$. Since the distance function is symmetric, the graph is undirected.

We do not require only the connectivity of G but also the property that the distances between nodes in G approximates the actual distances in the plane, i.e., for any pair of nodes x and y whose distance on the plane is l , there is a path from x to y in G with $\Theta(l/r)$ edges. The lowerbound clearly holds. The following lemma states the sufficient condition for ρ that ensure this property with high probability.

Lemma 1 *If $\rho = \Omega(r^2 \log D/r)$, or equivalently $n = \Omega((D/r)^2 \log(D/r))$, the distances in the graph approximate the actual distances of the nodes in the plane.*

²This is not a strict requirement. We only need to be able to bound the expected number of nodes in a given area.

Proof: Cover the circular area with non-overlapping $(r/2\sqrt{2})$ by $(r/2\sqrt{2})$ squares whose diagonals are of length r . This can be done with $O((D/r)^2)$ squares. Now consider only the squares which lie entirely in the area; call them *full* squares. The rest are called *partial* squares. If there is at least one sensor node in each of the full squares, the graph is clearly connected. Furthermore, for any pair of nodes x and y whose distance is l , every full square intersecting a line segment from x to y contains at least one sensor; this implies that there is a path of length at most $O(l/r)$ connecting x and y .

We now think of a random process of placing sensors into all the squares. We claim that the probability that we pick full squares is at least a constant. There are $O(D^2/((r/2\sqrt{2}) \cdot (r/2\sqrt{2}))) = O((D/r)^2)$ full squares. We want to figure out the expected number of nodes to cover these squares. This is a coupon collector problem, and we have what we need, on expectation, $\Omega((D/r)^2 \log(D/r))$. The lemma thus follows. ■

We believe that this assumption is not an artificial one. Normally, sensor nodes are placed so that they can perform a sensing task. To be able to cover the entire area, the designer must ensure the coverage. The lowerbound on the density ρ if the coverage is required can be proved as well (see [8, 18]). For the study on the connectivity and coverage for sensor networks, see [15, 21].

2.1.2 Relative power of the network and the intruders

The *intruder* is a software agent that can migrate itself and (possibly) do harms on the network. In this paper, we allow the intruder to move infinitely faster than the agents.

We have three scenarios.

- *Adversarial intruder, general network.* In this scenario, not only the intruder is infinitely faster, it is also adversarial, i.e., it knows completely all the state of the network. It has the information on where the agents are and how they move.
- *Adversarial intruder, configurable network.* In this second scenario, we make the network more powerful. Each node can configure itself from which node it can communicate with. The problem turns out to be a network design problem, i.e., to find a subgraph having a good search strategy.
- *Oblivious intruder.* In the final scenario, we remove almost all power of the intruder. It can move as fast as the previous scenario, but it knows nothing about the agents.

Except for the last scenario, we also allow the intruder to duplicate itself, i.e., there can be more than one intruder at the same time.

2.2 The result

We show the following.

- In Section 3, we show that in the first scenario a simple algorithm using breadth-first search can clean the network using $O(rn/D)$ agents. We also prove that the number of agents is optimal for this case (up to a constant factor).
- Section 4 shows a simple search strategy on a spanning tree. This can be applied in the second scenario where the network is configurable. We show that a breadth-first-search tree can be used and prove that the number of agents needed is $O(D/r)$, independent of the number of sensor nodes.
- Finally, if the intruder has limited power, i.e., the case that the intruder does not know the status of the agents, we describe in Section 5 how to detect the intruder by random walks, the same approach that has been used in [4]. The analysis of that work uses the cover time of the random walk, because its goal is for a single agent to see the trace of the intruders once. We, however, do not require the intruder to leave any trace on the network. The property that we use is the near-uniform distribution of the agents, resulted from the memory loss property of a random walk.

We also provide experiment results for these algorithms.

3 BFS clean-up: adversarial intruder, general network

In this case, the intruder is most powerful. To clean the network, one needs to quarantine the intruder. We perform this task in a breadth-first-search fashion. The algorithm proceeds in iterations. In the first iteration, the agent, initially at s , migrates³ itself to all neighbors of s . Next iteration, each agent migrates to its uncleaned neighbors. The process continues until all nodes are cleaned (See Figure 1 for illustration).

We need to give a technical detail on how an agent migrates. When an agent at node v migrates to all nodes in set S , it first duplicates itself to nodes in S , activates them, and deletes itself from v . It must guarantee that the copies in S are running before deleting itself to prevent the intruder from migrating back to v .

³We allow a single agent to migrate to more than one nodes by means of duplication.

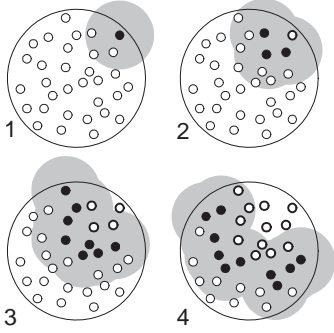


Figure 1: How the search progresses in BFS levels.

Theorem 1 *The above algorithm cleans the network, and the expected number of agents running simultaneously is $O(rn/D)$.*

Proof: We note that each level of a BFS tree is a node cut of the graph. Specifically, there is no edge adjacent to nodes at level i and nodes at level $i + 2$. This observation implies the correctness.

We consider now the number of agents. On iteration i , the agents reside only in the nodes whose distances from s are between $(i - 1)r$ and ir . The area of this strip of width r is at most $(2\pi ir) \cdot r$. The maximum is obtained when $i = O(D/r)$, the maximum number of iterations. This gives the upperbound of $O(Dr)$ on the area. Hence, the expected number of agents is $O(Dr \cdot \rho) = O(rn/D)$. ■

It can also be proved that the bound of $O(rn/D)$ is the best one could hope for in this case.

Lemma 2 *No deterministic algorithm performs better than $O(rn/D)$.*

Proof: Again cover the circular area with squares whose diagonals are of length r . At any step t of an algorithm, we define the square to be *cleaned* if all of its nodes are cleaned. For any algorithm that performs better than $O(rn/D)$, each new cleaned square needs agents for all of its nodes. The number of new squares must be bounded by $O(D/r)$ in order to maintain the expected number of agents at $O(\frac{D}{r} \cdot r^2 \cdot \rho) = O(rn/D)$. Hence, there exists the step t such that the number of cleaned squares is within $[(D/r)^2/2 - O(D/r), (D/r)^2/2] \subset \Theta((D/r)^2)$.

We called the cleaned square *risky* if it is adjacent to the uncleaned square. There must be an agent residing at each node of the risky squares. We will show that if there are k cleaned squares, the number of risky squares is $\Omega(\sqrt{k})$. Given a fixed number of cleaned squares, we can obtain a configuration that contains least number of risky squares when the

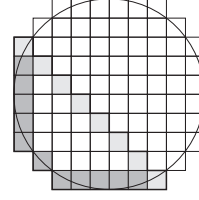


Figure 2: Configuration with smallest proportion of risky squares to peripheral squares. Peripheral squares are shown in gray, with lighter ones being risky squares.

cleaned squares form a connected shape and are adjacent to the boundary of the network area. Let the *peripheral* squares be the squares on the border of the shape, i.e., all the risky squares and the cleaned squares adjacent to the network boundary. The number of risky squares can be proved to be at least proportional to the number of the peripheral squares (the proportion is minimum at $\Omega(1)$, when the risky squares form a cord of the circular area, as illustrated in Figure 2). Let s denote the number of cleaned squares, and s is not more than half of the number of all squares, the minimum number of peripheral squares is bounded to $\Omega(\sqrt{s})$ by forming circle-like shape. Our argument thus follows.

Now we prove the theorem. At some step of the algorithm, there are $\Theta((D/r)^2)$ cleaned squares, which require $\Omega(\sqrt{(D/r)^2}) = \Omega(D/r)$ agents. It follows that the expected number of agents is $\Omega(rn/D)$. ■

4 Cleaning along a tree: adversarial intruder, configurable network

In this section, we introduce a problem on graphs called Minimum Search Number Spanning Tree, and show that it is NP-Hard in general graph. Although in this paper we neither solve the problem nor find an approximation algorithm for the problem, we prove that a simple BFS tree gives a provable bound on the number of agents in Section 4.1 We also show the existence of a good tree under the density assumption, in Section 4.2.

Given a graph $G = (V, E)$, the *Minimum Search Number Spanning Tree* problem is to find a spanning tree of G that minimizes the contagious search number. The following proposition states that the problem is NP-Hard in general graphs.

Proposition 1 *The Minimum Search Number Spanning Tree is NP-Hard in general graphs.*

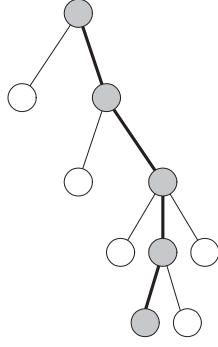


Figure 3: Agents remains at shaded nodes while doing a depth-first search along the tree.

Proof: We prove the hardness by a reduction to the Hamiltonian Path problem. Given a graph G , we want to find a spanning tree that minimizes the contagious search number. Note that only one agent is needed to clean a path. Therefore, if G contains a Hamiltonian path, that path must be the tree with minimum search number. Now, if one can solve the Minimum Search Number Spanning Tree, one can determine if a graph contains a Hamiltonian path. This completes our reduction. ■

4.1 Cleaning along a BFS tree

In this case, we have some control over how the nodes communicate. More specifically, we can enforce all the communications to take place only on the edges of a communication subgraph H of G . The goal is to find the subgraph that admits a good search strategy. We focus only in the case that the subgraph is a tree. This case has been studied by Barrière *et al.* [1] who show that given a communication tree, one can find the optimal contagious search strategy in linear time. However, to find the best tree seems to be difficult because the best possible tree is a Hamiltonian path, which is NP-Hard in general graphs.

In this paper, we cannot find the best tree or a tree which approximates it. However, we settle for a provable bound. The following lemma gives an upper-bound on the number of agents needed in a tree with some structure. A node in a tree is called a branching node if its degree is greater than two.

Lemma 3 Consider a tree T rooted at s . Let m be the maximum number of branching nodes along any path from s to any nodes in the tree. There is a search strategy which uses m agents.

Proof: (sketch) Consider running a depth-first search on the tree starting from s . At any branching node v ,

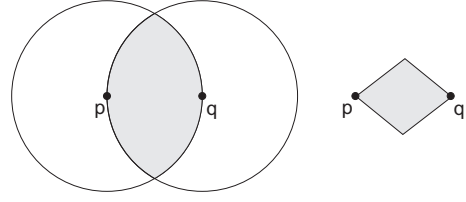


Figure 4: Illustration of lune and diamond

when we move further to visit its children we leave a copy of the agent with v (see Figure 3). This agent is deleted when the search backtracks. Clearly, there are at most m agents at any time. Furthermore, it is straight forward to show that any node remains cleaned after the search leaves it. ■

We have a simple corollary.

Corollary 1 A tree of depth d requires at most d agents.

Note that Lemma 1 implies that the depth of a breadth-first-search tree is $O(D/r)$. We have the following theorem.

Theorem 2 When the network is configurable, cleaning it along a breadth-first-search tree needs $O(D/r)$ agents.

4.2 Existence of a good tree

We note that if the sensor nodes are location-aware, a spanning tree with small search number can be constructed. In fact, only three agents are required. We briefly discuss how to find such a tree here. Consider, again, the filling of the area with $r/(2\sqrt{2})$ by $r/(2\sqrt{2})$ squares as in Lemma 1. First, we work on each row of full squares. All nodes in each row can be linked up as a path. We then join each pair of adjacent rows with some edge. Finally, we connect all the nodes in each partial square to some node in their adjacent full square. We note that these steps can be done if the nodes are location-aware. One can verify that in this tree, three agents are enough to clean it.

4.3 Cleaning networks with no assumption

Cleaning along a BFS tree also works on the network with no distribution assumption, i.e., the network is not required to have a minimum density. The number of agents is bounded by the depth of the tree.

Lemma 4 The depth of BFS tree on any sensor network is $O(D^2/r^2)$

Proof: We focus on path from root to any leaf. It can be proved in the same way as Lemma 10 in [20] that for every edge e we can determine a diamond of size $\frac{\sqrt{3}}{6}||e||^2$ which is disjoint from the diamonds of other edges (on the same path). To prove this, we claim that, on each path from root to a leaf, there are no pair of edges that cross each other, no node contained in the lune determined by any edge in that path, and the angle between two edges in different levels connecting the same node are not less than $\pi/3$. Because there are edges of length not less than $r/2$ in every other levels of the path, its depth is not more than $\frac{2\pi(d/2)^2}{\frac{\sqrt{3}}{6}(r/2)^2} = O(D^2/r^2)$. ■

5 Wandering agents: oblivious intruder

This section describe a algorithm for detecting intruders in the case of oblivious intruders. Oblivious intruders are those which can move infinitely faster than the agents, however, they do not have any information on the status of the set of the agents.

In fact, if we allow the agent to “jump” randomly to any nodes, a simple bound can be proven. I.e., if m agents are placed independently randomly on n nodes, the probability that the intruders escape the detection is at most

$$\left(\frac{n-1}{n}\right)^m = \left(1 - \frac{1}{n}\right)^m \approx e^{-m/n}.$$

To get this failure probability for this “one-shot” detection to be smaller than a constant, we need the number of agents to be a constant fraction of n . However, if we allow many rounds of detection, m could be made much smaller. Suppose we allow k iteration. We have that the failure probability is roughly

$$(e^{-m/n})^k = e^{-mk/n}.$$

Thus, if we require the failure probability to be less than δ , the number of agents we need is $O(n/k + \log \delta)$.

The problem we are left with is how to create this uniform distribution of agents which can only move along the network. The answer comes from the theory of Markov chains. We can have m agents, each randomly walks on the network, i.e., at any time step the agent picks one of its neighbor nodes to migrate to. If they walk long enough, the distribution of the locations of the agents converges to some fixed stationary distribution. It is well-known that on an undirected graph, the probability that an agent would end up at any given node is proportional to the node’s degree. In our case the expect degree of a node is proportional

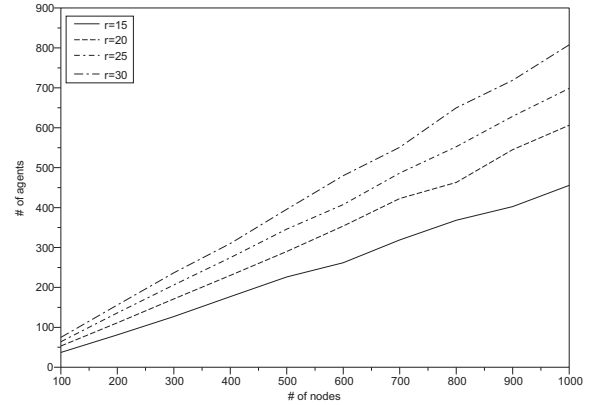


Figure 5: Cleaning in a BFS fashion

to the broadcasting range inside the area. Thus, in the case that $D \geq r$, the ratio between the smallest area and the largest area is at most $1/3$; this implies that after the agents walk long enough, the distribution gets close to uniform.

The expected number of steps needed to get close to uniform is called *mixing time* (see [12, 11], for example). Clearly, the cover time of a graph—the expected time for a random walk to visit all the nodes—can be used as an upperbound of the mixing time. It is known that the cover time of any graph is $O(mn)$; thus, the agents would find the intruders in polynomial time. A more sophisticated technique based on graph expansion can be used to give closer bounds.

6 Experiments

Section 6.1 reports the experiment results for the first two cases. Section 6.2 describes the result for the intruder detection using random walks.

6.1 First two algorithms: using a BFS tree to search

We performed experiments by placing 100,200,300 upto 1000 nodes on a disk of diameter 100 units. We varied the transmission radii to be 15,20,25 and 30 units.

In the first algorithm, we clean the network level-by-level on the breadth-first-search tree as in Section 6.1. The result, which is averaged over 100 trials, is shown in Figure 5. The number of agents needed is thus proportional to the number of nodes (as the radius fixed, the density grows with the number of nodes).

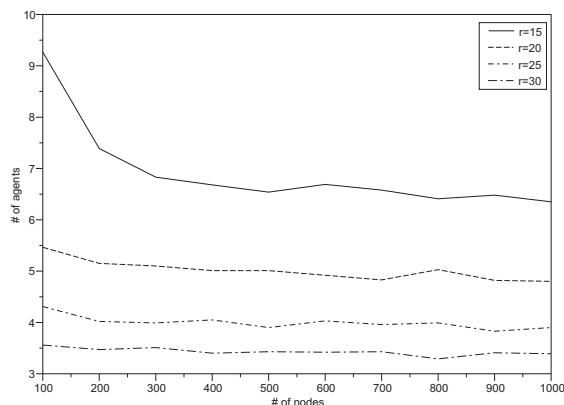


Figure 6: Cleaning on a BFS tree

When the network is configurable, we use a BFS tree to guide the search. Figure 6 plots the number of agents required in each case. We note that the number of agents required is independent of the number of nodes.

6.2 Catching oblivious intruders by wandering agents

We experimented with 500, 1000, and 1500 nodes, placed randomly on a disk of diameter 100 units. Here the metric is the number of search steps, since we fixed the number of agents on the network. Figure 7 and Figure 8 show the number of steps required for various transmission radii on the network of 1000 and 1500 nodes, respectively. Some cases experimented on 500-node networks were unsuccessful due to the relatively low density of nodes, causing the graph to be unconnected. The numbers plotted are average of 10 trials.

7 Conclusion and open problems

We present multi-agent algorithms for detecting and cleaning intruders on sensor networks. Our aim is to minimize the number of agents required at any given time.

Here are some open problems.

- Can we find an approximation algorithm to the Minimum Search Number Spanning Tree problem?
- Can we generalize results of [1] to general graphs?

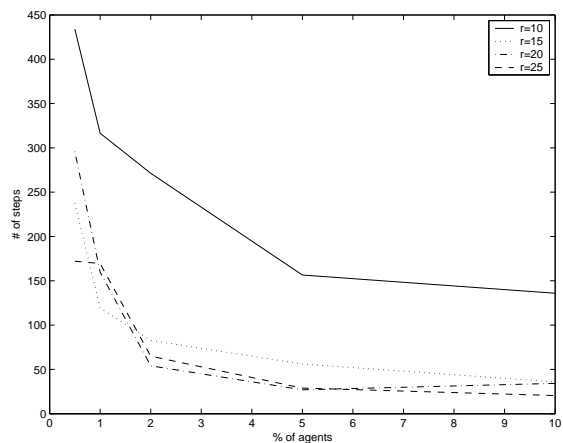


Figure 7: Wandering agents on 1000-node networks

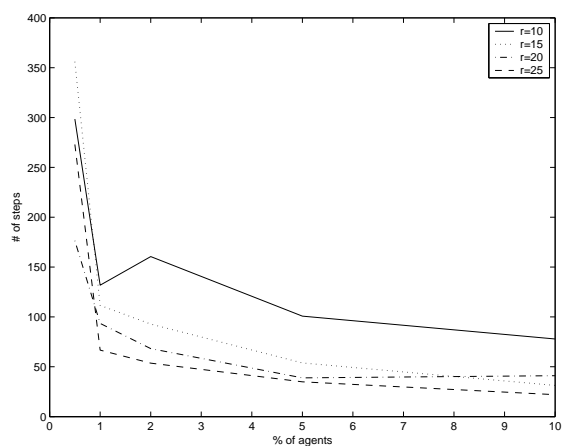


Figure 8: Wandering agents on 1500-node networks

- Given that the sensor nodes are not location-aware, can we find a better tree?

References

- [1] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209. ACM Press, 2002.
- [2] D. Bienstock and Paul Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.
- [3] R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, VI(5):72–78, 1967.

- [4] M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. In *Proceedings of the Sixth Symposium on Self-Stabilizing Systems*, 2003.
- [5] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270. ACM Press, 1999.
- [6] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebraNet. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 96–107. ACM Press, 2002.
- [7] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for “smart dust”. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278. ACM Press, 1999.
- [8] Bhaskar Krishnamachari, Stephen Wicker, Ramon Bejar, and Marc Pearlman. *Communications, Information and Network Security*, chapter Critical Density Thresholds in Distributed Wireless Networks. Kluwer Publishers, 2002.
- [9] Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.
- [10] Christos H. Papadimitriou Lefteris M. Kirousis. Searching and pebbling. *Theor. Comput. Sci*, 4(3):205–218, 1986.
- [11] László Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdős is Eighty, Vol.2*, pages 353–398, 1996.
- [12] László Lovász and Peter Winkler. Mixing of random walks and other diffusions on a graph. *Survey in Combinatorics*, 218:119–154, 1995. London Math. Soc. Lecture Notes Series.
- [13] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM Press, 2002.
- [14] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.
- [15] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proceedings of IEEE Infocom*, pages 1380–1387, 2001.
- [16] T. Parson. *Theory and Applications of Graphs*, chapter Pursuit-evasion in a graph, pages 426–441. Lecture notes in mathematics. Springer-Verlag, 1076.
- [17] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.
- [18] E.M. Royer, P.M. Melliar-Smith, and L.E. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *Proc. International Conference on Communications*, pages 857–861, 2001.
- [19] P. D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory Ser. B*, 58(1):22–33, 1993.
- [20] P. Wan, G. Călinescu, X. Li, and O. Frieder. Minimum-energy broadcasting in static ad hoc wireless networks, 2002.
- [21] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. Technical Report UIUCDCS-R-2003-2351, UIUC, 2003.