1

Computing with Mobile Agents in Distributed Networks

Evangelos Kranakis
School of Computer Science, Carleton University, Ottawa, ON, Canada

Danny Krizanc
Department of Mathematics and Computer Science, Wesleyan University, Middletown, Connecticut 06459, USA

Sergio Rajsbaum
Instituto de Matemáticas, Universidad Nacional Autónoma de México (UNAM), Ciudad Universitaria, D. F. 04510, Mexico

1.1 Introduction

Mobile agents are software entities with the capacity for motion that can act on behalf of their user with a certain degree of autonomy in order to accomplish a variety of computing tasks. Today they find applications in numerous computer environments such as operating system daemons, data mining, web crawlers, monitoring and surveillance, just to mention a few. Interest in mobile agents has been fueled by two overriding concerns. First, to simplify the complexities of distributed computing, and second to overcome the limitations of user interfaces.

1.1.1 What is a mobile agent?

It is not easy to come up with a precise definition of mobile agent. A variety of formulations exist ranging from the naive to the sophisticated. According to Shoham [Sho97], in software engineering a mobile agent is *a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes.* According to Bradshaw [Bra97] *an agent that inhabits an environment with other agents and processes is expected to be able to communicate and cooperate with them, and perhaps move from place to place in doing so.* Nwana et al. [NN97] attempt to divide mobile agents in categories of either simple (cooperative, learning, autonomous) or composite (collaborative, collaborative learning, interface, smart) behavior.

0-8493-8597-0/01/$0.00+$1.50

An alternative approach is to define mobile agents by associating attributes and identifying properties that agents are supposed to be able to perform. Some of these properties may include (see Wooldridge [Woo02, Woo99]) *reactivity:* the ability to selectively sense and act, *autonomy:* goal-directedness, proactive and self-starting behavior, *collaborative behavior:* working in concert with other agents to achieve a common goal, *adaptivity:* being able to learn and improve with experience, and *mobility:* being able to migrate in a self-directed way from one host platform to another.

In this chapter we examine mobile agents from the perspective of traditional research on distributed algorithms [Tel99]. We attempt to provide a framework for and a short survey of the recent research on the *theory of mobile agent computing.*

1.1.2 Outline of the chapter

An outline of the chapter is as follows. Section 1.2 describes the mobile agent model being used and the distributed network they operate on. Section 1.3 focuses on the rendezvous problem and addresses its solvability in the symmetric and asymmetric cases of rendezvous. Section 1.4 concerns graph exploration. Section 1.5 surveys work on the problem of searching with uncertainty where agents are given faulty advice from the nodes during the search process. Section 1.6 is on game-theoretic approaches to search and rendezvous. Section 1.7 focuses on intrusion detection and avoidance by studying the decontamination problem and black hole search.

1.2 Modeling Mobile Agents in Distributed Networks

1.2.1 Mobile agents

We are interested in modeling a set of software entities that act more or less autonomously from their originator and have the ability to move from node to node in a distributed network maintaining some sort of state with the nodes of the network providing some amount of (possibly longterm) storage and computational support. Either explicitly or implicitly such a mobile (software) agent has most often been modeled using a finite automaton consisting of a set of states and a transition function. The transition function takes as input the agent's current state as well as possibly the state of the node it resides in and outputs a new agent state, possible modifications to the current node's state and a possible move to another node. In some instances we consider probabilistic automata which have available a source of randomness that is used as part of their input. Such agents are referred to as *randomized* agents.

An important property to consider is whether or not the agents are distinguishable, i.e., if they have distinct labels or identities. Agents without identities are referred to as *anonymous* agents. Anonymous agents are limited to running precisely the same program, i.e., they are identical finite automata. As the identity is assumed to be part of the starting state of the automaton, agents with identities have the potential to run different programs.

The knowledge the agent has about the network it is on and about the other agents can make a difference in the solvability and efficiency of various tasks. For example, knowledge of the size of the network or its topology or the number of and identities of the other agents may be used as part of an agent's program. If available to the agents, this information is assumed to be part of its starting state. (One could imagine situations where the information is made available by the nodes of the network and not necessarily encoded in the agent.)

Other properties that may be considered in mobile agent computing include whether or not the agents have the ability to "clone" themselves, whether or not they have the ability to "merge" upon meeting (sometimes referred to as "sticky" agents) or whether or not they can send self-generated messages. At this point, most of the theoretical research on mobile agents ignores these properties and they will not be discussed below.

1.2.2 Distributed networks

The model of a distributed network is essentially inherited directly from the theory of distributed computing [Tel99]. We model the network by a graph whose vertices comprise the computing nodes and edges correspond to communication links.

The nodes of the network may or may not have distinct identities. In an *anonymous* network the nodes have no identities. In particular this means that an agent can not distinguish two nodes except perhaps by their degree. The outgoing edges of a node are usually thought of as distinguishable but an important distinction is made between a globally consistent edge-labeling versus a locally independent edge-labeling. A simple example is the case of a ring where clockwise and counterclockwise edges are marked consistently around the ring in one case, and the edges are arbitrarily - say by an adversary - marked 1 and 2 in the other case. If the labeling satisfies certain coding properties it is called a *sense of direction* [FMS98]. Sense of direction has turned out to greatly effect the solvability and efficiency of solution of a number of problems in distributed computing and has been shown to be important for mobile agent computing as well.

Networks are also classified by how they deal with time. In a synchronous network there exists a global clock available to all nodes. This global clock is inherited by the agents. In particular it is usually assumed that in a single step an agent arrives at a node, performs some calculation, and exits the node and that all agents are performing these tasks "in sync". In an asynchronous network such a global clock is not available. The speed with which an agent computes or moves between nodes, while guaranteed to be finite, is not a priori determined.

Finally we have to consider the resources provided by the nodes to the agents. All nodes are assumed to provide enough space to store the agent temporarily and computing power for it to perform its tasks. (The case of malicious nodes refusing agents or even worse destroying agents - so-called *black holes* - is also sometimes considered.) Beyond these basic services one considers nodes that might provide some form of long-term storage, i.e., state that is left behind when the agent leaves. In the rendezvous problem the idea of leaving an indistinguishable mark or *token* at a node (introduced in [BG01]) has been studied. In graph exploration a similar notion of a *pebble* is used[BS94]. More accommodating nodes might provide a *whiteboard* for agents to write messages to be left for themselves or for other agents.

1.3 The Rendezvous Problem

The mobile agent rendezvous problem is concerned with how should mobile agents move along the vertices of a given network in order to optimize the number of steps required for all of them to meet at the same node of the network. Requiring such agents to meet in order to synchronize, share information, divide up duties, etc. would seem to be a natural fundamental operation useful as a subroutine in more complicated applications such as web-crawling, meeting scheduling, etc. For example, rendezvous is recognized as an effective paradigm for realizing the caching of popular information essential to the operation of a P2P network and thus reducing network traffic [ws].

In this section, we provide a short survey of recent work done on rendezvous within the distributed computing paradigm. We note that rendezvous has been studied in other settings such as robotics [RD01] and operations research [AG03]. This research is extensive and many of the solutions found can be applied here. But it is often the case that the models used and the concerns studied are sufficiently different as to require new approaches.

1.3.1 Solvability of rendezvous

Given a particular agent model (e.g., deterministic, anonymous agents with knowledge they are on a ring of size n) and network model (e.g., anonymous, synchronous with tokens) a set of k agents distributed arbitrarily over the nodes of the network are said to *rendezvous* if after running their programs after some finite time they all occupy the same node of the network at the same time. It is generally assumed that two agents occupying the same node can recognize this fact (though in many instances this fact is not required for rendezvous to occur). As stated, rendezvous is assumed to occur at nodes. In some instances one considers the possibility of rendezvous on an edge, i.e., if both agents use the same edge (in opposite directions) at the same time. (For physical robots this makes sense. For software agents this perhaps is not so realistic but sometimes necessary to allow for the possibility of rendezvous at all - especially in instances where the network lacks a sense of direction.)

The first question one asks for an instance of rendezvous is whether or not it is solvable. There are many situations where it is not possible to rendezvous at all. This will depend upon both the properties of the agents (deterministic or randomized, anonymous or with identities, knowledge of the size of the network or not, etc.) and the network (synchronous or asynchronous, anonymous or with identities, tokens available or not, etc.). The solvability is also a function of the starting positions chosen for the agents. For example, if the agents start at the same node and can recognize this fact, rendezvous is possible in this instance. Given a situation where some starting positions are not solvable (i.e., rendezvous is not possible) but others are, we distinguish between algorithms that are guaranteed to finish for all starting positions, with successful rendezvous when possible but otherwise recognizing that rendezvous is impossible, versus algorithms that are only guaranteed to halt when rendezvous is possible. Algorithms of the former type are said to solve *rendezvous with detection.* (The distinction is perhaps analogous to Turing machines deciding versus accepting a language.)

For solvable instances of rendezvous one is interested in comparing the efficiency of different solutions. Much of the research focuses on the time required to rendezvous. In the synchronous setting the time is measured via the global clock. (In some situations, it makes a difference if the agents begin their rendezvous procedure at the same time or there is possible delay between start times.) In the asynchronous setting we adapt the standard time measures from the distributed computing model. Also of interest is the size of the program required by the agents to solve the problem. This is referred to as the memory requirement of the agents and is considered to be proportional to the base two logarithm of the number of states required by the finite state machine encoding the agent.

As is often the case, researchers are interested in examining the extremes in order to get an understanding of the limits a problem imposes. Over time it has become clear that for rendezvous symmetry (of the agents and the network) plays a central role in determining its solvability and the efficiency of its solutions. As such we divide our discussion below into the asymmetric and symmetric cases. For simplicity we restrict ourselves to the case of just two agents in most of the discussion below.

1.3.2 Asymmetric rendezvous

Asymmetry in a rendezvous problem may arise from either the network or the agents.

Network asymmetry

A network is asymmetric if it has one or more uniquely distinguishable vertices. A simple example is the case of a network where all of the nodes have unique identities chosen from a subset of some totally ordered set such as the integers. In this case, the node labelled with the smallest identity (for example) is unique and may be used as a meeting point for a rendezvous algorithm. Uniqueness need not be conferred using node labels. For example, in a network where there is a unique node of degree one, it may be used as a focal point.

If a "map" of the graph with an agent's starting position marked on it is available to the agents then the problem of rendezvous is easily solved by just traversing the path to an agreed upon unique node. Algorithms that use an agreed upon meeting place are referred to by Alpern and Gal [AG03] as FOCAL strategies. In the case where the graph is not available in advance but the agents know that a focal point exists (e.g., they know the nodes are uniquely labeled and therefore there exists a unique minimum label node) this strategy reduces to the problem of graph traversal or graph exploration whereby all of the nodes (sometimes edges) of the graph are to be visited by an agent. This has been extensively studied and is surveyed in the Section 1.4.

Agent asymmetry

By agent asymmetry one generally means the agents have unique identities that allow them to act differently depending upon their values. In the simplest scenario of two agents, the agent with the smaller value could decide to wait at its starting position for the other agent to find it by exploring the graph as above. Alpern and Gal [AG03] refer to this as the Wait For Mommy (WFM) strategy and they show it to be optimal under certain conditions.

WFM depends upon the fact that the agents know in advance the identities associated with the other agents. In some situations this may be an unrealistic assumption. Yu and Yang [YY96] were the first to consider this problem. Under the assumption that the algorithm designer may assign the identities to the agents (as well as the existence of distinct whiteboards for each agent), they show that rendezvous may be achieved deterministically on a synchronous network in $O(nl)$ steps where n is the size of the network and l is the size of the identities assigned. The perhaps more interesting case where an adversary assigns the labels was first considered in [DFP03]. Extensions to this work including showing rendezvous on an arbitrary graph is possible in time polynomial in n and l and that there exist graphs requiring $\Omega(n^2)$ time for rendezvous are described in [KP04, KM06]. The case of an asynchronous network is considered in [MGK$^+$05] where a (non-polynomial) upper bound is set for rendezvous in arbitrary graphs (assuming the agents have an upper bound on the size of the graph). Improvements (in some cases optimal) for the case of the ring network are discussed in each of the above papers.

1.3.3 Symmetric rendezvous

In the case of symmetric rendezvous, both the (generally synchronous) network and the agents are assumed to be anonymous. Further one considers classes of networks that in the worst case contain highly symmetric networks that do not submit to a FOCAL strategy. As might be expected some mechanism is required to break symmetry in order for rendezvous to be possible. The use of randomization and of tokens to break symmetry have both been studied extensively.

Randomized rendezvous

Many authors have observed that rendezvous may be solved by anonymous agents on an anonymous network by having the agents perform a random walk. The expected time to rendezvous is then a (polynomial) function of the (size of the) network and is directly related to the cover time of the network. (See [MR95] for definitions relating to random walks.)

For example, it is straightforward to show that two agents performing a symmetric random walk on ring of size n will rendezvous within expected $O(n^2)$ time. This expected time can be improved by considering the following strategy (for a ring with sense of direction). Repeat the following until rendezvous is achieved: flip a (fair) coin and walk $n/2$ steps to the right if the result is heads, $n/2$ steps to the left if the result is tails. If the two agents choose different directions (which they do with probability $1/2$) then they will rendezvous (at least on an edge if not at a node). It is easy to see that expected time until rendezvous is $O(n)$. Alpern refers to this strategy as Coin Half Tour and studies it in detail in [Alp95a]. Note that the agents are required to count up to n and thus seem to require $O(\log n)$ bits of memory to perform this algorithm (whereas the straightforward random walk requires only a constant number of states to implement). This can be reduced to $O(\log \log n)$ bits and this can be shown to be tight [Mor] for achieving linear expected rendezvous time.

Rendezvous using tokens

The idea of using tokens or marks to break symmetry for rendezvous was first suggested in [BG01] and expanded upon for the case of the ring in [Saw04]. The first observation to make is that rendezvous is impossible for deterministic agents with tokens (or whiteboards) on an even size ring when the agents start at distance $n/2$ as the agents will remain in symmetric positions indefinitely. However, this is the only starting position for the agents for which rendezvous is impossible. This leads one to consider algorithms for rendezvous with detection where rendezvous is achieved when possible and otherwise the agents detect they are in an impossible to rendezvous situation. In this case, a simple algorithm suffices (described here for the oriented case). Each agent marks their starting position with a token. They then travel once around the ring counting the distances between their tokens. If the two distances are the same, they halt declaring rendezvous impossible. If they are different they agree to meet (for example) in the middle of the shorter side.

Again, one observes that the algorithm as stated requires $O(\log n)$ bits of memory for each agent in order to keep track of the distances. Interestingly enough this can be reduced to $O(\log \log n)$ bits and this can be shown to be tight for unidirectional algorithms [KKSS03]. If we are allowed two movable tokens (i.e., the indistinguishable marks can be erased and written with at most two marks total per agent at any time) then rendezvous with detection becomes possible with an agent with constant size memory [KKM06].

Multi-agent rendezvous, i.e., more than two agents, on the ring is considered in [FKK$^+$04b, GKKZ06], the second reference establishing optimal memory bounds for the problem. Two agent rendezvous on the torus is studied in [KKM06] where tradeoffs between memory and the number of (movable) tokens used are given. Finally [FKK$^+$04a] considers a model in which tokens may disappear or fail over time.

Rendezvous and leader election

Consider k identical, asynchronous mobile agents on an arbitrary anonymous network of n nodes. Suppose that all the agents execute the same protocol. Each node has a whiteboard where the agents can write to and read from. *Leader election* refers to the problem of electing a leader among those agents.

In Barrierre et al. [BFFS03] both the leader election and rendezvous problems are studied and shown to be equivalent for networks with appropriate sense of direction. For example, rendezvous and election are unsolvable (i.e., there are no deterministic generic solutions) if $\gcd(k, n) > 1$, regardless of whether or not the edge-labeling is a sense of direction. On the other hand, if $\gcd(k, n) = 1$ then the initial placement of the mobile agents in the network creates topological asymmetries that can be exploited to solve the problems.

1.4 Graph Exploration

The problem

The main components of the graph exploration problem are a connected graph $G = (V, E)$ with V as its set of nodes and E as its set of links, as well as a unique start node s. The graph represents a distributed network, the nodes of which are autonomous processing elements with limited storage while s is the initial position of the mobile agent. The goal of graph exploration is for the agent (or team of agents) to visit all of the nodes (or possibly edges) of the graph. Apart from completing the exploration of the graph it is often required that the agent also draw a map of the graph. The core of graph exploration often involves the construction of a simpler underlying subgraph (typically some kind of spanner of the graph, like a tree) that spans the original network G. Some of the important constructions of such trees involve breadth-first search (BFS), depth-first search (DFS), and minimum spanning trees (MST). For a detailed discussion of distributed constructions of BFS, DFS and MST the reader is advised to consult Peleg [Pel00].

The performance of graph exploration algorithms is limited by the static and non-adaptive nature of the knowledge incorporated in the input data. Mobile agents can take sensory input from the environment, learn of potential alternatives, analyze information collected, possibly undertake local action consistent with design requirements and report it to an authorized source for subsequent investigation and adaptation.

Efficiency measures

There is a variety of efficiency measures that can be considered. Time is important when dealing with exploration and many of the papers mentioned below measure it by the number of edge traversals required by the agent in order to complete the task. On the other hand, they do not impose any restrictions on the memory of the agent, which can be an important consideration in applications. This of course gives rise to interesting time/memory trade-offs. In Diks et al. [DFKP04] the problem of minimizing the memory of the agent is studied.

Underlying graph

Exploration and navigation problems for agents in an unknown environment have been extensively studied in the literature (see the survey Rao et al. [RHSI93]). The underlying graph can be directed (e.g., in [AH00, BFR^{+}98, BS94, DP99] the agent explores strongly connected directed graphs and it can move only in the direction from head to tail of an edge, not vice-versa) or undirected (e.g., in [ABRS95, BRS95, DKK01, PP99] the explored graph is undirected and the agent can traverse edges in both directions).

Labeling of the graph

In graph exploration scenarios it can be assumed either that the nodes of the graph have unique labels recognizable by the agent or that they are anonymous. Exploration in directed graphs with such labels was investigated in [AH00, DP99]. Exploration of undirected labeled graphs was studied in [ABRS95, BRS95, DKK01, PP99]. In this case a simple exploration based on DFS can be completed in time $2e$, where e is the number of edges of the underlying graph. For an unrestricted agent an exploration algorithm working in time $e + O(n)$, with n being the number of nodes, was proposed in Panaite et al. [PP99]. Restricted agents were investigated in [ABRS95, BRS95, DKK01]. It was assumed that the agent has either a restricted tank [ABRS95, BRS95], forcing it to periodically return to the base for refueling, or that it is tethered, i.e., attached to the base by a rope or cable of restricted length Duncan et al. [DKK01]. It was proved that exploration and mapping can be done in time $O(e)$ under both scenarios.

Anonymity

Suppose that a mobile agent has to explore an undirected graph by visiting all its nodes and traversing all edges, without any a priori knowledge either of the topology of the graph or of its size. This can be done easily by depth-first search if nodes and edges have unique labels. In some navigation problems in unknown environments such unique labeling either may not be available or simply the mobile agents cannot collect such labels because of limited sensory capabilities. Hence it is important to be able to program the mobile agent to explore graphs anonymously without unique labeling of nodes or edges. Arbitrary graphs cannot be explored under such weak assumptions. For example, an algorithm on a cycle of unknown size without any labels on nodes and without the possibility of putting marks on them cannot terminate after complete exploration.

Hence, [BFR$^+$98, BS94] allow *pebbles* which the agent can drop on nodes to recognize already visited ones, and then remove them and drop in other places. They focus their attention on the minimum number of pebbles allowing for efficient exploration and mapping of arbitrary directed n-node graphs. (In the case of undirected graphs, one pebble suffices for efficient exploration.) In was shown in Bender et al. [BFR$^+$98] that one pebble is enough if the agent knows an upper bound on the size of the graph, and $\Theta(\log \log n)$ pebbles are necessary and sufficient otherwise. Also a comparison of the exploration power of one agent versus two cooperating agents with a constant number of pebbles is investigated in Bender et al. [BS94] (Recall that a similar comparison for the rendezvous problem in a torus is investigated in Kranakis et al. [KKM06]).

Anonymous tree exploration

If we do not allow any marks it turns out that the class of graphs that can potentially be explored has to be restricted to trees, i.e., connected graphs without cycles. Another important requirement is that the mobile agent be able to distinguish ports at a node, otherwise it would be impossible to explore even a star tree with three leaves (after visiting the second leaf the agent cannot distinguish the port leading to the first visited leaf from that leading to the unvisited one). Hence a natural assumption is that all ports at a node be locally labeled 1,...,d, where d is the degree of the node, although no coherence between those local labelings is required.

Exploration algorithms and solutions discussed in the sequel are from Diks et al. [DFKP04] where additional details can be found. Two exploration methodologies are considered. The results are summarized in Table 1.1. First, *exploration with stop:* starting at any node of the

TABLE 1.1 Upper and lower bounds on the memory of a mobile agent for various types of exploration in trees on n nodes and maximum degree d.

Tree	Mobile Agent	Memory Bounds	
Exploration	Knowledge	Lower	Upper
Perpetual	$\emptyset$	None	$O(\log d)$
w/Stopping	$n \le N$	$\Omega(\log\log\log n)$	$O(\log N)$
w/Return	$\emptyset$	$\Omega(\log n)$	$O(\log^2 n)$

tree, the mobile agent has to traverse all edges and stop at some node. Second, *exploration with return:* starting at any node of the tree, the mobile agent has to traverse all edges and stop at the starting node. In both instances it is of interest to find algorithms for a mobile agent performing the given task using as little memory as possible. From the point of view of memory use, the most demanding task of exploration with stop is indeed stopping. This is confirmed by analyzing *perpetual exploration* whereby the mobile agent has to traverse all edges of the tree but is not required to stop. The following simple algorithm will traverse all edges of an n-node tree after at most $2(n-1)$ steps and will perform perpetual exploration using only $O(\log d)$ memory bits. The agent leaves the starting node by port 1. After entering any node of degree d by port i, the agent leaves it by port $i+1 \bmod d$. If in addition we know an upper bound N on the size n of the tree, we can explore it with stop using $O(\log N)$ bits of memory.

For every agent there exists a tree of maximum degree 3 which this agent cannot explore with stop. Even more so, it can be shown that a agent which can explore with stop any n-node tree of maximum degree 3 must have $\Omega(\log\log\log n)$ bits of memory. Additional memory is essentially needed to decide when to stop in an anonymous environment. Exploration with stopping is even more demanding on memory. For exploration with return, a lower bound $\Omega(\log n)$ can be shown on the number of memory bits required for trees of size n. As for upper bounds, an efficient algorithm for exploration with return is given that uses only $O(\log^2 n)$ memory bits for all trees of size n.

1.5 Searching with Uncertainty

Another problem of interest in mobile agent computing is that of searching for an item in a distributed network in the presence of uncertainty. We assume that a mobile agent in a network is interested in locating some item (such as a piece of information or the location of a service) available at one node in the network. It is assumed that each of the nodes of the network maintains a database indicating the first edge on a shortest path to the items of interest. Uncertainty arises from the fact that inaccuracies may occur in the database for reasons such as the movement of items, out-of-date information, etc. The problem is how to use the inaccurate databases to find the desired item. The item occupies an unknown location but information about its whereabouts can be obtained by querying the nodes of the network. The goal of the agent is to find the requested item while traversing the minimum number of edges. In the sequel we discuss algorithms for searching in a distributed network under various models for (random) faults and different topologies.

Network and search models

The network has n nodes and is represented as a connected, undirected graph $G = (V, E)$ with V its set of nodes and E its set of links. The universe of items of interest is denoted by S. Let $Adj(x)$ be the set of links adjacent to node x in the network. For $x \in V$ and $v \in S$

define $S_v(x)$ to be the set of links adjacent to x which are on a shortest path from x to the node containing v. Clearly, $\emptyset \subset S_v(x) \subseteq Adj(x)$, for all x, v. Let $1 \geq p > \frac{1}{2}$ be constant.

Among the several possible search models (see [KK99]) we consider only two. Under the *single shortest path* (SSP) model, we assume that for each item $v \in S$ a tree representing a single shortest path from each node in V to the node containing v has been defined. If $e \in S_v(x)$ is the edge chosen in this shortest path tree then e appears in the entry for v in the database of x with probability greater than or equal to p. The probability the entry is some other $e' \in Adj(x) \setminus \{e\}$ is less than or equal to $1-p$. Furthermore, for each v and for each x and x' these events are independent, i.e., an error occurs in x's entry for v with probability at most $1-p$ independent of errors occurring in the v entry of any other nodes, where by an error we mean that the node reports an edge other than the shortest path tree edge when queried about v. Note the "error" may in fact be an edge on a different shortest path to v and therefore may not be an error at all. The *SSP with global information* is a submodel of the SSP model. Here, some global information about how the shortest path trees were originally constructed is available to the search algorithm. For example, on a mesh we may assume that the original databases, before the introduction of errors, were constructed using the standard row column shortest path trees and this is used in the construction of the searching algorithm. Under the *all shortest paths* (ASP) model, the probability that the entry for v in the routing table of x is some $e \in S_v(x)$ is greater than or equal to p and the probability it is some $e \in Adj(x) \setminus S_v(x)$ is less or equal to $1-p$. Again errors at distinct nodes (in this case reporting an edge in $Adj(x) \setminus S_v(x)$) occur independently.

Note that in both models, no restriction is placed upon what is reported by the database in the case of an error, i.e., in the worst case the errors may be set by a malicious adversary. In the ASP model, no restriction is placed on which of the correct edges is reported, i.e., again, the answers to queries may be set by a malicious adversary. Finally, we note that for any geodesic (unique shortest path) network the two models are the same, e.g., trees, or rings of odd size.

Ring network

For the n node ring we give below a searching algorithm in the SSP or ASP model which with high probability finds a given item in $d + O(\log n)$ steps where d is the distance from the starting node to the node containing the desired item. Note that for $d = \omega(\log n)$ this is optimal to within an additive term.

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

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

TABLE 1.2 Expected number of steps.

	memoryless	limited memory	unlimited memory
randomized	$n-1$	$2/p$	$1/p$
deterministic	∞	∞	$1/p$

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

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

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

Observe that $p_k = \Pr[S_k \geq k/2]$ and using Chernoff-Hoeffding bounds (setting $\mu = kp$ and $\delta = 1 - \frac{1}{2p}$) we derive that

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

where S_k is the sum of k independent and identically distributed random variables $X_1, \ldots, X_k$ such that $\Pr[X_i = 1] = p$, for all i. Choosing $k \geq c\frac{2p}{(p-1/2)^2} \ln n$, we conclude the probability that the algorithm requires less than $d + 2k$ steps is $\geq 1 - n^{-c}$ for $c > 0$.

A similar approach can be followed for the torus network. For additional details the reader is advised to consult [KK99].

Complete network

Next we consider the case of a complete network on n nodes. (The results of this section are mainly from Kirousis et al. [KKKS03].) The algorithms considered may use randomization or be deterministic, may be either *memoryless*, have *limited memory* or have *unlimited memory* and can execute one of the following operations: (i) query a node of the complete network about the location of the information node (ii) follow the advice given by a queried node and (iii) select the next node to visit using some probability distribution function on the network nodes (that may be, for example, a function of the number of previously seen nodes or the number of steps up to now). A summary of the expected number of steps for various types of algorithms is summarized in table 1.2. The algorithm given below, simply alternates between following the advice of the currently visited node and selecting a random node as the next node to visit. It only needs to remember if at the last step it followed the advice or not. Although one bit suffices, the algorithm is stated as if it knew the number of steps it has taken. Then it simply checks if this number is odd or even. However, one bit would be sufficient.

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

```
2.      current = RANDOM($V$)
3.      $l \leftarrow 1$
4.      while current(information) $\neq$ true
5.          $l \leftarrow l + 1$
6.          if $l \bmod 2 = 0$
7.              current = current(advice)
8.          else
9.              current $\leftarrow$ RANDOM($V -$ current)
10.     end while
11. end
```

We will now give a randomized algorithm for locating the information node in a clique that has unlimited memory. More specifically, the algorithm can store the previously visited nodes and use this information in order to decide its next move. In this way, the algorithm avoids visiting again previously visited nodes. Such an algorithm always terminates within $n - 1$ steps in the worst case.

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

```
1.  begin
2.      $l \leftarrow 1$
3.      current = RANDOM($V$)
4.      $M \leftarrow$ {current} // $M$ holds the up to now visited nodes
5.      while current(information) $\neq$ true
6.          read(current(advice))
7.          if current $\notin M$
8.              $M \leftarrow M \cup$ {current}
9.              current = advice
10.         end if
11.         else
12.             current $\leftarrow$ RANDOM($V - M$)
13.         $l \leftarrow l + 1$
14.     end while
15. end
```

For additional details the reader is advised to consult Kirousis et al. [KKKS03] and Kaporis et al. [KKK$^+$01] For results in a model where the faults occurring worst-case deterministically see Hanusse et al. [HKKK02]

1.6 Game-theoretic Approaches

Search games

In general, search theory deals with the problem of optimizing the time it takes in order to find a target in a graph. The P2P paradigm demands a fully-distributed and cooperative network design, whereby nodes collectively form a system without any supervision. In such networks peers connect in an ad-hoc manner and the location of resources is neither controlled by the system nor are there any guarantees for the success of a search are offered to the users. This setup has its advantages as well, including anonymity, resource-sharing, self-organization, load balancing and robustness to failures. Therefore in this context search games might provide a more flexible basis for offering realistic and effective solutions for searching. There is extensive literature on games and game theory, however for our purposes it will be sufficient to refer the reader to Osborne et al. [OR94] for a mathematical treatment and to the specialized but related multi-agent perspective offered by Sandholm [San99].

Search and rendezvous models

As defined by Alpern and Gal [AG03], search games consist of a *search space* usually represented by a graph G (with vertex set V and edge set E) a *searcher* starting from a given vertex (usually called the origin) of the graph and a *hider* (mobile or otherwise) choosing its hiding point independently of the searcher and occupying vertices of the graph. It is assumed that neither the searcher nor the hider has any a priori knowledge of the movement or location of the other until they are some critical distance r apart, also called the *discovery radius.*

Search problems are defined as two-person, zero-sum games with associated strategies available to the searcher and receiver, respectively (see, for example, Watson [Wat02] or Osborne et al. [OR94]). Strategies may be pure or mixed (i.e., probabilistic) and the set of pure strategies for the searcher (respectively, hider) is denoted by $\mathcal{S}$ (respectively, $\mathcal{H}$). A strategy $S \in \mathcal{S}$ (respectively, $H \in \mathcal{H}$) for the searcher (respectively, hider) is a trajectory in the graph G such that $S(t)$ (respectively, $H(t)$) is the point visited by the searcher (respectively, at which the hider is hiding) at time t. (If the hider is immobile then it assumed that $H(t) := H$ is a fixed vertex in the given graph.)

The cost of the game is specified by a cost function $c : \mathcal{S} \times \mathcal{H} \to R : (S, H) \to c(S, H)$ representing the *loss* (or effort) by searcher S when using trajectory S while the hider uses trajectory H. Since the game is zero-sum, $c(S, H)$ also represents the *gain* of the hider. Given the sets of available pure strategies $\mathcal{S}, \mathcal{H}$, and the cost function $c(\cdot, \cdot)$ we define the *value*, *minimax value*, and *minimax search trajectory* as follows

$$\begin{array}{ll} \textit{value:} & v(S) := \sup_{H \in \mathcal{H}} c(S, H) \\ \textit{minimax value:} & \hat{V} := \inf_{S \in \mathcal{S}} v_c(S) \\ \textit{minimax search trajectory:} & \hat{S} \text{ s.t. } \hat{V} = v(\hat{S}), \end{array} \tag{1.1}$$

where mention of the cost function c is suppressed in Identities 1.1. A natural cost function c is the time it takes until the searcher captures the hider (also called *capture time*) in which case V_c represents the minimal capture time. Formally, the capture time is defined as

$$c(S, H) := \min_t \{d(S(t), H(t)) \leq r\},$$

where r is the discovery radius.

In most interesting search games the searcher can do better by using random choices out of the pure strategies and these choices are called *mixed strategies.* In this case the capture time is a random variable and each player cannot guarantee a fixed cost but rather only an expected cost. Mixed strategies of a searcher and a hider are denoted by s, h, respectively, while $c(s, h)$ denotes the expected cost. More formally, we are given probability distributions $s = \{p_S : S \in \mathcal{S}\}, h = \{q_H : H \in \mathcal{H}\}$ on $\mathcal{S}$ and $\mathcal{H}$, respectively, and the expected cost $c(s, h)$ is defined by

$$c(s, h) = \sum_{S \in \mathcal{S}} \sum_{H \in \mathcal{H}} c(S, H) p_S q_H. \tag{1.2}$$

As with pure stategies, we define

$$\begin{array}{ll} \textit{value of } s\textit{:} & v(s) := \sup_h c(s,h) = \sup_{H\in\mathcal{H}} c(s,H), \\ \textit{value of } h\textit{:} & v(h) := \inf_s c(s,h) = \inf_{S\in\mathcal{S}} c(S,h), \\ \textit{minimax value:} & v := \inf_s v(s) = \sup_h v(h). \end{array} \tag{1.3}$$

When $\mathcal{S}$ and $\mathcal{H}$ are finite, the fundamental theorem of Von Neumann (see von Neumann et al. [vNM53]) on two-person, zero-sum games applies and states that

$$\max_h \min_s c(s,h) = \min_s \max_h c(s,h).$$

(If only one of either $\mathcal{S}$ or $\mathcal{H}$ is finite then the minimax value can be proven to exist. If both are infinite the minimax value may not exist, but under certain conditions, like uniform convergence of trajectories, it can be proved to exist. See Alpern and Gal [AG03] for additional results.)

If the hider is immobile a pure hiding strategy in the search graph G is simply a vertex, while a mixed hiding strategy is a probability distribution on the vertices of G. A pure search strategy is a trajectory in G starting at the origin O of the graph. We denote by $X_S(t)$ the set of vertices of G which have been searched using strategy S by time t. Obviously, the set $X_S(0)$ of points discovered at time 0 does not depend on the strategy S but only on the discovery radius r. A mixed search strategy is a probability distribution over these pure search strategies.

The *rendezvous search problem* is also a search game but it differs from the search problem in that it concerns two searchers placed in a known graph that want to minimize the time required to rendezvous (usually) at the same node. At any given time the searchers may occupy a vertex of a graph and can either stay still or move from vertex to vertex. The searchers are interested in minimizing the time required to rendezvous. A detailed investigation of this problem has been carried out by Alpern and collaborators (see [Alp95b, Alp02]) and a full discussion can also be found in the book of Alpern and Gal [AG03]. As in search games we can define pure and mixed rendezvous strategies as well as the expected rendezvous time.

Example of a search game

As an example, consider the *uniform* mixed hiding strategy whereby the hider is immobile and chooses the hiding vertex randomly with the uniform distribution among the vertices of a graph with n vertices. Further assume that the discovery radius satisfies $r = 0$, which means that the searcher discovers the hider only when it lands in the same vertex. Since the searcher discovers at most one new vertex per time unit we must have that $\Pr[T \leq t] \leq \min\{1, \frac{t}{n}\}$, where T is the capture time. It follows that the expected capture time satisfies

$$E[T] = \sum_t \Pr[T > t] \geq \sum_t \max\left\{0, 1 - \frac{t}{n}\right\} = \frac{n+1}{2}.$$

To show this is tight, consider the complete graph K_n on n vertices. Let the hider strategy be uniform as before. A pure search strategy is a permutation $S \in S_n$. For a given S, H the capture time $c(S,H)$ is equal to the smallest t such that $S_t = H$. Given t, H, there are exactly $(n-1)!$ permutations $S \in S_n$ such that $S_t = H$. Consider a mixed search strategy whereby the searcher selects a permutation $S \in S_n$ at random with the uniform distribution. We have that

$$\sum_{H=1}^{n} \sum_{S\in S_n} c(S,H) = \sum_{H=1}^{n} \sum_{t=1}^{n} \sum_{\substack{S\in S_n \\ S_t = H}} t = \frac{n(n+1)}{2} \cdot (n-1)!.$$

Consequently, it follows that the expected capture time is

$$E[T] = \frac{1}{n!} \cdot \frac{1}{n} \cdot \sum_{H=1}^{n} \sum_{S \in S_n} c(S, H) = \frac{n+1}{2}.$$

The problem is more difficult on arbitrary networks. For example, on trees it can be shown that the optimal hiding strategy is to hide among leaves according to a certain probability distribution that is constructed recursively, while an optimal search strategy is a *Chinese postman tour* (i.e., a closed trajectory traversing all vertices of the graph which has minimum length). The value of this search game on trees is the minimum length of a Chinese postman tour. We refer the reader to Alpern and Gal [AG03] for a proof of this result as well as for search games on other networks.

Example of a rendezvous search game

Consider again the complete graph K_n. If searchers select their initial position at random the probability they occupy the same vertex is $1/n$. In a rendezvous strategy the players select trajectories (i.e., paths) $S = s_0, s_1, \ldots$ and $S' = s'_0, s'_1, \ldots$, where s_i, s'_i are the nodes occupied by the two searchers at time i, respectively. For a given pair S, S' of trajectories the rendezvous time is equal to the smallest t such that $s_t = s'_t$. Hence, the probability that they meet within the first t steps is at most t/n. The expected rendezvous time is equal to

$$E[T] = \sum_{t=0}^{\infty} \Pr[T > t] = \sum_{t=0}^{\infty} (1 - \Pr[T \le t]) \ge \sum_{t=0}^{n-1} \frac{n-t}{n} = \frac{n-1}{2}. \tag{1.4}$$

In general, the rendezvous problem is easy if there is a distinguished vertex since the searchers can rendezvous there. This possibility can be discounted by considering an appropriate symmetry assumption on the graph, e.g., the group A of vertex automorphisms is a transitive group. As suggested by Anderson et al. [AR90], it can be shown that the lower bound in Inequality 1.4 holds for any graph with a transitive group of automorphisms with essentially the same proof as above.

In the *asymmetric* version of the rendezvous game players are allowed to play different strategies. For example, assuming that the graph is hamiltonian, one of the searchers can stay stationary while the moving player can choose to traverse a hamiltonian path (this is also known as *wait for mammy* strategy), in which case it is not difficult to show that the expected meeting time is at most $\frac{n-1}{2}$, i.e., the lower bound in Inequality 1.4 is also an upper bound.

In the *symmetric* rendezvous game the players do not have the option of playing different strategies. A natural strategy without searcher cooperation, for example, is a simultaneous random walk. If the searchers cooperate they can build a spanning tree and then they can perform a preorder traversal on the this tree which can search all the nodes exhaustively in $2n$ steps. Now the searchers in each time interval of length $2n - 1$ search exhaustively or wait each with probability $1/2$. This scheme has expected rendezvous time $2(2n - 1)$. For the complete graph there is an even better strategy proposed by Anderson et al. [AR90] that has expected rendezvous time $\sim .82 \cdot n$. Certainly, there are interesting questions concerning other topologies as well as memory and expected rendezvous time tradeoffs. We refer the reader to Alpern and Gal [AG03] for additional studies.

1.7 Network Decontamination and Black Holes

Mobile agents have been found to be useful in a number of network security applications. Intrusion is the act of (illegally) *invading* or *forcing* into the premises of a system without right or welcome in order to compromise the confidentiality, integrity or availability of a resource. Computer systems need to be monitored for intrusions and their detection is a difficult problem in network and computer system security. Once detected it is important to remove the intruders or decontaminate the network. In this section we look at two different problems related to these issues. First we discuss a model for decontaminating a network efficiently, and second we discuss algorithms for detecting malicious nodes called black holes.

1.7.1 Network decontamination

In the graph searching problem a set of searchers want to catch a fugitive hidden in a graph. The fugitive moves with a speed that may be significantly higher than the speed of searchers, while the fugitive is assumed to have complete knowledge of the searchers' strategy. In an equivalent formulation, we are given a graph with *contaminated* edges and, via a sequence of steps using searchers, we want to arrive at a situation whereby all edges of the graph are simultaneously *clean*. The problem is to determine the minimum number of searchers sufficient to clean the graph and the goal is to obtain a state of the graph in which all edges are simultaneously clean. An edge is cleaned if a searcher traverses it from one of its vertices to the other. A clean edge is preserved from recontamination if either another searcher remains in a vertex adjacent to the edge, or all other edges incident to this vertex are clean. In other words, a clean edge e is recontaminated if there exists a path between e and a contaminated edge, with no searcher on any node of the path. The basic operations, also called *search steps*, are the following: 1) place a searcher on a node, 2) move a searcher along an edge, 3) remove a searcher from a node. Graph searching is the problem of developing a search strategy, i.e., a sequence of search steps resulting in all edges being simultaneously clean.

The main issues worth investigating include devising efficient search strategies and minimizing the number of searchers used by a strategy. In general, the smallest number of searchers for which a search strategy exists for a graph G is called the search number $s(G)$ of G. Determining the search number of a graph is NP-hard (see Meggido et al. [MHG$^+$88]). Two properties of search strategies are of particular interest: first, absence of recontamination, and second, connectivity of the cleared area. A search strategy is *monotone* if no recontamination ever occurs. Monotone searching becomes more important when the cost of clearing an edge far exceeds the cost of traversing an edge. Hence each edge should be cleared only once. Lapaugh [Lap93] has proved that for every G there is always a monotone search strategy that uses $s(G)$ searchers.

Another type of search is *connected* search whereby clean edges always remain connected. Connectivity is an important safety consideration and can be important when searcher communication can only occur within clean areas of the network. Such strategies can be defined by not allowing operation searcher removals and allowing searcher insertions either only in the beginning of the search or when applied to vertices incident to an already cleared edge. Optimal connected strategies are not necessary monotone. A family of graphs (depending on a parameter k) that can be contiguously searched by $280k + 1$ searchers but require at least $290k$ searchers for connected monotone search are constructed by Dyer [Dye05]. The problem of determining optimal search strategies under the connectivity constraint is NP-hard even on planar graphs but it has been shown that optimal connected strategies can be computed in linear time for trees.

Let $G = (V, E)$ be a graph (with possible multiple edges and loops), with n vertices and m edges. For subsets $A, B \subseteq E$ such that $A \cap B = \emptyset$ we define $\delta(A, B)$ to be the set of vertices adjacent to at least one edge in A and to at least one edge in B. We also use $\delta(A)$ to denote $\delta(A, E - A)$. A *k-expansion* in G is a sequence $X_0, X_1, \dots, X_r$, where $X_i \subseteq E$ for every $i = 0, \dots, r$, $X_0 = \emptyset, X_r = E$, and satisfying the following:

- $|X_{i+1} - X_i| \leq 1$, for every $i = 0, \dots, r - 1$,
- $|\delta(X_i)| \leq k$, for every $i = 0, \dots, r$.

The expansion number $x(G)$ of G is the minimum k for which there is a k-expansion in G. A graph with expansion number k can thus be obtained by adding one edge after the other, while at the same time preserving the property that no more than k nodes are on the boundary between the current and remaining set of edges. The following relation between expansion and searching holds: $x(G) \leq s(G) \leq x(G) + 1$, for any graph G.

Branch decomposition of a graph G is a tree T all of whose internal nodes have degree three, with a one-to-one correspondence between the leaves of the tree and the edges of G. Given an edge e of T, removing e from T results in two trees $T_1^{(e)}$ and $T_2^{(e)}$, and an e-cut is defined as the pair $\{E_1^{(e)}, E_2^{(e)}\}$, where $E_i^{(e)} \subset E$ is the set of leaves of $T_i^{(e)}$ for $i = 1, 2$. Note that $E_1^{(e)} \cap E_2^{(e)} = \emptyset$ and $E_1^{(e)} \cup E_2^{(e)} = E$. The width of T is defined as $\omega(T) = \max_e |\delta(E_1^{(e)})|$, where the maximum is taken over all e-cuts in T. The *branchwidth* $bw(G)$ of G is then $\min_T \omega(T)$, where the minimum is taken over all branch decompositions T of G. It follows easily that for any graph G, $bw(G) \leq \max\{x(G), 2\}$. Branchwidth is related to the ability of recursively splitting the graph into several components separated by few nodes. In particular, there is an edge of the optimal branch decomposition of a graph G whose removal corresponds to splitting G into components of size at most $m/2$ edges, and with at most $bw(G)$ nodes in common. For any graph G, we have that $bw(G) - 1 \leq s(G) = O(bw(G) \log n)$.

A k-expansion $X_0, X_1, \dots, X_r$ of a graph G is connected if, for any $i = 1, \dots, r$, the subgraph induced by X_i is connected. The *connected expansion number $cx(G)$* of G is the minimum k for which there is a connected k-expansion in G. Similarly, a search strategy is connected if the set of clear edges induces a connected subgraph at every step of the search. The *connected search number $cs(G)$* of a graph G is the minimum k for which there is a connected search strategy in G using at most k searchers. It can be proved that $cx(G) \leq cs(G) \leq cx(G) + 1$.

There is a polynomial-time algorithm that, given a branch decomposition T of a 2-edge-connected graph G of width k, returns a connected branch decomposition of G of width k (see Fomin et al. [FFT04]). Therefore, the connected branchwidth of any 2-edge-connected graph is equal to its branchwidth. In other words, there is no additional price for imposing the connectedness in branch decompositions. As a consequence, it is possible to partition the edges of any 2-edge-connected graph of branchwidth k into at most three connected subgraphs of size $m/2$ edges, sharing at most k nodes. The connected expansion cannot be too large in comparison with the expansion, and the same holds for graph searching. More specifically, it can be proved (see Fomin et al. [FFT04]) that for any connected graph G,

$$\begin{aligned} cx(G) &\leq bw(G) \cdot (1 + \log m), \\ cx(G) &\leq x(G) \cdot (1 + \log m), \\ cs(G) &\leq s(G) \cdot (2 + \log m), \end{aligned}$$

where m is the number of edges of the network.

1.7.2 Black holes

Mobile code is designed to run on arbitrary computers and as such it may be vulnerable to "hijacking" and "brainwashing". In particular, when an agent is loaded onto a host, this host can gain full control over the agent and either change the agent code or alter, delete or read/copy (possibly confidential) information that the agent has recorded from previous hosts (see Schelderup et al. [SØ99]). An important question is whether a mobile agent can shelter itself against an execution environment that tries to tamper and/or divert its intended execution (see Sander et al. [ST98]).

Blackoles were introduced by Dobrev et al. [DFPS01] in order to provide an abstract representation of a highly harmful object of unknown whereabouts but whose existence we are aware of. A black hole is a stationary process which destroys visiting mobile agents upon their arrival without leaving any observable trace of such a destruction. The problem posed by Dobrev et al. [DFPS01] "is to unambiguously determine and report the location of the black hole". The Black Hole Search (BHS) problem is to determine the location of the black hole using mobile agents. Thus, BHS is solved if at least one of the agent survives, while all surviving agents know the location of the black hole.

Consider a ring of n nodes and a set A of $k \geq 2$ mobile agents traversing the nodes of this ring. Assume that the size n of the ring is known to all the mobile agents but the number k of agents may not be a priori known to them. Suppose that the agents can move from node to (neighboring) node and have computing capabilities and bounded storage, follow the same protocol, and all their actions take a finite but otherwise unpredictable amount of time. Each node has a bounded amount of storage ($O(\log n)$ bits suffice), called a whiteboard, which can be used by the agents to communicate by reading from and writing on the whiteboards.

Let us consider the case of the ring network. At least two agents are needed to locate the black hole. Moreover, as a lower bound, it is shown in Dobrev et al. [DFPS01] that at least $(n-1)\log(n-1) + O(n)$ moves are needed in order to find a black hole, regardless of the number of agents being used. In addition, it is shown that two agents can find the black hole performing $2n \log n + O(n)$ moves (in time $2n \log n + O(n)$).

Questions of related interest include 1) how many agents are necessary and sufficient to locate a black hole, and 2) time required to do so and with what a priori knowledge. For additional results and details on the black hole search problem in arbitrary networks we refer the reader to Dobrev et al. [DFPS02]. For results on multi-agent black hole search see [DFPS04].

Acknowledgements

Research of the first author was supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants. Research of the third author was supported in part by PAPIIT-UNAM.

References

References

[ABRS95] B. Awerbuch, M. Betke, R. Rivest, and M. Singh. Piecemeal graph learning by a mobile robot. In *Proc. 8th Conf. on Comput. Learning Theory*, pages 321–328, 1995.

[AG03] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous.* Kluwer Academic Publishers, Norwell, Massachusetts, 2003.

[AH00] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[Alp95a] S. Alpern. The rendezvous search problem. *SIAM Journal of Control and Optimization*, 33:673–683, 1995.

[Alp95b] S. Alpern. The rendezvous search problem. *SIAM Journal of Control and Optimization*, 33:673–683, 1995. Earlier version: LSE CDAM Research Report, 53, 1993.

[Alp02] S. Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002.

[AR90] E.J. Anderson and R.R.Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability*, 28:839–851, 1990.

[BFFS03] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Election and rendezvous of anonymous mobile agents in anonymous networks with sense of direction. In *Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 17–32, 2003.

[BFR$^+$98] M.A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. 30th Ann. Symp. on Theory of Computing*, pages 269–278, 1998.

[BG01] V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 47(6):722–731, 2001.

[Bra97] J. Bradshaw. *Software Agents.* MIT Press, 1997.

[BRS95] M. Betke, R. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18:231–254, 1995.

[BS94] M.A. Bender and D. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proc. 35th Ann. Symp. on Foundations of Computer Science*, pages 75–85, 1994.

[DFKP04] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.

[DFP03] A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *11th Annual European Symposium on Algorithms (ESA)*, pages 184–195, 2003.

[DFPS01] Dobrev, Flocchini, Prencipe, and Santoro. Mobile search for a black hole in an anonymous ring. In *DISC: International Symposium on Distributed Computing.* Springer Verlag Lecture Notes in Computer Science, 2001.

[DFPS02] Dobrev, Flocchini, Prencipe, and Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. In *PODC: 21th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2002.

[DFPS04] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In *Symposium on Principles of Distributed Systems (OPODIS '03)*, pages 34–46, 2004. Springer Verlag Lecture Notes in Computer Science.

[DKK01] C.A. Duncan, S.G. Kobourov, and V.S.A. Kumar. Optimal constrained graph exploration. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 807–814, 2001.

[DP99] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32:265–297, 1999.

[Dye05] D. Dyer, 2005. PhD Thesis, Simon Fraser University, Burnaby B.C., Canada.

[FFT04] F. V. Fomin, P. Fraigniaud, and D. M. Thilikos. The price of connectedness in expansions. Technical Report LSI-04-28-R, Departament de Llenguatges i Sistemes Informaticas, Universitat Politecnica de Catalunya, Barcelona, Spain, 2004.

[FKK$^+$04a] P. Flocchini, E. Kranakis, D. Krizanc, F. Luccio amd N. Santoro, and C. Sawchuk. Mobile agent rendezvous when tokens fail. In *Proc. of 11th Sirocco*, 2004. Springer Verlag Lecture Notes in Computer Science.

[FKK$^+$04b] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in the ring. In *LATIN*, pages 599–608, 2004. Springer Verlag Lecture Notes in Computer Science.

[FMS98] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: definition, properties and classes. *Networks*, 32:29–53, 1998.

[GKKZ06] L. Gasieniec, E. Kranakis, D. Krizanc, and X. Zhang. Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring. In *Proceedings of SOFSEM 2006, 32nd International Conference on Current Trends in Theory and Practice of Computer Science*, 2006. January 21 - 27, 2006 Merin, Czech Republic, to appear.

[HKKK02] N. Hanusse, D. Kavvadias, E. Kranakis, and D. Krizanc. Memoryless search algorithms in a network with faulty advice. In *IFIP International Conference on Theoretical Computer Science*, pages 206–216, 2002.

[KK99] E. Kranakis and D. Krizanc. Searching with uncertainty. In *Proceedings of SIROCCO'99*, pages 194–203, 1999. C. Gavoille, J.-C. Bermond, and A. Raspaud, eds., Carleton Scientific.

[KKK$^+$01] A. Kaporis, L. M. Kirousis, E. Kranakis, D. Krizanc, Y. Stamatiou, and E. Stavropulos. Locating information with uncertainty in fully interconnected networks with applications to world wide web retrieval. *Computer Journal*, 44(4):221–229, 2001.

[KKKS03] L. M. Kirousis, E. Kranakis, D. Krizanc, and Y. Stamatiou. Locating information with uncertainty in fully interconnected networks: The case of non-distributed memory. *Networks*, 42(3):169–180, 2003.

[KKM06] E. Kranakis, D. Krizanc, and E. Markou. Mobile agent rendezvous in a synchronous torus. In *Proceedings of LATIN 2006, March 20-24, Valdivia, Chile*, 2006. Springer Verlag Lecture Notes in Computer Science.

[KKSS03] E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Mobile agent rendezvous search problem in the ring. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 592–599, 2003.

[KM06] D. Kowalski and A. Malinowski. How to meet in an anonymous network. In *Proc. 13th Sirocco*, 2006. Springer Verlag Lecture Notes in Computer Science.

[KP04] D. Kowalski and A. Pelc. Polynomial deterministic rendezvous in arbitrary graphs. In *Proc. 15th ISAAC*, 2004. Springer Verlag Lecture Notes in Computer Science.

[Lap93] A. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.

[MGK$^+$05] G. De Marco, L Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vacaro. Asynchronous deterministic rendezvous in graphs. In *Proc. 30th MFCS*, pages 271–282, 2005. Springer Verlag Lecture Notes in Computer Science.

[MHG$^+$88] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, 1988.

[Mor] P. Morin. Personal Communication.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.

[NN97] H. S. Nwana and D. T. Ndumu. An introduction to agent technology. In *Software Agents and Soft Computing*, pages 3–26, 1997.

[OR94] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[Pel00] D. Peleg. *Distributed Computing: A Locality Sensitive Approach*. SIAM Monographs in Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, 2000.

[PP99] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33:281–295, 1999.

[RD01] N. Roy and G. Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11:117–136, 2001.

[RHSI93] N. S. V. Rao, S. Hareti, W. Shi, and S.S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical report, Oak Ridge National Laboratory, 1993. Tech. Report ORNL/TM-12410.

[San99] T. W. Sandholm. Distributed rational decision making. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. MIT Press, 1999.

[Saw04] C. Sawchuk. *Mobile Agent Rendezvous in the Ring*. PhD thesis, Carleton University, School of Computer Science, Ottawa, Canada, 2004.

[Sho97] Yoav Shoham. An overview of agent-oriented programming. In J. M. Bradshaw, editor, *Software Agents*, chapter 13, pages 271–290. AAAI Press / The MIT Press, 1997.

[SØ99] K. Schelderup and J. Ølnes. Mobile agent security — issues and directions. In *6th International Conference on Intelligence and Services in Networks*, volume 1597, pages 155–167, 1999. Springer Verlag Lecture Notes in Computer Science.

[ST98] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, pages 44–60, 1998.

[Tel99] G. Tel. Distributed control algorithms for ai. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 539–580. MIT Press, 1999.

[vNM53] J. von Neumann and G. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

[Wat02] J. Watson. *Strategy: An Introduction to Game Theory*. W. W. Norton and Company, 2002.

[Woo99] M. Wooldridge. Intelligent agents. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 27–77. MIT Press, 1999.

[Woo02] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2002.

[ws] JXTA web site. http://www.jxta.org/. Accessed Jan 22, 2006.

[YY96] X. Yu and M. Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *Proceedings of ICALP '96*, pages 610–621, 1996. Springer Verlag Lecture Notes in Computer Science.