Experimental Analysis of Black Virus Decontamination by DisJ

Jie Cai

School of Computer Science
Carleton University
Ottawa, ON, Canada K1S 5B6
Email: jie.cai@carleton.ca

***Abstract*—In this paper, we experimentally investigate the problem of black virus decontamination. The black virus decontamination is a recently investigated network security problem occurring in networked systems supporting mobile agents. The existing research work on the topic has been focusing on theoretical investigations and analyses. Among the existing simulators for reactive distributed algorithms in network applications, we use Distributed Algorithm Simulation Java, which combines many advantages and overcomes many shortcomings of existing simulators. We consider the basic solution protocol for decontaminating an arbitrary network. We investigate its behaviour, properties and performance through an extensive number of computer simulation runs. The simulation results not only confirm the existing theoretical results, but also disclose many interesting behaviour/properties of the solution protocol. In particular, they show that the examined protocol outperforms random search. The influence of graph connectivity density and size on complexities (move, time, and agent size) is clearly depicted.**

***Keywords*—Black Virus; Mobile Agent; Graph Exploration and Decontamination; Simulation.**

I. Introduction

Recently, many investigations have been performed on various distributed security issues caused by introducing mobile agents into computer networks [1]. For example, a malicious agent can cause computer nodes to malfunction or crash, while a contaminated computer node can in turn destroy mobile agents for various malicious purposes. The former situation is categorized as *harmful agent*, and the latter as *harmful host*.

In the *harmful agent* problem, a dangerous mobile agent moves through the network infecting the visited sites; the task is to decontaminate the network using a team of system agents avoiding recontamination. The problem is referred as *intruder capture* (IC), *graph decontamination*, or *connected graph search*. The mobile intruder is harmful to network sites, but not to the system agents. This problem has been investigated in different settings and topologies by Barrière et al. [2], [3], Blin, Fraignaud, Nisse, and Vial [4], Dereniowski [5], Flocchini et al. [6], [7], [8], Fomin, Thilikos, and Todineau [9], Imani, Sarbazi-Azad, Zomaya, and Moinzadeh [10], Luccio et al. [11], [12], [13], Nisse [14], Shareghi, Sarbazi-Azad, and Imani [15], Yanga, Dyerb, and Alspach [16], among others.

For harmful host, the theoretical focus has been on the *black hole search* (BHS) problem, in which a network node is infected by a process which destroys any arriving agent without any detectable trace. The problem has been extensively investigated in different settings and topologies by Chalopin, Das, Labourel, and Markou [17], [18], Cooper, Klasing, and Radzik [19], Czyzowicz et al. [20], [21], Dobrev et al. [24], [22], [25], [26], [23], Glaus [27], Klasing, Markou, Radzik, and Sarracco [28], and Shi [29]. A black hole is *static*, that is, it does not propagate in the network and so it is not harmful to other sites.

A. Black Virus Decontamination

The Black Virus (BV) was introduced in [30], [31] to combine factors missed by BHS and IC to create a new model, in which a harmful process is mobile (like an intruder) and harmful to the system agents (like a black hole). Therefore, the Black Virus Decontamination (BVD) problem is to model a novel security issues caused by mobile agents. The authors studied in detail the BVD problem for three large classes of common network topologies: multi-dimensional grids, tori and hypercubes. Recently, a deterministic exploration protocol for BVD in arbitrary network was developed [32]. It has been proven that *monotonicity* (that is, once a node is explored or decontaminated, it is never recontaminated again) is a necessary condition for a solution protocol to be damage optimal. Theoretical complexity analyses on BV spreads, number of agents, moves, and simulation time are performed for all the solutions. All protocols are optimal both in terms of spread (the number of casualties) and size (the number of agents).

B. Simulation Work on Mobile Agents

1) Mobile Agents in Network Applications and Their Simulation: Mobile agents are used in some practical network applications, for example, distributed data mining [33], network management [34], routing [36], consensus problems, network mapping, multiagent coordination for connection, and etc. Amin and Mikler attempted to use mobile agents to design and implement agent based Distance Vector Routing (ADVR) to reduce the overhead and overcome the robustness issues associated with conventional routing protocol [36]. However, except mentioning the fact that Object-Oriented paradigm is adopted, no details on the simulations were provided. Rubinstein and Duarte investigated a mobile agent based network management solution to address scalability and efficiency issues [35]. Network Simulator (NS) is used in the simulation work. Olfati-Saber and Murray studied consensus problems for networks of dynamic agents with fixed and switching topologies in [37], [38]. Experimental results are provided to demonstrate the effectiveness of the theoretical results. Again the details of the simulation were not presented. Minar, Kramer, and Maes investigated the cooperation of mobile agents for mapping networks to overcome the shortcomings of a centralized solution in [39]. The mobile-agents approach is chosen to obtain routing maps in a distributed and decentralized strategy. Ji and Egerstedt address the connectedness issue in multiagent coordination, i.e., the problem of ensuring that a group of mobile agents stays connected while achieving some performance objective in [44]. In particular, they study the rendezvous and the formation control problems over dynamic interaction graphs.

2) Simulation Platforms for Distributed Algorithms in Network Applications: In this subsection, we briefly describe existing simulators for reactive distributed algorithms in network applications with the purpose of: comparing simulation and platforms in network systems; and having reference terms for Distributed Algorithm Simulation in Java (DisJ), a simulation tool we introduce later in this paper.

Distributed Algorithms in Java (DAJ) [45], Toolkit for Distributed Algorithms in Java (T-DAJ) [40], Distributed Algorithms Platform (DAP) [41], Simulation of Network Algorithm (SinAlgo) [42], Distributed Algorithm Simulation Terrain (DisASTer) [43] are all platforms for designing, implementing, testing, simulating, and visualizing distributed algorithms. DAP and SinAlgo are mainly suited for wireless network. The above simulators provide various good features although they usually do not have all of them: Object-Oriented Design and implemented by popular languages, i.e., C++ and Java; user friendly GUI; synchronous and/or asynchronous settings; fix or mobile networks; and various level debug capabilities. However, they have some common limitations or disadvantages: only supporting message passing model; only supporting bi-directional link; requiring some level of configuration or coding to create network topology; tightly coupled algorithm implementation and topology creation; lack of statistics calculation and/or display; no support on adversary events; limited interactive information display during simulation.

DisJ, used in this investigation, overcomes almost all the above shortcomings. In addition to its rich functionalities, one of the main advantages is it decouples users' protocol developing activity from defining network topology and executing the protocol. This means intended protocol and network topology are developed, defined, and built separately from each other and from simulation engine. However, one of the main inconveniences is lack of automation to run large number of simulations to generate statistics results.

C. Main Contributions

The problem of exploring and decontaminating a Black Virus in arbitrary graph by multiple mobile agents has been studied by simulation using DisJ. The model, objective, constraints, and a deterministic solution are reviewed and presented. Large number of simulations on different sizes of graphs with many connectivity densities are carried out.

The algorithm beats random exploration for all graphs at each connectivity level. The simulation disclosed many interesting behaviours of the solution protocol. Simulation demonstrates the worst case complexity analysis in [32], e.g., agents may move in one direction to explore one node, then move to an opposite direction to explore next node at the other end of a graph. In addition, agents may pass some nodes multiple times. In addition to proving the analytical results, simulation provides deep understanding on influence of graph connectivity density and size on complexities (move, time, and agent size). The move, time, and agent size increase with connectivity, but move and time start decreasing after reaching maximum at 40%-60% connectivity levels. The move, and time and agent size seem to increase quadratically and linearly respectively with the graph size.

With regard to statistics of simulation results, it is observed that the standard deviations for moves, agents, and times are all very small compared with average estimations. Particularly the larger graphs and the higher the connectivity levels, the better statistical results are.

The rest of the paper is organized as follows. Section 2 introduces the framework and model, and basic strategy and algorithm. Section 3 introduces the DisJ platform, describes sample graphs, and presents simulation results and analyses. Section 4 presents our conclusion.

II. Framework and Algorithm

In this section, we first introduce the framework and model, then present general strategy and algorithm, finally discuss synchronous and asynchronous settings.

A. Framework and Model

The agents operate in a network whose topology is modeled as a simple undirected connected graph $G = (V, E)$. We denote by $E(v) \subseteq E$ the set of edges incident on $v \in$ V, by $d(v) = |E(v)|$ its degree, and by $\triangle$ the maximum degree of G. Every node has a distinct id, visible to the agents visiting it. The links incident to a node are labeled with distinct port numbers.

Agents are modeled as entities with computing power. They can move from a node to one of its neighbour. Communication among agents occurs when they meet at the same node. Each agent has a unique id. In G there is a node infected by a BV whose location is unknown, and any agent arriving at the BV is destroyed. When that occurs, the BV clones itself and spreads from the $current$ node to all the neighbouring nodes. Arriving at a node, a clone BV infects the node and stays inactive (until further triggering by agents) if there is no agent on the node; otherwise, the clone BV is destroyed. Thus, the only way to eliminate a BV from the system is to surround it completely and let an agent deactivate the BV by moving to the BV node. In this case, the node where the BV resides is cleaned and all the generated clones of that BV are destroyed.

The *BVD* problem is to permanently remove any presence of the BVs from the network using a team of agents. A protocol defining the actions of the agents solves the BVD problem if, within finite time, at least one agent survives and the network is free of BVs. The main constraint of a solution protocol is to minimize the number of nodes infected by BVs (i.e., agent *casualties*). It has been proven that *monotonicity* (i.e., once a node is explored or cleaned, it is never recontaminated) is a necessary condition for a protocol to be infection-optimal [30]. The BVD model is shown in Figure 1.

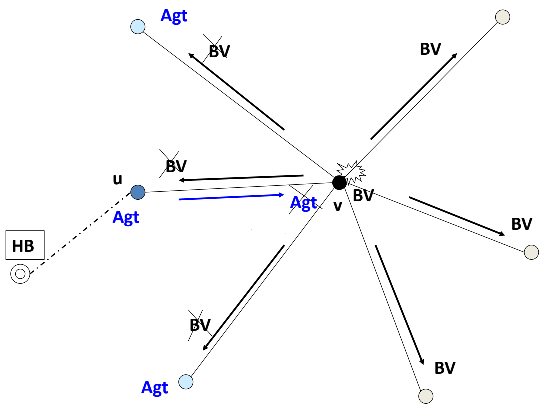


Figure 1. BVD Model.

Agents run the same protocol, but they can play different roles. There is an unique system agent called Leader Exploration Agent (LEA). During different stages, LEA can generate *Exploration Agent* (EA) and *Shadowing Agents* (SA). At any time, there is only one LEA and EA. When the EA is killed by exploring or cleaning a BV, LEA will create a new EA.

B. General Strategy and Algorithm

Agents locate the unknown BV by exploring the network with a constraint of minimizing infections, so at any stage of the exploration, agents always choose the next target with minimum chance of contaminations from unexplored portion of the network, G_{ux}. The degree of a node minus the number of explored neighbours of the node (defined as *residual degree*, $d_r(v)$) is equal to the number of new BV contaminations if a BV resides on this node. Calculating $d_r(v)$ of an unexplored node is possible because agents know the graph map, and the explored portion of the graph, G_{ex}. The solution strategy consists of three separate phases, *computing exploration sequence*, *shadow exploration*, and *surrounding and elimination*. Let us describe each phase in more detail.

a) Computing exploration sequence: LEA at Home Base (HB) calculates the Search Sequence (SS, the order of nodes) before exploring any other nodes. The algorithm is similar to Prim's algorithm to build MST. Here our algorithm builds a Minimum Residual Degree Spanning Tree $(M_{RD}ST)$. HB is the first node added to the processed part of the graph. The algorithm chooses next the node with minimum d_r among all the unprocessed neighbours which directly connect to the processed part of the graph. If there are multiple candidates with the same minimum d_r, the one with the shortest distance from the last chosen node is selected as the new target. We call this algorithm, *Minimum Residual Degree Exploration (MRDE)*.

b) Shadowed exploration: Let v (i.e., *target*) represent a node to be explored , $N(v)$ be the neighbours of v, and $N_{ex}(v)$ (resp. $N_{ux}(v)$) denote the set of explored (resp. unexplored) neighbours of node v. Let *current* denote a just explored node from which agents explore the next target, v. Exploring a *target* takes three sub-steps:

Deploying shadow agents: To insure monotonicity, our strategy employs some *SAs* to guard the previously visited neighbours of the *target*. Before exploring a new *target*, LEA computes $N_{ex}(v)$ and calculates the shortest distance paths in G_{ex} from the *current* node to them. A SA is sent along the shorted path to each one of $N_{ex}(v)$. The remaining agents also move to one of $N_{ex}(v)$, designated as u.

Exploring the *target*: LEA makes sure all SA's are in their positions before exploring the *target*. To minimize agent casualty, a *safe-exploration* technique is used: only EA moves from u to v to check if v contains a BV. If EA survives, it moves back to u and all agents at u move to v by using the same link which EA just explored; otherwise, LEA at u knows EA met a BV node.

Assembling SAs: If the *target* is not a BV node, LEA and other agents move to v, and LEA sends agents to the shadowed neighbours to bring SAs back to v. LEA updates v as new *current* node and fetch a new *target* node from SS. The above steps are iterated until a BV node is found.

c) Surrounding and elimination: Once the BV node is detected, all $N_{ux}(v)$ are contaminated. The new BVs are surrounded and eliminated sequentially. LEA assembles SAs and instructs them to surround the newly created BVs. Surrounding a node $w \in N_{ux}(v)$ means deploying an agent along the shortest paths to each one of w's neighbours unoccupied by a BV $(N(w) \setminus N_{ux}(v))$. Once a BV is surrounded, an extra agent (a *cleaning agent*) is instructed to move to it in order to clean it; such an agent dies.

C. Synchronous setting vs Asynchronous setting

In synchronous networks, it takes one unit of time for an agent to move from a node to another, while computing and communication times are assumed negligible compared with moving time. In asynchronous networks, the time of each activity (processing, communication, moving) is finite but otherwise, unpredictable.

Let us analyze where synchronization is needed in the general strategy discussed above. When EA is sent to explore a target, synchronization is not needed because, whether the target contains a BV or not, EA or a BV eventually returns to ”current” node, so LEA knows what happens. Synchronization is concerned only in one task, i.e., sending SAs to $N_{ex}(v)$ during shadow exploration. LEA instructs EA to explore the target only after all $N_{ex}(v)$ are protected by SAs. In synchronous setting, LEA calculates all $N_{ex}(v)$ and the shortest paths to them. Let $dist_{max}$ be the maximum of all shortest paths. LEA uses $dist_{max}$ to make sure that all SAs arrive at their destination nodes for protection before it instructs EA to explore the target. In asynchronous setting, LEA has to visit all shadow destinations one by one to be sure that all of them are properly shadowed by SAs before it instructs EA to explore the target. Both synchronous and asynchronous protocols have been implemented, and simulations are executed for both. Asynchronous complexities (agent move and simulation time) are larger than synchronous complexities, but within the same order, so in the following, the simulation discussions and results are focus on the synchronous protocol.

III. Simulation of BVD

In this section, we first introduce the DisJ platform, then describe sample graphs, and finally present simulation results and analyses.

A. Simulation Platform, DisJ

The simulation software used in this work is called DisJ, implemented in Eclipse environment as a plug-in [46]. The simulation engine per se is an event based simulation engine, which is driven by events put on an event heap. One of the main characteristics is that DisJ decouples the users’ protocol developing activity from defining user network topology. DisJ can be used to assist teaching or develop distributed protocols by researchers. It provides basic utilities (nodes, links, events, timers, and etc.) and other extensive features, e.g., random delays, different faults with probability, communication (unicast, multi-cast, and broadcast), and etc.

The original simulation engine supports only the message-passing model. To support mobile agents, more functions have been added: injecting agents on nodes; allowing an agent to move from one node to a neighbouring node; supporting different communication mechanisms among agents on the same node via whiteboard, token, or message; and finally supporting agents to create new agents.

DisJ provides rich features for developing protocols, designing network topologies, and debugging. It provides nice GUI interface to allow the users to define their network topologies by automatically generating, drawing, or inputting network topology from network matrix files. In addition to basic debugging features, it also provides advanced features, watching variables and states, restarting, logging execution and replaying, adjusting speed, and etc. It calculates basic statistics pertaining to distributed algorithms, e.g., the number of messages, agent moves, simulation time, and etc. The APIs for programming a protocol is small and simple. DisJ also has been designed with extensibility in mind. Some of the new features which can be imaged now are: network capability where 2 or more engines can link up in a network, dynamic addition of nodes, and more. DisJ has very good documentation, which includes a detailed user manual and a cookbook. They show step-by-step instructions for installing DisJ plug-in into Eclipse, defining a topology, writing protocol, and executing a protocol in defined topology. Figure 2 shows a sample DisJ Interface.

B. Simulation Sample Graphs

In order to obtain reliable data, we prepared 2255 graphs with different sizes and network connectivity densities. The sizes of the sample graphs are 20 ($graph20$), 40 ($graph40$), 60 ($graph60$), and 80 ($graph80$), and 100 ($graph100$) respectively. We define the network connectivity density/level as a ratio of the number of links of a graph to the one of a complete graph with the same size, i.e., $\frac{2m}{n(n-1)}$. For each size of graphs, we consider 10 connectivity levels, i.e., 10%, 20% to 100%. We implemented a computer program to randomly generate 50 graphs for each connectivity level of each size. Each graph is represented as $graph\#1\_\#2\_\#3$ ($\#1$: graph size; $\#2$: connectivity; $\#3$: graph instance). Two graphs are generated manually, i.e., graph20_18 and graph40_11, with special arrangement of degrees for certain nodes for easy visualization of the behaviour of different solution protocols. Following Figure 3 shows the sample graph20_18.

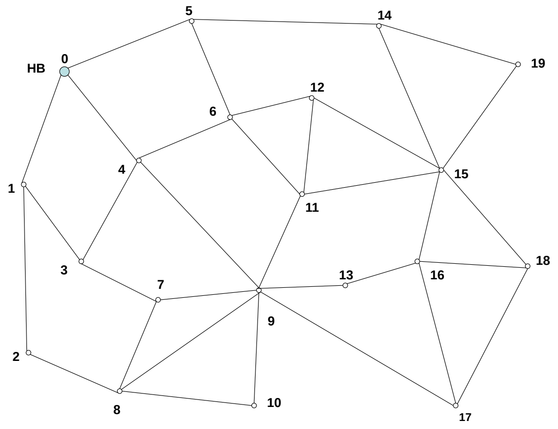
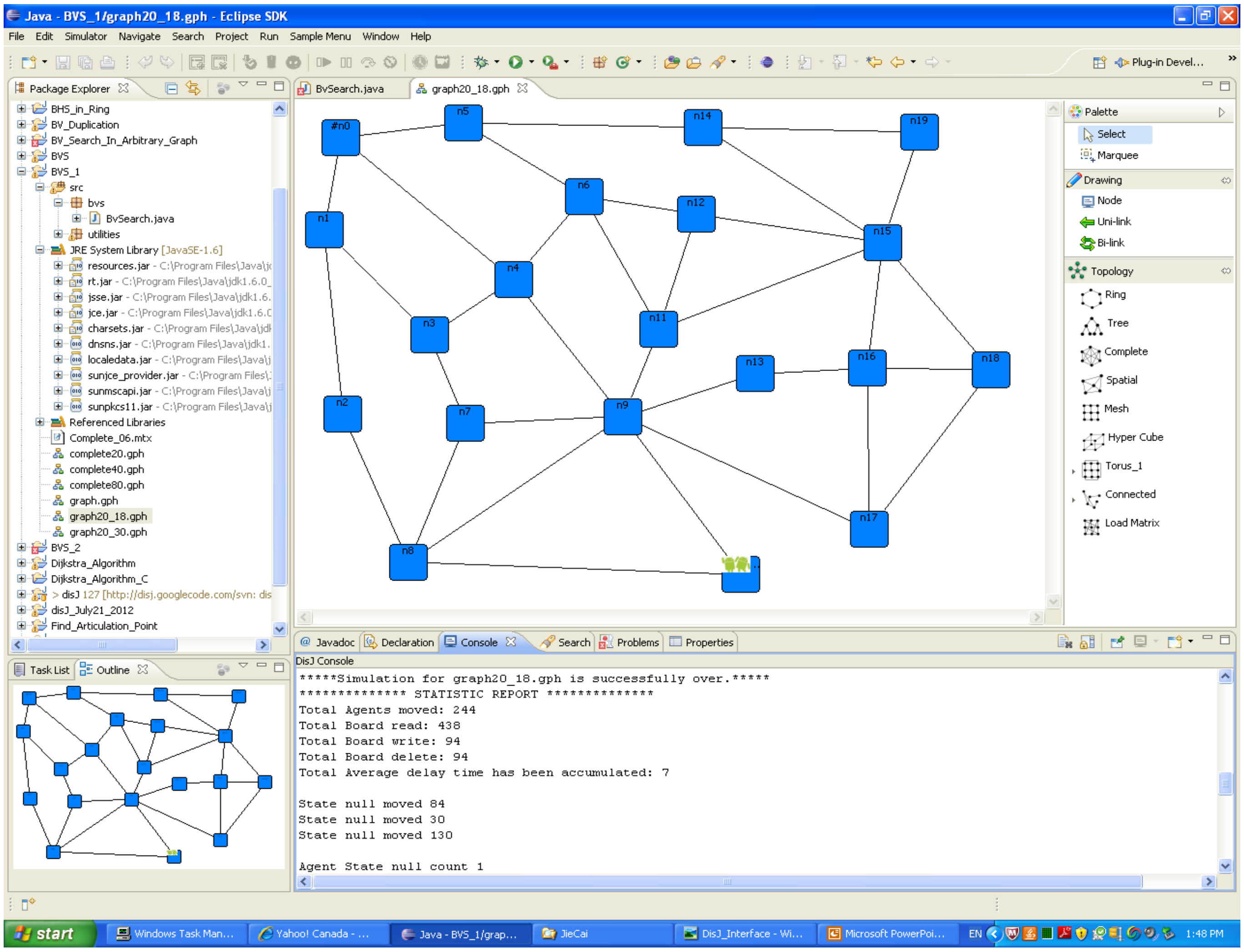


Figure 3. A sample graph20_18.

C. Simulation Results

1) Comparison between MRDE (Minimum Residual Degree Exploration) and Random Exploration (RANE) : Given two different exploration protocols, P_1 and P_2, how do we determine which one is better? Recall that the objective of the BVD is to remove all BVs with minimum contamination to the graph. Because we do not know the location of the BV a priori, the d_r of a node when the node is being explored represents the extent of possible contamination.

When a node i is explored, its residual degree, d_{ri}, is recorded. After exploring all nodes of the graph, they are



```
*****Simulation for graph20_18.gph is successfully over.*****
************** STATISTIC REPORT **************
Total Agents moved: 244
Total Board read: 438
Total Board write: 94
Total Board delete: 94
Total Average delay time has been accumulated: 7

State null moved 84
State null moved 30
State null moved 130

Agent State null count 1
```

Figure 2. DisJ Interface.

sorted into an array according to their residual degrees in descending order. If the same d_{ri} in the array appears multiple times, we use a coefficient, i.e., c_i, to record this. The resultant array is called *Array of Residual Degree (ARD)*. Comparing two exploration strategies becomes equivalent to comparing two ARDs. The arrays are compared lexicographically. If P_1's largest $d_{r1}(P_1)$ is smaller than P_2's largest $d_{r1}(P_2)$, P_1 is better than P_2. If $d_{r1}(P_1) = d_{r1}(P_2)$, we compare their coefficients. Whichever protocol has a smaller c_1 is better. If two protocol's c_1's are the same, we continue to compare the second largest d_{r2}'s of P_1 and P_2, and this comparison continues until a better protocol is determined.

In RANE, the next target node is randomly chosen among the unexplored direct neighbours of G_{ex}. We compared MRDE and RANE for the graphs with sizes of 20, 40, and 80 nodes. For each size category, we created 10 connectivity levels, so there are 30 comparisons in total. Simulation results demonstrate MRDE is never worse than RANE for all test scenarios. As shown in Figure 4, there are two curves, one for d_rs of each SS. The figures clearly show MRDE is better than RANE. In RANE, d_r changes dramatically, while in MRDE, d_r changes smoothly. It also demonstrated that, when the connectivity increases, the difference between MRDE and RANE decreases. When connectivity approaches 100%, i.e., the complete graph, MRDE and RANE are the same.

2) Exploration Behavior and Properties: Running the simulation on sample graph20_18 shown in Figure 3, we observed the following behaviours of the algorithm:

- G_{ex} is continuous. BV and unexplored nodes never cut G_{ex} into isolated pieces. This behaviour matches the monotonous property of the algorithm.
- Changing HB has effect on local SS, but has no impacts on SS in far areas. The behaviour is obvious when watching the searching process starting from nodes 0 to 8. This is a good property because it provides a little flexibility for users to choose where to start to explore the graph.
- SSs are influenced by the graph structures. It is observed that the initial SS starting from nodes 13, 16, 17, or 18 are

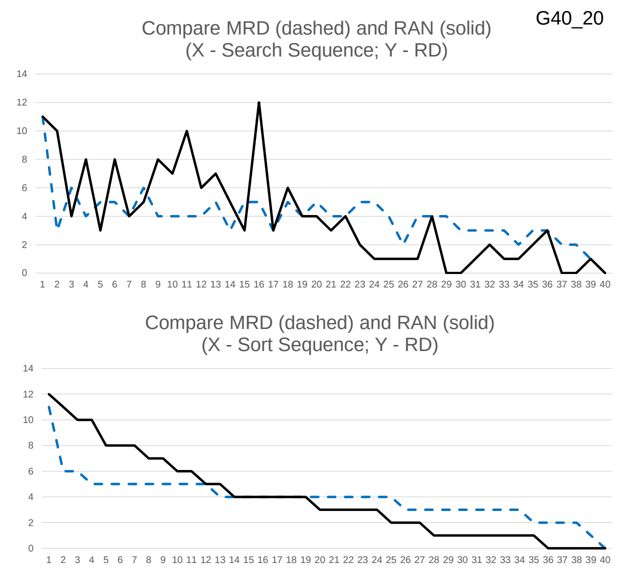


Figure 4. MRD_vs_RAN_G40_20.

dramatically different from those starting from node 0 to 8. This is because initially the exploration is confined by nodes 9 and 15, which have high degrees and are the exits for agents to explore other areas. After breaking through a barrier at node 9 or 15, all SSs follow the same/similar paths in the rest of the graph.

- Starting from the nodes with high degrees could reduce the *Size(G)* and exploring cost.
- The exploration sequence demonstrates the worst case complexity analyses in [32]. Agents may move to one direction to explore one node, then move opposite to explore another node at far end of a graph. In addition, agents may pass some nodes multiple times. When agents start from node 0, the exploration first moves in one direction along nodes 1 and 2, follows an opposite direction alone nodes 2, 1, 3, 4, and etc., and repeats this changing direction behaviour for several times.

3) Statistics of Simulation Results: As mentioned before, at each connectivity level of a given graph size, 50 graphs were generated, simulation is run for each graph, and complexities (agents, moves, and time) are recorded. Then we calculate averages and standard deviations of the complexities for each connectivity level of a given graph size. The ratio of standard deviation over average for complexities are plotted. Figure 5 shows the result for move complexity.

It is observed that, in all cases simulated, the standard deviations for moves, agents, and times are all very small compared with average estimations. With regard to the graph sizes, graph20 has the largest ratio of standard deviation over average for complexities. Large graphs produce small ratio, i.e., better statistical results. This is obviously true because small size graphs do not generate as good statistical results as compared with large size graphs. With regard to the connectivity levels, 10% connectivity generates the largest

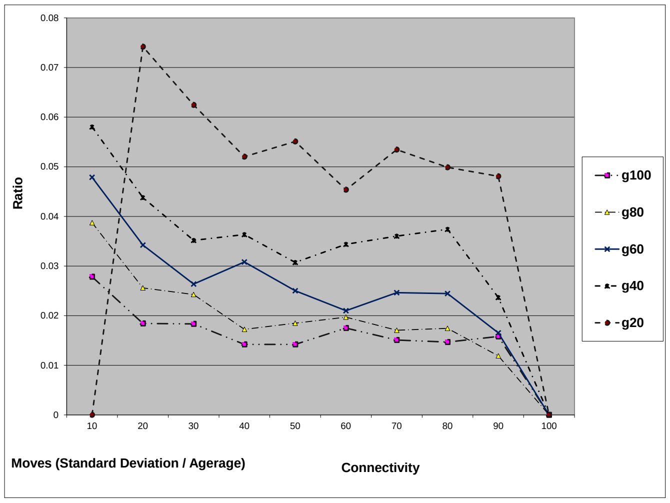


Figure 5. Moves SD Ratio vs Connectivity.

ratio of standard deviation over average for complexities. This is true because at connectivity level 10%, the number of links of the graphs is very small (close to the connectivity of ring or tree of the same size). The higher the connectivity levels, the smaller the ratio of standard deviation over average. At 100% connectivity, the ratio reaches zero. In this case, all the nodes are the same, so no mater where agents start and which path to take to explore the graph, the simulation results are the same.

4) Influence of connectivity density on complexity: Refer to the simulation results in Figures 6, 7, and 8, for move, agents, and time in graph20's, graph40's, graph60's, graph80's, and graph100's.

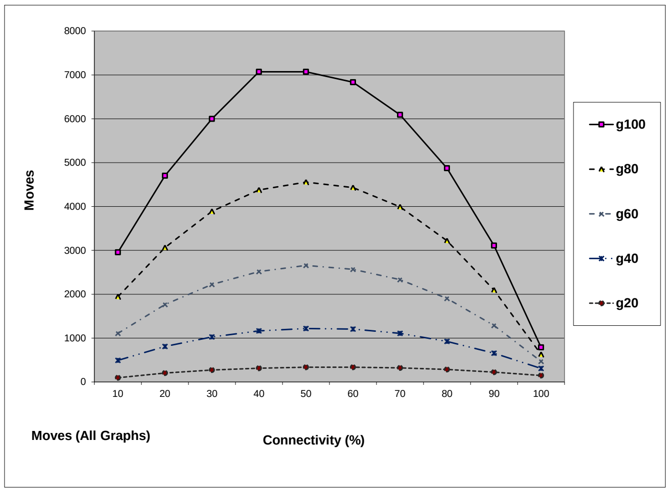


Figure 6. Moves vs Connectivity.

For given size of a graph, with the increase of connectivity level, we have the following results and observations:

a. Move cost gradually increases to maximum at 40%-60% connectivity levels, then it gradually decreases. It is interesting

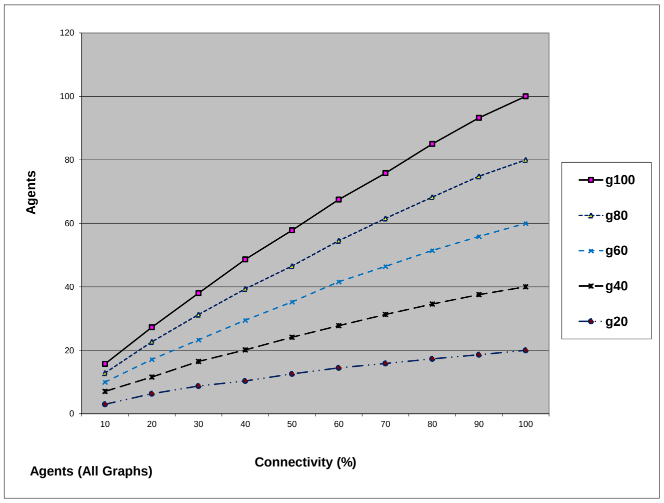


Figure 7. Agents vs Connectivity.

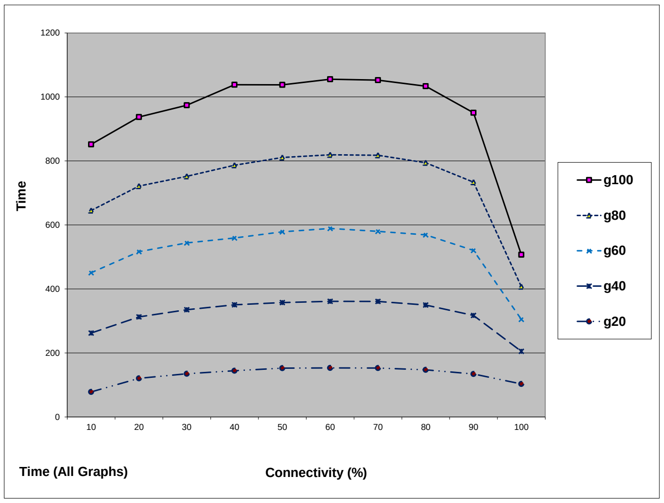


Figure 8. Time vs Connectivity.

to note that move costs are close for different graph sizes at 100% connectivity level. This demonstrates that dense graph optimization does improve move cost.

b. Time cost gradually increase to maximum at middle connectivity levels, then gradually decreases. It is noted that time costs are close for different graph sizes at 100% connectivity level. This demonstrates that dense graph optimization does improve time cost.

c. Team size continuously increases with the increase of connectivity. It reaches the maximum, i.e., graph size, at 100% connectivity level. It is obvious that high connectivity means high degrees, thus more shadow agents are needed to shadow exploration.

d. Regardless of the graph size, the behaviour of MRDE and RANE differentiates in low connectivity levels, but gradually becomes close at high connectivity. At 100% connectivity, the behaviours are the same. Complete graphs are completely symmetrical, i.e., choosing any one node as next target to explore gives exactly the same simulation results.

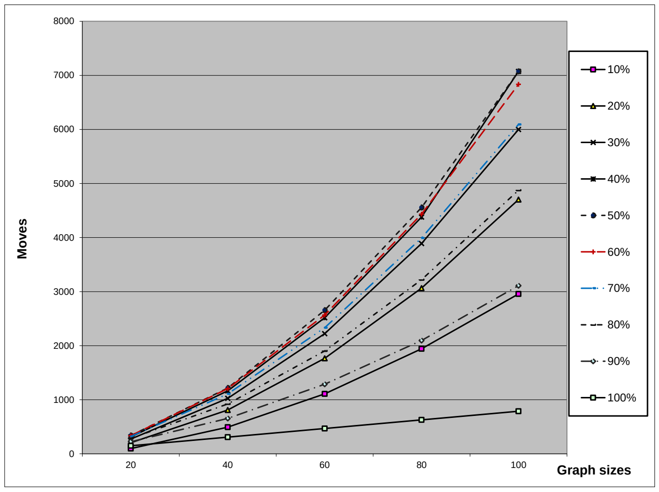


Figure 9. Moves vs Graph Size.

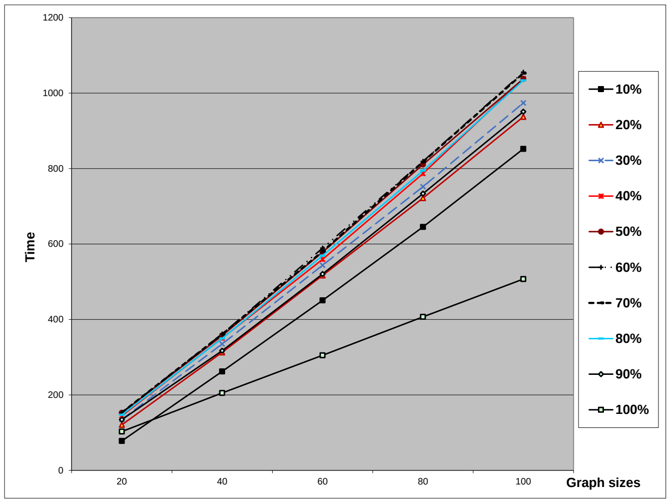


Figure 10. Time vs Graph Size.

5) Influence of graph size on complexity: For given connectivity levels, with the increase of graph sizes, the move, agent, and time are shown in Figures 9, 10, and 11:

a. From Figure 9, it is observed that the move costs seem to increase quadratically with the graph sizes. The move costs for 10% and 90%, 20% and 80%, 30% and 70%, 40%, 50% and 60% are very close respectively. Among all the connectivity levels, the complete graph has the lowest move costs, while 40%, 50% and 60% connectivity levels have the highest move costs. These behaviours demonstrate well that move costs increase with connectivity levels. It saturates at around 50% connectivity, then decreases to minimum when connectivity is 100%.

b. From Figure 10, it is observed that the execution time seems to increase linearly with the graph sizes. The execution times for 20% - 80% are relatively close to each other. Among

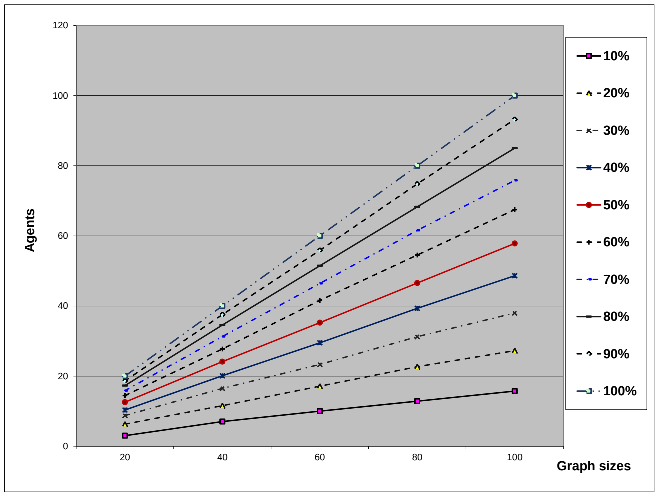


Figure 11. Agents vs Graph Size.

all the connectivity levels, th complete graph has the lowest time costs, while other connectivity levels have relatively high time costs. Generally speaking, the time costs are lower than move costs for same graph size and connectivity level because many activities happen in parallel.

c. Team size all increase when graph size increases. From Figure 11, it is observed that the Team size increases (linearly) with the graph size. As mentioned before, the connectivity also impacts the needed agents linearly.

IV. Conclusion

We have investigated the BVD problem in arbitrary graphs by mobile agents, a newly introduced network security problem. All the existing work has been focusing on theoretical investigation and analyses. In this paper, we instead focused on the experimental investigation of BVD. Among the existing simulators for reactive distributed algorithms in network applications, we used DisJ, which combines many advantages and overcomes many shortcomings of existing simulators. In addition to basic features, it offers mobile-agent support, flexible network element settings, simple APIs, user-friendly GUI, strong debugging capabilities. We considered the basic solution protocol for decontaminating an arbitrary network, MRDE; we investigated its behaviour, properties and performance through an extensive number of computer simulation runs; in the investigation we used as a term of comparison the natural "randomized" strategy, RANE.

The simulation results have disclosed many interesting behaviour and properties of the solution protocol. In particular,

a. With regard to the influence of connectivity density on complexity, we demonstrate that:

- Move cost gradually increases to maximum at 40%-60% connectivity levels, then gradually decreases.
- Time cost gradually increase to maximum at middle connectivity levels, then gradually decreases.
- Team size continuously increases with the increase of connectivity. It reaches the maximum, i.e., graph size, at 100% connectivity level.
- MRDE and RANE differentiate in low connectivity levels, but gradually get close at high connectivity. At 100% connectivity, the behaviours are exactly the same.

b. With regard to the influence of graph size on complexity, we demonstrate that:

- The move costs seem to increase quadratically with the graph sizes.
- The execution time seems to increase linearly with the graph sizes.
- Team size all increases when graph size increases.

These results constitute the first experimental investigation of solution protocols for the Black Virus decontamination problem. Indeed, simulation is a valuable tool to gain an insight into the nature of the problem. Future work could be: 1) some improvements on DisJ to automatically process a batch of graphs and collect results; 2) some simulation work on systems with multiple black virus.

Acknowledgment

The author would like to express his sincere appreciation to Professor Paola Flocchini and Nicola Santoro for their enthusiastic guidance and advice throughout this research.

References

[1] P. Flocchini and N. Santoro, "Distributed Security Algorithms by Mobile Agents," *Distributed Computing and Networking, Lecture Notes in Computer Science*, 2011, pp. 1-14.
[2] L. Barrière, P. Flocchini, F. V. Fomin, P. Fraignaud, N. Nisse, N. Santoro, and D. M. Thilikos, "Connected graph searching," *Information and Computation*, 2012, pp. 1-16.
[3] L. Barrière, P. Flocchini, P. Fraignaud, and N. Santoro, "Capture of an intruder by mobile agents," *14th Symposium on Parallel Algorithms and Architectures* (SPAA), 2002, pp. 200-209.
[4] L. Blin, P. Fraignaud, N. Nisse, and S. Vial, "Distributed chasing of network intruders," *Theoretical Computer Science*, 2006, pp. 70-84.
[5] D. Dereniowski. "Connected searching of weighted trees," *Theoretical Computer Science*, 2011, pp. 5700-5713.
[6] P. Flocchini, M. J. Huang, and F. L. Luccio, "Decontamination of hypercubes by mobile agents," *Networks*, 2008, pp. 167-178.
[7] P. Flocchini, M. J. Huang, and F. L. Luccio, "Decontaminating chordal rings and tori using mobile agents," *International Journal on Foundations of Computer Science*, 2006, pp. 547-563.
[8] P. Flocchini, F. L. Luccio, and L. X. Song, "Size optimal strategies for capturing an intruder in mesh networks," *International Conference on Communications in Computing* (CIC), 2005, pp. 200-206.
[9] F. V. Fomin, D. M. Thilikos, and I. Todineau, "Connected graph searching in outerplanar graphs," *7th International Conference on Graph Theory* (ICGT), 2005, pp. 213216.
[10] N. Imani, H. Sarbazi-Azad, A. Y. Zomaya, and P. Moinzadeh, "Detecting threats in star graphs," *IEEE Transactions on Parallel and Distributed Systems*, 2009, pp. 474-483.
[11] F. Luccio and L. Pagli, "A general approach to toroidal mesh decontamination with local immunity," *23rd IEEE International Parallel and Distributed Processing Symposium* (IPDPS), 2009, pp. 1-8.
[12] F. Luccio, L. Pagli, and N. Santoro, "Network decontamination in presence of local immunity," *International Journal of Foundation of Computer Science*, 2007, pp. 457–474.
[13] F. L. Luccio, "Contiguous search problem in Sierpinski graphs," *Theory of Computing Systems*, 2009, pp. 186-204.

[14] N. Nisse, "Connected graph searching in chordal graphs," *Discrete Applied Mathematics*, 2009, pp. 2603-2610

[15] P. Shareghi, H. Sarbazi-Azad, and N. Imani, "Capturing an Intruder in the pyramid," *International Computer Science Symposium in Russia* (CSR), 2006, pp. 580-590.

[16] B. Yanga, D. Dyerb, and B. Alspach, "Sweeping graphs with large clique number," *Discrete Mathematics*, 2009, pp. 5770-5780.

[17] J. Chalopin, S. Das, A. Labourel, and E. Markou, "Tight Bounds for Scattered Black Hole Search in a Ring," *Lecture Notes in Computer Science*, 2011, pp. 186-197.

[18] J. Chalopin, S. Das, A. Labourel, and E. Markou, "Black hole search with finite automata scattered in a synchronous torus," *25th International Symposium on Distributed Computing (DISC)*, 2011, pp. 432-446.

[19] C. Cooper, R. Klasing, and T. Radzik, "Searching for black-hole faults in a network using multiple agents," *10th International Conference on Principles of Distributed Systems (OPODIS)*, 2006, pp. 320-332.

[20] J. Czyzowicz, S. Dobrev, R. Královic, S. Miklík, and D. Pardubská, "Black Hole Search in Directed Graphs," *16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2009, pp. 182-194.

[21] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc, "Searching for a black hole in synchronous tree networks," *Combinatorics, Probability & Computing*, 2007, pp. 595-619.

[22] S. Dobrev, P. Flocchini, R. Královic, P. Ruzicka, G. Prencipe, and N. Santoro, "Black hole search in common interconnection networks," *Networks*, 2006, pp. 61-71.

[23] S. Dobrev, P. Flocchini, R. Královic, and N. Santoro, "Exploring an unknown dangerous graph using tokens," *Theoretical Computer Science*, 2013, pp. 2845.

[24] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, "Searching for a black hole in arbitrary networks: Optimal mobile agents protocols," *Distributed Computing*, 2006, pp. 1-18.

[25] S. Dobrev, P. Flocchini, and N. Santoro, "Mobile search for a black hole in an anonymous ring," *Algorithmica*, 2007, pp. 67-90.

[26] S. Dobrev, N. Santoro, and W. Shi, "Using scattered mobile agents to locate a black hole in a unoriented ring with tokens," *Int. Journal of Foundations of Computer Science*, 2008, pp. 1355-1372.

[27] P. Glaus, "Locating a black hole without the knowledge of incoming link," *5th Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSOR)*, 2009, pp. 128-138.

[28] R. Klasing, E. Markou, T. Radzik, and F. Sarracco, "Approximation bounds for black hole search problems," *Networks*, 2008, pp. 216-226.

[29] W. Shi, "Black hole search with tokens in interconnected networks," *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2009, pp. 670-682.

[30] J. Cai, P. Flocchini, and N. Santoro, "Network Decontamination from a Black Virus," *27th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), Boston (MA), 2013, pp. 696-705.

[31] J. Cai, P. Flocchini, and N. Santoro, "Decontaminating a Network From a Black Virus," *International Journal of Networking and Computing* 2014, pp. 151-173.

[32] J. Cai, P. Flocchini, and N. Santoro, "Black Virus Decontamination in Arbitrary Networks," *The 3rd World Conference on Info Systems and Technologies* (WorldCIST'15), Azores, Portugal, 2015, pp. 991-1000.

[33] M. Yubao, and D. Renyuan, "Mobile Agent technology and Its Application in Distributed Data Mining," *First International Workshop on Database Technology and Applications*, 2009, pp. 151-155.

[34] M. Kona, and Z. Xu, "A framework for network management using mobile agents," *27th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), Florida, 2002, pp. 15-19.

[35] M. G. Rubinstein, and O. C. M. B. Duarte, "Evaluating the performance of mobile agents in network management," *Global Telecommunications Conference*, GLOBECOM, 1999, pp. 386-390.

[36] K. A. Amin, and A. R. Mikler, "Agent-based distance vector routing: a resource efficient and scalable approach to routing in large communication networks," *Journal of Systems and Software*, 2004, pp. 215227.

[37] R. Olfati-Saber, and R. M. Murray, "Consensus Protocols for Networks of Dynamic Agents," *Proc. Amer. Control Conf.*, 2003, pp. 951956.

[38] R. Olfati-Saber, and R. M. Murray, "Consensus Problems in Networks of Agents With Switching Topology and Time-Delays," *IEEE Transactions on Automatic Control*, 2004, pp. 1520-1533.

[39] N. Minar, K. Kramer, and P. Maes, "Cooperating Mobile Agents for Mapping Networks," *Proceedings of the First Hungarian National Conference on Agent Based Computing*, 1999, pp. 287-304.

[40] W. Schreiner, "A java toolkit for teaching distributed algorithms," *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, 2002, pp. 111-115.

[41] I. Chatzigiannakis, A. Kinalis, A. Poulakidas, G. Prasinos, and C. Zaroliagis, "DAP: A Generic Platform for the Simulation of Distributed Algorithms," *Proceedings of the 37th Annual Symposium on Simulation*, ANSS, 2004, pp. 166-177.

[42] Distributed Computing Group, "SinAlgo - Simulator for Network Algorithms," *http://www.disco.ethz.ch/projects/sinalgo/*, August 2015

[43] R. Oechsle, and T. Gottwald, "DisASTer (distributed algorithms simulation terrain): a platform for the implementation of distributed algorithms," *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '05, 2005, pp. 44-48.

[44] M. Ji, and M. Egerstedt, "Distributed Coordination Control of Multiagent Systems While Preserving Connectedness," *IEEE Transactions on Robotics*, 2007, pp. 693-703.

[45] M. Ben-Ari, "Interactive Execution of Distributed Algorithms," *ACM Journal of Education Resources in Computing*, 2001, article 2.

[46] N. Santoro, "Development of a Simulation Engine for Distributed Algorithms System Manual," *Internal Report*, Carleton University, 2001.