Cleaning an Arbitrary Regular Network with Mobile Agents*

Paola Flocchini, Amiya Nayak and Arno Schulz

School of Information Technology and Engineering
University of Ottawa
800 King Edward Avenue
Ottawa, ON K1N 6N5, Canada
Email: {flocchin,anayak,aschulz}@site.uottawa.ca

Abstract. In this paper, we consider a contaminated network with an intruder. The task for the mobile agents is to decontaminate all hosts while preventing a recontamination and to do so as efficiently as possible. We study under what conditions and what cost a team of mobile agents can do this in synchronous arbitrary regular graphs using the breadth-first-search strategy. Due to the nature of the experiment we use a genetic algorithm to find the minimum number of agents required to decontaminate a given network. The results show that there is a relation between the degree, the size of the graph, and the number of starting locations of the mobile agents. in particular, this relation demonstrates the possibility of improvements in reducing the number of mobile agents used depending on the number of starting location in arbitrary regular graphs.

Keywords: Mobile Agents, Intruder Capture, Graph Search, Mesh.

1 Introduction

1.1 The Problem

Consider a network where nodes represent hosts and edges represent connections between hosts. An *intruder* is a dangerous piece of software (e.g., a virus) that moves arbitrarily fast from host to host contaminating the nodes. The intruder capture problem consists of deploying a team of collaborative software agents to capture the intruder.

We can formulate the intruder capture problem equivalently in terms of a *decontamination (or cleaning)* problem in which each node of the network can be in one of three possible states: *clean*, *contaminated*, *guarded*. Initially all nodes are *contaminated* except for the nodes containing an agent (which are *guarded*). A node becomes *clean* when an agent passes by it, and an unguarded node becomes *contaminated* if one of its neighbors is contaminated. A guarded node is also clean. The decontamination problem consists of reaching a situation where all

* Work partially supported by Natural Sciences and Engineering Research Council of Canada.

the nodes are simultaneously *clean*. In particular, we are interested in *monotone* strategies of decontamination; i.e., we want that a node that becomes clean will never be contaminated again. In other words, we have to design cleaning strategies that "protect" clean node from recontamination; once a node is clean, all its neighbors must be clean or guarded.

An agent is a mobile entity that can move from node to a neighboring node. Agents can communicate by accessing local small whiteboard located at the nodes (whiteboards of size $O(\log n)$ are enough for our purposes). The whiteboard of a node will contain the state of the node (*clean*, *contaminated* or *guarded* and any other information the agents need to communicate to the other agents). Finally, we assume that an agent can "see" the state of its neighbors. The possible actions of an agent are:

- LOCAL COMPUTATIONS (or local actions): Each agent is provided with $O(\log n)$ of local memory to be used when performing local computations, where n is the number of nodes. Particular local action are: the observation of the neighbors' states; cloning (the agent can clone several copies of itself); termination (an agent can terminate its execution).
- MOVEMENTS from a host to a neighboring host: In this paper, we consider synchronous movements, i.e., it takes one unit of time for an agent to traverse a link. Moreover, we assume that the agents start simultaneously.

When designing a decontamination strategy, the efficiency is measured in terms of the number of agents deployed (maximum number of agents simultaneously active), number of moves, and the time it takes to clean a network. To determine the minimum number of mobile agents required to decontaminate a given network using this strategy, we use a genetic algorithm to find which combination of starting home bases will reduce the number of mobile agents used. The genetic algorithm provides a good estimate of the number of mobile agents required to clean a given network of fixed degree, size, and the number of starting home bases for the mobile agents.

1.2 Related Work

A variation of the intruder capturing (or decontamination) problem has been widely studied in the literature under the name of *graph search problem*. This problem was first introduced by Breish [3] and Parson [12] and was studied extensively under different variations (edge search, node search, mixed search) (e.g., see [4, 8–11, 13]). The main goal of all these investigations was to determine the minimum number of agents required to perform the search. Determining such a number ("searching number") in an arbitrary network is an NP-complete problem.

In all the graph search variations studied in the literature, searchers may be removed from a node and placed on any other node of the graph being searched, i.e., they are allowed to *"jump"* while they perform the searching task.

However, in a networked environment, agents cannot jump, but can only move from node to neighboring node. Simulating the jump by neighboring moves may make the strategy non monotone.

A variation of the node/edge-search problem is called the *contiguous search problem* [1, 2] which adds the requirement that 1) the agents can move only from node to neighboring node without jumping, 2) the strategy is monotone, and 3) the decontaminated area is connected. This problem is harder than the non-contiguous one as it has been shown in [2] that there are networks where the contiguous searching number is strictly greater then the non-contiguous searching number. Finding the contiguous searching number is still an NP-complete problem for general graphs. A few specific topologies have been studied; for example, it has been shown that the problem can be solved in linear time in trees [1], meshes and tori [5]. Moreover, strategies and upper bounds have been studied only in hypercubes [6]. All the previous investigations have been carried out in asynchronous environment.

1.3 Our Result

In this paper, we consider the contiguous decontamination problem in a synchronous arbitrary regular network without the restriction that the decontaminated must be connected. We first describe a general strategy in which the agents perform the decontamination by moving in a breadth-first manner, making sure that no recontamination will occur. This technique can be applied to any arbitrary topology. This general strategy can be initiated by an arbitrary number of starting locations, and its efficiency depends on the number of starting places and their location.

Generally, starting from more locations will increase the number of agents but will decrease the time. An interesting question is: given a network and a number of starting agents what is the optimal placement of the agents? Even in symmetric networks, increasing the number of starting locations, the problem becomes quite complex; thus, in order to obtain minimum number of agents we resorted to simulations using a genetic algorithm. In fact, to choose good starting locations, we design a genetic algorithm that will find the solution that will use the least number of agents in a single step, for a given graph and a fixed number starting locations. Through experiments, we show that as the number of home bases increases, the number of agents required decreases in all network topologies considered.

2 The Strategy

The cleaning strategy (protocol CLEAN illustrated in Figure 1) is very simple. Initially, the agents are placed in arbitrary starting location. Each starting agent will try to move its clones on a breadth-first-search (BFS) tree of the network rooted at its starting position. More precisely, at each step, if an agent arrives

to a node alone, it cleans the node, clones itself as many times as the number of contaminated neighbors, and sends them on the corresponding links. If, however, more than one agent arrives at a node simultaneously, only one of them survives, cleans the node, clones itself as many times as the number of contaminated neighbors, and sends them on the corresponding links; the other agents terminate here.

Protocol CLEAN (for an agent a arriving at node x)

If a is alone:
  Clean x.
  Check the state of neighbors.
  Let $N_{D(x)}$ be the set of decontaminated neighbors of x.
  Clone $|N_{D(x)}|$ agents.
  Send the cloned agents to the decontaminated neighbors.
If a is not alone:
  Locally choose a leader.
  If I am the leader:
    Clean x.
    Check the state of neighbors.
    Let $N_{D(x)}$ be the set of decontaminated neighbors of x.
    Clone $|N_{D(x)}|$ agents.
    Send the cloned agents to the decontaminated neighbors.
  Otherwise
    Terminate.

Fig. 1. Protocol CLEAN

The procedure *locally choose a leader* in Figure 1 consists in selecting one of the agents to continue the cleaning operation, the leader is the agent that first accesses the local whiteboard.

In a clean area, internal nodes are the nodes whose neighbors are all clean, border nodes have some clean neighbors and some contaminated neighbors.

Theorem 1. *Protocol* CLEAN *performs a monotone decontamination of the network.*

Proof. We want to prove by induction that once a node is clean, it will never be re-contaminated.
Basis. The starting locations of the agents are clean and since, by definition, enough agents are sent simultaneously to all their contaminated neighbors, they will not be re-contaminated in the subsequent step. Moreover, the border nodes are all guarded (in this case border nodes are just single nodes).
Induction. At step k, some nodes are clean and the border nodes are guarded. At step $k + 1$ the agents that are on the border nodes will clone themselves,

by definition of the algorithm, and then proceed to the contaminated nodes. As all contaminated neighbors receive at least one agent, the old border nodes become internal nodes, and thus cannot be re-contaminated in the subsequent step, while the border node are all guarded.

We have shown that once a node is clean, it will never be re-contaminated. Since the graph is connected, we have that all nodes will eventually be decontaminated. □

Notice that, for some specific topologies, it can be easy to compute the number of agents needed for decontamination, when considering a single starting location ("home base"). In this case, the number of steps is always equal to the diameter of the network, while the number of agents clearly depends on the topology. For example, in the case of the hypercube, the maximum number of agents simultaneously active would be equal to the maximum number of edges between levels of the broadcast tree (which is $(\lfloor \frac{n}{2} \rfloor + 1) \cdot (\lfloor \frac{\log n}{2} \rfloor + 1)$).

However, when we have more than one home base for the mobile agents, the nature of the problem becomes more complex with the added difficulty of finding the optimal configuration. Even in symmetric networks, adding multiple starting locations increases the complexity; thus, in order to obtain minimum number of agents we resorted to simulations using a genetic algorithm.

3 The Genetic Algorithm

3.1 Genetic Algorithm Background

The genetic algorithm as described by Goldberg [7] is a process composed of two elements: a population of strings and a fitness function. As can be seen in Figure 2, at each round of the genetic algorithm, the fitness function allows to pick the fittest individuals (the strings that achieve a better score according to the fitness function) and then uses them as a basis to generate a new generation. As each generation brings a new population generated from the fittest individuals from the previous generation, over the course of several generations, the individuals tend to get better and better fit, thus providing a good solution in a relatively short period of time as the least fit individuals are no longer considered. However, evolving the optimal solution is more difficult and will take more time as in the later generations. When there are many individuals with equal fitness, there is no bias towards any particular individual.

3.2 Our Implementation

For our experiment, we use individuals where each individual represents the starting positions of the mobile agents within the graph. The graphs are represented by a Boolean array where each entry of the array corresponds to a node in the graph. If the Boolean value corresponding to a node is true it means that the node is going to be a starting location for a mobile agent. These arrays are

```
Genetic Algorithm:

For 52 generations do
    Set total fitness TS to 0.
    While there exists a candidate string out of 1024 candidates
        s = next candidate string.
        Evaluate s with the fitness function F(s).
        Add F(s) to TS.
    End While
    Create a biased wheel of size TS where for each s,
        a space corresponding to the F(s) fitness is allocated.
    Pick randomly 1024 candidate strings from the wheel.
End For
```

Fig. 2. Genetic Algorithm

stored in matrix composed of 1024 individuals (the population size used by the genetic algorithm).

In order to have a starting point for the genetic algorithm, the initial population is generated in a random manner. Each individual, having the same number of agents, will place these agents in random positions of the array (if an agent is already there, another random position is selected until an empty position is found). Once the population has been generated, it is then passed on to the genetic algorithm which then runs 52 times, each time generating a new population (from the previous generation).

Each population that is passed on to the genetic algorithm is evaluated according to a "fitness" function. The fitness function used is the maximum number of agents that were used at any one step during the decontamination of the network. The fitness obtained is then stored in order to determine the maximum fitness and the best individual of each generation later on.

After all the fitnesses have been collected, the algorithm sets up a biased evolutionary wheel, that gives to each individual a percentage of the wheel proportional to its fitness compared to the total fitness of the population (note that the wheel represents the total fitness of the population). This favors the fittest individuals as they will represent a greater percentage. Thus, while creating a new generation, as the selection is done in a random manner, the fittest individuals are most likely to be selected. The new generation is then created using one of two techniques, *mutation* and *cross-over*, described below.

The *mutation* consists of selecting an individual randomly in the evolutionary wheel. We then change the position of one of the mobile agent's starting home base within the Boolean array which represents an individual (note that it is still possible for a selected individual not to get changed during the mutation).

The *cross-over* consists of selecting two individuals randomly in the evolutionary wheel and then creating two new individuals by switching the second half of each of the selected individuals. For example, if the two parts of the first selected individual is (x1, x2) and the second selected individual is (y1, y2), the cross-over process creates two new individuals with two parts as (x1, y2) and (y1, x2).

After the new generation has been bred, the best of this generation is then compared with the best of all previous generations; if the fitness is higher, the new champion individual is preserved. Finally the next generation is passed to the genetic algorithm, and the process is repeated until we reach the 52nd generation.

3.3 The Benefits and Restrictions of Genetic Algorithms

The advantage of using the genetic algorithm is that it allows us to narrow down to an acceptable solution for a given topology. Most importantly, it gives us a good approximation (if not the maximum number of agents required) for a particular network. We are also able to use it instead of doing an exhaustive search to evaluate each and every possible configuration of the mobile agents starting home bases for a given network topology.

There are two limitations with this approach. First, without resorting to an exhaustive search, it is not possible to confirm whether the results obtained by the genetic algorithm truly represent the optimal solution for a particular graph. Second, there is the possibility of premature convergence giving a local minima as a result instead of finding a lower minimum. To minimize the effects of these limitations, we run each configurations 300 times as explained in the following section.

4 Experimental Results

4.1 Methodology

Using the described genetic algorithm we are able to test specific arbitrary regular graph groups of given size, degree with different number of starting home bases for the mobile agents. For statistical significance, we run the genetic algorithm over 60,000 individual experiments. The entire experiment uses 200 different graph groups (five graph sizes with 512, 768, 1024, 1576, 2048 nodes; four degrees: 4, 8, 16, 32; and ten different numbers of home bases: 1, 2, 3, 4, 6, 8, 10, 12, 14, 16). Each graph group can be viewed as a set of graphs with parameters (size, degree, number of home bases); for example, (512, 4, 10) represents a graph group where all graphs have 512 nodes, degree 4, and 10 home bases. For each graph group, we run 300 experiments which is divided into 30 series of experiments, each series uses a different graph with the same set of parameters. Each series is then run 10 times. Each experiment yields two results: the number of mobile agents used and the number of steps used.

We observed that within a graph group, the variation of results for the number of mobile agents between individual experiments and between series of experiments is less than 3%. Occasional abnormalities in the results due to pre-mature convergence in the genetic algorithm are less than .5% and are ignored. For the number of steps, we also observed less than .5% abnormality (considered when the variation is not within ± 1 step).

Once all the results have been collected for each graph group; the median is taken for both sets of results for each series of experiments (i.e., the number of mobile agents used and the number of steps used). We then take the median of all the medians within a graph group which is then plotted (please refer to Figure 4 for the number of mobile agents used and to Figure 3 for the number of steps used).

4.2 Number of Steps Used

Given the synchronous nature of the experiments, one of the interesting facts to consider is the number of steps taken by the mobile agents to decontaminate the network, and how it relates to the size, the degree of the network, as well as the number of starting home bases for the mobile agents.

The results are shown in Figure 3. These results were as expected due to the BFS strategy used by the mobile agents. At each step, in this strategy, it is possible to predict that the number of agents at the next step. For example, in a network of degree 4 with 512 nodes and one starting base, the number of mobile agents would roughly increase 4 fold at every step. One can then see that once we are at a step where the number of agents exceeds the number of nodes in the graph it would be the maximum of steps used for that graph.

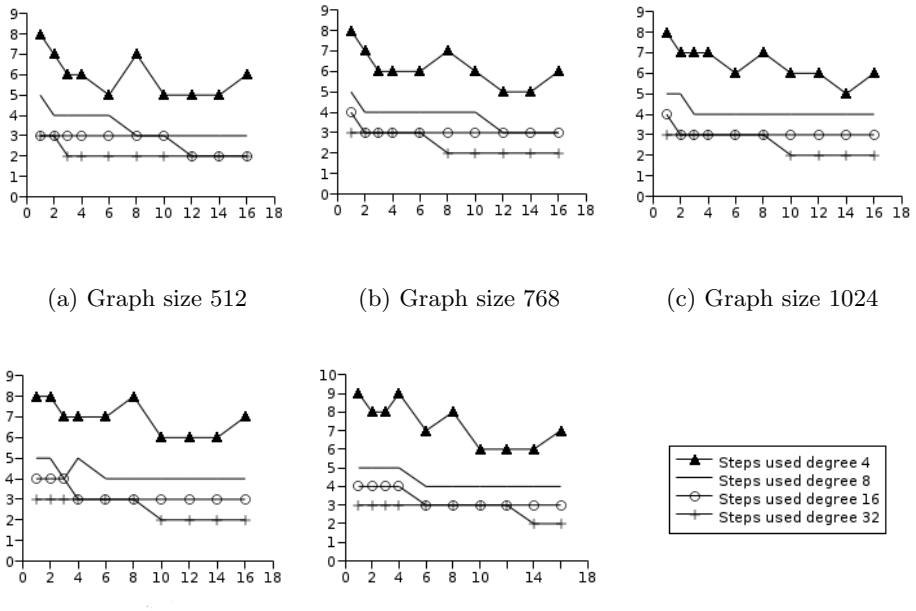
Figure 3 shows that the number of steps varies between 2 and 9. The expected behavior can be seen for all graph groups; that is, as the number of home bases increases the number of steps taken decreases.

4.3 Number of Agents Used

The main goal of this experiment was to determine the minimum number of mobiles agents used for decontaminating different networks with one or more home bases. The results for different graph groups are shown in Figure 4. We observe the following. First, for a given graph group, fewer agents are required for certain numbers of home bases. For example, the graph group (2048, 32, 2) uses fewer agents than the graph group (2048, 32, 1). Second, the variations of the number of agents used depends on the degree of the graph groups. The variation is larger for graph groups of higher degrees.

These results were expected. As we increase the number of home bases, we see a drop in the number of agents. This is true for all graph groups.

Due to the nature of the decontamination strategy based on BFS technique, there will always be an overuse of the number of agents. A contaminated node



(a) Graph size 512 (b) Graph size 768 (c) Graph size 1024

(d) Graph size 1576 (e) Graph size 2048 (f) Legend

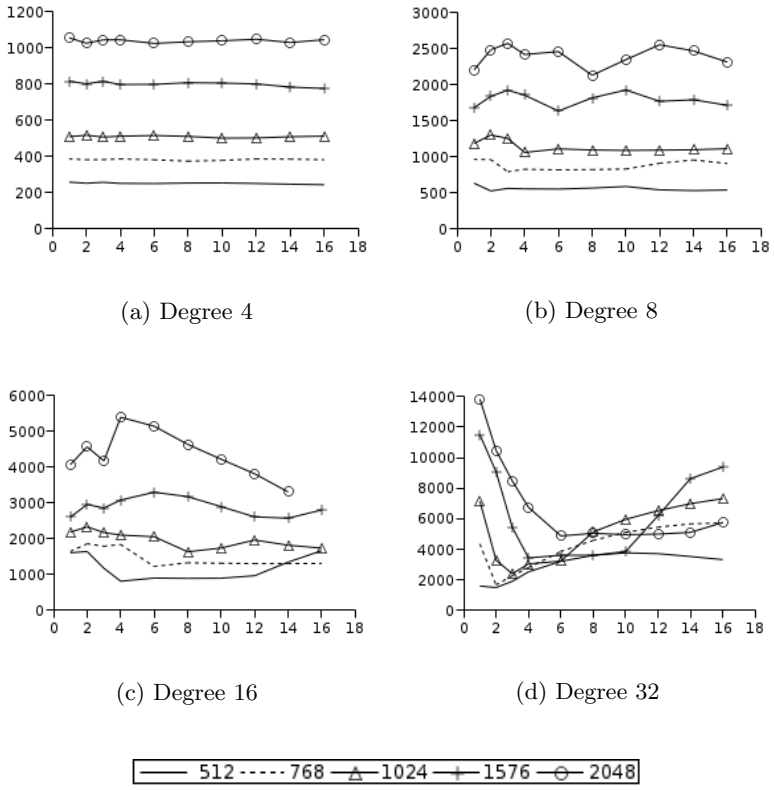
Fig. 3. Steps Used (x-axis: number of home bases; y-axis: number of steps)

will always receive an agent from all its decontaminated neighbors. The overuse of agents is proportional to the ratio of decontaminated and contaminated nodes. Initially, the overuse is low as there are fewer decontaminated neighbors for each contaminated node. The overuse is highest in the last stage of decontamination as most of the neighbors of contaminated nodes have been decontaminated.

Figure 5 illustrates the propagation of the mobile agents, in the given graph of degree 4 with 7 nodes and 1 home base, through the network using the BFS technique. In the last step, while there are only 2 nodes left to be decontaminated, 6 agents are used by the algorithm. This example clearly shows the overuse of agents when using the BFS technique in a synchronous network.

5 Conclusions

In this paper, we considered the problem of decontaminating synchronous networks with mobile agents using BFS technique. We used a genetic algorithm to avoid exhaustive search. The genetic algorithm allowed to find a good approximation of the minimum number of agents needed to decontaminate the network. We considered various networks with different number of home bases to study



(a) Degree 4

(b) Degree 8

(c) Degree 16

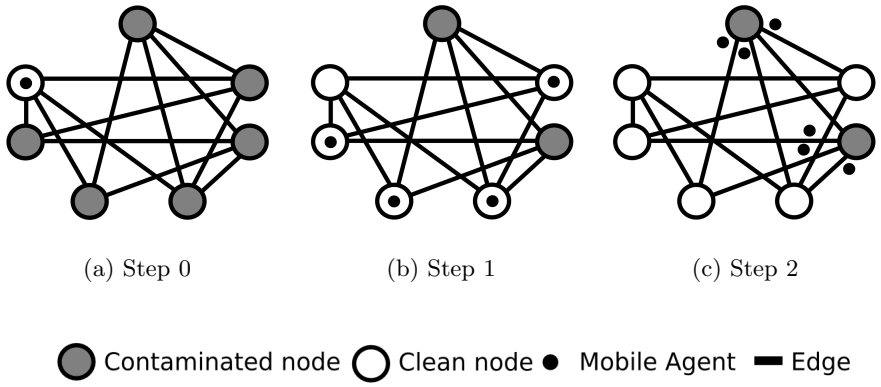
(d) Degree 32

(e) Graph Sizes

Fig. 4. Agents Used (x-axis: number of home bases; y-axis: number of agents)

the relationship between the number of home bases and the agents/steps required. The experiments allowed us to confirm that as the number of home bases increases, the number of agents required decreases in all network topologies considered. We observed that as the number of home bases increases the number of steps taken to decontaminate the network also decreases. The overuse of agents due to the BFS strategy increases with the decrease in the number of contaminated nodes.

Currently, we are investigating the use of global blackboard, other models to coordinate mobile agents, and asynchronous networks to see which model further reduces the number of mobile agents used. We will also consider simulating classic networks such as the hypercube, the mesh, and other networks to see if the use of several home bases can reduce the number of mobile agents required



(a) Step 0 (b) Step 1 (c) Step 2

(d) Legend

Fig. 5. Overuse of mobile agents

for one home base [6] when using the BFS strategy. Finally, we are considering the use of genetic algorithm to find the optimal number of home bases for a given network.

References

1. L. Barrière and P. Flocchini and P. Fraignaud and N. Santoro. Capture of an Intruder by Mobile Agents. *Proc. 14-th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 200-209, 2002.
2. L. Barrière and P. Fraignaud and N. Santoro and D.M. Thilikos. Searching is not Jumping. *Proc. 29th Workshop on Graph Theoretic Concepts in Computer Science, (WG)* LNCS, vol. 2880, 34-45, 2003.
3. R. Breish. An intuitive approach to speleotopology. *Southwestern cavers*, **VI** (5), 72-28, 1967.
4. J.A. Ellis and I.H. Sudborough and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113: 50-79, 1994.
5. P. Flocchini, L. Song, F. L. Luccio. Size Optimal Strategies for Capturing an Intruder in Mesh Networks, *Proc. of 2005 International Conference on Communications in Computing (CIC 2005)*.
6. P. Flocchini, M. J. Huang, F. L. Luccio. Contiguous Search in the Hypercube for Capturing an Intruder. *Proc. 19th IEEE International Parallel and Distributed Processing Symposium* (IPDPS 2005).
7. D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesley.* , 1989.
8. L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, **47**, 205-218, 1986.
9. A. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM*, **40** (2), 224-245, 1993.

10. N. Megiddo and S. Hakimi and M. Garey and D. Johnson and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, **35** (1), 18-44, 1988.
11. B. Monien and I.H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, **58**, 209-229, 1988.
12. T. Parson. Pursuit-evasion problem on a graph. *Theory and applications in graphs*, Lecture Notes in Mathematics, Springer-Verlag, 426-441, 1976.
13. S. Peng and M. Ko and C. Ho and T. Hsu and C. Tang. Graph searching on chordal graphs. *Algorithmica*, **27**, 395-426, 2000.