The Distributed Computing Column

by

Panagiota Fatourou

Department of Computer Science, University of Crete
P.O. Box 2208 GR-714 09 Heraklion, Crete, Greece
and
Institute of Computer Science (ICS)
Foundation for Research and Technology (FORTH)
N. Plastira 100. Vassilika Vouton
GR-700 13 Heraklion, Crete, Greece
faturu@csd.uoc.gr

Mobile Agents in Distributed Computing: Network Exploration

Shantanu Das
LIF, Aix-Marseille University & CNRS, France
sdas@bitvalve.org

Abstract

One of the recent paradigms in networked distributed computing is the use of mobile agents. Mobile agents are software robots that can autonomously migrate from node to node within a network. Although mobile agents can be easily implemented over a message passing network, they provide an abstraction for designing algorithms in a non-traditional way which can be

Bulletin of the EATCS no 109, pp. 54–69, February 2013

quite natural for certain problems, such as searching, monitoring or intruder detection. A principle sub-task in most algorithms for mobile agents is the traversal of the network. We focus on this problem of exploring an initially unknown network with one or more mobile agents. We also consider the related problem of constructing a map of the environment being explored by the mobile agents.

1 Introduction

Mobile agents are software robots (mobile code) that can move autonomously from node to node within a network, executing its operations and interacting with the host environment at each node that it visits. Such software robots (sometimes called bots, or agents) are already prevalent in the Internet, and are used for performing a variety of tasks such as collecting information, negotiating a business deal, or for online shopping. When the data needed for a computation is physically dispersed, it can be sometimes beneficial to move the computation to the data, instead of moving all the data to the node performing the computation. The paradigm of mobile agent computing is based on this idea.

The use of mobile agents has been advocated for various reasons such robustness against disruptions in network connectivity, improving the network latency and reducing network load, providing more autonomy, and so on (see e.g. [26]). In the context of distributed computing, the use of mobile agents has been suggested by Fukuda et al. [22] as early as the 90's and some algorithms for message-passing networks that use token circulation (notably [25]) can be regarded as early examples of mobile agent algorithms in this area.

On the practical side, one of the major concerns with mobile agents has been ensuring the agents are safe from tampering by potentially malicious host nodes and on the other hand the host computer are not harmed by malicious agents[33, 28]. One typical example of a malicious agent is a computer virus that propagates itself through the network. As there could be malicious agents, we could use good agents to track down and eliminate the harmful agents. This gave rise to several *cops-and-robber* games on networks, where a team of "good" agents move through the network searching for the malicious agent (the robber) and the latter tries to evade the good agents (the cops). This is only one of several problems involving mobile agents that has recently captured the interest of the distributed computing community. Other problems that have studied in the context of distributed computing are:

- The *Rendezvous* problem : Gathering multiple mobile agents at a single location.

- *Black hole search*: Locating harmful nodes in the network.
- *Distributed treasure-hunt*: Locating a resource available at an unknown node of the network.
- *Map construction*: Building a map of an unknown network.
- *Distributed verification*: Verifying some global property of the network.

Some of these areas are already too vast to cover in one combined survey. The rendezvous problem has been discussed in the recent surveys [30] and [9] as well as the book [2], while the black hole search problem has been studied by Markou [27] in the previous edition of this column.

In this article we would focus on the exploration problem which is usually a basic subtask useful for solving the other problems. The algorithm used for traversal of a network usually has a significant impact on both the solvability and the cost of solving these other problems. We would discuss efficient techniques for traversing an unknown network and for building a map of the network. The article is structured as follows. Section 2 explains the mobile agent model in detail. The following section presents exploration techniques for a single mobile agent. We consider both exploration of labelled as well as unlabelled networks. Section 4 discusses the special case of an agent having limited memory. Finally, Section 5 considers the collaborative exploration of networks by multiple mobile agents.

2 The Mobile Agent Model

In the mobile agent model, the network or the environment is modelled as an undirected[1] connected graph $G = (V(G), E(G))$. The computation is performed by a set Q of mobile entities called agents. An agent is an automaton that starts at some vertex of G, in some given state s_0. Each agent has a finite number of states and the size of the state-space S depends on the amount of memory available to an agent (which may or may not depend on the size of the network). An agent with b bits of memory is assumed to have 2^b states. The initial placement of the agents is denoted by the function $p : Q \to V(G)$. For any $a \in Q$, $p(a)$ is called the *homebase* of agent a.

At any step of the algorithm, a mobile agent located at a node v of the network, may (1) detect the presence of other agents at the node v and communicate with them, and (2) perform any computation at node v, using the information available at node v and its own state information, and (3) change its state and either decide

[1]This article considers only the undirected environment although there exists results for directed graphs too.

to stay at node v or move to an adjacent node. When performing computations at a node v, the agent has access to the (possibly unbounded) memory and computational power of the node v. When moving from a node v to an adjacent node u, the agent is allowed to carry only its state information.

In order to enable navigation of the agents in the graph, at each node $v \in V(G)$, the edges incident to v are distinguishable to any agent a at node v. In other words, there is a bijective function

$$\delta_v : \{(v,u) \in E(G) : u \in V(G)\} \rightarrow \{1, 2, \ldots d(v)\}$$

which assigns unique labels to the edges incident at node v (where $d(v)$ is the degree of v). The function $\delta = \{\delta_v : v \in V(G)\}$ is called the local orientation or port-numbering[2].

We consider three different models for communication between mobile agents: (1) Face-to-Face: In this model, two agents can exchange information only when they are both located in the same node. (2) Pebble Model: In this case, the agent is allowed to leave a pebble at its current location v. This pebble is visible to (and can be picked up by) any agent that subsequently visits node v. (3) Whiteboard Model: In this case, an agent may write any information at public whiteboards at its current node v. This information is visible to (and can be modified by) any agent that subsequently visits node v.

In some cases, the vertices of G may be initially labelled over the set of symbols L by $\lambda\colon V(G) \rightarrow L$ which is the labelling function. When this labelling is injective, we say that the nodes have distinct identifiers. On the other hand, when all nodes have the same label $c \in L$ we say that the network is *anonymous*.

The environment is thus, represented by the tuple $(G, \lambda, \mathcal{Q}, p, \delta)$. In case the nodes of the graph are anonymous, we shall omit λ. For the rest of this paper, $n = |V(G)|$ and $m = |E(G)|$ respectively denotes the numbers of vertices and of edges of G, while $k = |\mathcal{Q}|$ denotes the number of agents and Δ denotes the maximum degree of a vertex in G. We shall use the words vertex and node as well as the words graph and network interchangeably.

Since we consider asynchronous networks, the cost of an algorithm is measured in terms of the number of *moves* performed by the agents where each move corresponds to the traversal of a single edge by a single agent. In case of single agent exploration, the *moves cost* is same as the time complexity, if we assume each move to take the same time.

Finally we remark here that the mobile agent model as explained above can be easily implemented on message-passing networks. In fact, there exist simulation techniques for importing algorithms from mobile agent systems to message-passing systems and vice versa [10, 15].

[2]The labels on the edges may correspond to port numbers on a network

3 Network Exploration by a Mobile Agent

The problem of exploring an unknown environment has been studied extensively starting from the work of Shannon [34] who first considered the problem for a finite state automaton moving in a maze. One of the questions that has been studied is whether finite state automata can explore graphs of arbitrary size. This question was answered by Rollik [32] who showed that any finite team of finite state automata cannot explore all graphs. Later Fraigniaud et al.[21] showed that a single agent needs $\Omega(logn)$ bits for traversing arbitrary graphs of size n and maximum degree at least three. We consider the problem of exploration with memory restrictions in Section 4.

Depending on the objective of exploration, an algorithm may terminate before visiting all nodes, or only after visiting each node at least once, or it may not stop and continue indefinitely. The latter scenario is called *perpetual exploration* and is sometimes useful e.g. for network monitoring. In the perpetual exploration problem [21], the agent is required to periodically visit each node and optimizing the period of the traversal may be one of the objectives. On the other hand, for the *treasure-hunt* problem [35], the agent can stop as soon as it reaches a node containing the “treasure" (i.e. the resource being searched).

In some cases visiting all edges (not just the nodes) may be necessary (e.g. for the *edge-search* problem). In some other cases, the agent may be required to build a map of the environment (such a map enables the agent to find a shorter traversal path for subsequent traversals). For any agent traversing a graph, a *map* of the network is a copy of the graph G where the edges are labelled with port-numbers and the node where the agent is located is specifically marked. Access to a map allows the agent to traverse the network faster and also to perform certain computations without the need to traverse the network. Thus, a map construction algorithm is a useful primitive in mobile agent computing.

3.1 Exploration of Labelled Environments

When the nodes of the graph are labelled with distinct identifiers, exploration can be achieved by the conventional depth-first search (DFS) or breadth-first search (BFS) algorithms. In fact, the depth-first traversal is asymptotically optimal in terms of number of moves because it requires $\Theta(m)$ edge traversals. A modified version of the algorithm has been proposed by Panaite and Pelc [29] which makes $m + O(n)$ moves. In this case, the penalty (i.e. the number of additional edges traversed) is linear in the numbers of nodes (and not the number of edges). The algorithm achieves this by using some clever rules for backtracking that avoids traversing too many of the already explored edges.

Another version of the problem that has been studied by Awerbuch *et al.* [3]

is called *piecemeal* exploration where the agent has to periodically return to its homebase (e.g. for refuelling), during the exploration. For this version of the problem, the standard DFS algorithm is not a feasible algorithm; During DFS, an agent may make $\Omega(n)$ moves between two subsequent visits to the homebase, even if the diameter of the graph is very small. The BFS algorithm can be used to solve the problem of piecemeal exploration, but it could require $O(m^2)$ moves in the worst case. Instead, the authors use a combination of DFS and BFS, where BFS is performed locally within small regions, called *strips*, while DFS is performed on a higher level spanning tree that connects these strips together. This algorithm requires $O(m + n^{1+\epsilon})$ moves. This was later improved upon by Awerbuch and Kobourov [4] who gave a recursive piecemeal exploration algorithm that requires $O(m + n \log^2 n)$ moves. Eventually Duncan et al. [19] gave an asymptotically optimal algorithm for performing piecemeal exploration in $O(m)$ moves.

Note that all the above traversal methods require the agent to have sufficient memory to remember the identifiers of the visited vertices. Thus the agent must have at least $\Omega(n \log n)$ bits of memory. For agents with smaller memory, more involved techniques may be required (c.f. Section 4), thus increasing the time complexity of exploration. On the other hand, an agent having sufficient memory can also construct the map of the graph using the information obtained during a traversal of the graph. Thus, in this case, the problem of map construction is not different from the task of simply traversing all edges of the graph in an organized manner. When the nodes of the network are anonymous (not labelled with unique identifiers), then the task of map construction could be more difficult than simply performing a traversal of the network. This scenario is considered in the next two sections.

3.2 Exploration of Anonymous Networks

In an anonymous network, the nodes are not labelled with identifiers and thus the agent may not be able to distinguish between any two nodes which have the same degree. Note the edges of the graph are still locally oriented with port-numbers since otherwise exploration is not possible (as explained before).

Traversal of an anonymous graph can always be achieved by a breadth-first traversal of all paths up to a depth of D, the diameter of the graph (if the value of D is known), or up to depth n or, any other upper bound on the diameter. Note that the knowledge of the size n or the diameter D of the graph is necessary only for terminating the traversal (one could in principle continue infinitely and thus ensure that all nodes and edges of the graph will be visited within a finite time). If the agent knows the value of n, it could perform a depth-first traversal of the tree containing all paths of length n from the starting vertex. This incurs a cost of $\Omega(\Delta^n)$ moves in graphs of maximum degree Δ. It is possible to have

a more efficient traversal of the graph using the concept of universal exploration sequences.

A universal traversal sequence for a graph G is a list of port numbers, such that, if an agent starting in any node of G chooses to move according to the specified port numbers, it will eventually visit all nodes of G. For our purpose, a more useful notion is that of a *Universal Exploration Sequence* (UXS) defined by Koucký [24] as follows. For any node $u \in G$, the ith successor of u, denoted by $succ(u, i)$ is the node v reached by taking port number i from node u (where $0 \leq i < d(u)$). Let $(a_1, a_2, \ldots, a_t)$ be a sequence of integers. An *application* of this sequence to a graph G at node u is the sequence of nodes $(u_0, \ldots, u_{t+1})$ obtained as follows: $u_0 = u$, $u_1 = succ(u_0, 0)$; for any $1 \leq i \leq t$, $u_{i+1} = succ(u_i, (p + a_i) \mod d(u_i))$, where p is the port-number at u_i corresponding to the edge $\{u_{i-1}, u_i\}$. A sequence $(a_1, a_2, \ldots, a_t)$ whose application to a graph G at any node u contains all nodes of this graph is called a UXS for this graph. A UXS for a class of graphs is a UXS for all graphs in this class. An important result shown in [1] is the following.

Property 3.1. *For any positive integers n, Δ, $\Delta < n$, there exists a UXS of length $O(n^3\Delta^2 \log n)$ for the family of all graphs with at most n nodes and maximum degree at most Δ.*

It is also known that such universal sequences are easy to construct and thus, using the above property a mobile agent can traverse any graph G of size n, incurring a cost polynomial in n, which compares favorably with the exponential cost of the brute-force algorithm described earlier. However, it is not known whether there is a more efficient way of traversing arbitrary unknown graphs other than using universal exploration sequences. Thus there is a big gap in the cost of exploring anonymous networks compared to networks with uniquely labelled nodes.

If we consider the pebble model of communication where an agent is allowed to place a pebble on a node to mark it, then it is possible to efficiently traverse an anonymous graph G (as well as build a map of G). The agent can start with an initial map containing only its current node and start exploring new edges and adding them to the map. Whenever the agent traverses an edge $e = (v, u)$ and arrives at an unknown vertex u, it can place the pebble on u, return back to the previous vertex v and perform a traversal of the known part of G (that is included in the current map). If the agent encounters the pebble during this traversal, it can identify the vertex u in its map. Otherwise, the node u is an unexplored node and the agent can add this node and the edge e to its map. In both cases, the agent has extended its map by a single edge and it can now pick up the pebble from node u and continue the exploration. Using this process a mobile agent with a single pebble can explore and map an anonymous graph in $O(nm)$ (i.e. $O(n^2\Delta)$) moves. Moreover the agent does not need any prior knowledge of the size n of the network or even an upper bound on n.

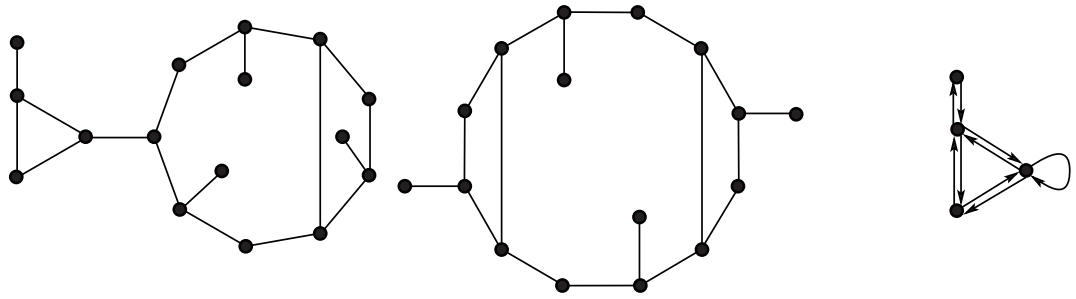

Figure 1: Two non-isomorphic graphs that are indistinguishable to a mobile agent, and the common base graph of the first two graphs.

As an aside, we remark here that exploration of (strongly connected) directed graphs using a pebble has also studied [6] and the best known algorithm for mapping a digraph using a single pebble has a cost of $O(n^8\Delta^2)$. Thus it seems that exploring digraphs is much more difficult than exploring undirected graphs and there exists a large gap between the costs of exploration in these two cases. Moreover, when the agent does not know any upper bound on the size of the network, a single pebble is not sufficient for exploring digraphs and at least $\Omega(\log\log n)$ pebbles are necessary, as shown in the above paper.

3.3 Map Construction in Anonymous Networks

Although any arbitrary graph can be traversed by an agent having sufficient memory, the problem of map construction cannot be solved in all graphs. There exists graphs which are not recognizable, i.e. an agent traversing such a graph can not build a map of the graph, even if it is traverses every edge of the graph and even if it has an unbounded amount of memory allowing it to remember everything that it sees. This impossibility comes from the existence of symmetries in certain graphs. Consider for example the two graphs shown in Figure 1. The two graphs are distinct (non-isomorphic) but an agent traversing the first graph sees exactly the same as an agent traversing the second graph. Thus, both these graphs are non-recognizable. There exists a characterization of the family of graphs that are not recognizable, given by Yamashita and Kameda [36]. This can be explained in terms of the concept of graph coverings [7]. For any two graphs G and H that are indistinguishable by an exploring agent, there exists a common base graph B which is the smallest multi-graph that cannot be distinguished from either G or H. We say that graph G covers graph B. Map construction can be solved in a graph G only if it is covering-minimal, i.e. there exists no smaller graph B that is covered by G.

Algorithm 1: Map Construction from a uniquely marked node r

```
Map := T := {r} ;
Add r to Queue;
ROOT_PATHS := ∅;
while Queue is not empty do
    Get next node v from Queue and go to v using Map;
    while node v has unexplored edges do
        Traverse the next unexplored edge e = (v, u);
        for each path P ∈ ROOT_PATHS do
            Apply label sequence P at node u ;
            if successfully reached a marked node then
                Add to Map a cross-edge from v to Start(P);
                Update the number of explored edges at the node Start(P);
                Return to node v using T and exit Loop;
            else
                Backtrack to node u ;
        if All path sequences failed to reach a marked node then
            Add a new node u to T and Map ;
            Add edge (v, u) to T and Map ;
            Insert u to Queue ;
            ROOT_PATHS := ROOT_PATHS ∪ Path_T(u, r) ;
            Backtrack to node v ;
```

The impossibility of mapping arbitrary anonymous graphs can be overcome when the mobile agent has some means of marking the nodes of the graph. As mentioned in the previous section, when the mobile agent is provided with a pebble, map construction can be solved in all connected graphs. A much weaker assumption, that the starting node of the agent is distinctly marked, is also a sufficient condition for mapping an arbitrary graph. An algorithm that achieves this was presented in [8] (see Algorithm 1 below). The idea of the algorithm is the following. Let the starting node (which is distinctly marked) be r. The agent can perform a breadth-first traversal building a BFS-tree T rooted at r. During the traversal, whenever the agent explores a new edge and reaches a node v, the agent checks whether v has been visited before (i.e. whether node v is the same as some node u in its tree). This can be done by successively applying the label-sequences for the reverse paths from each node $u \in T$ to the root r, and checking if one of these paths hits the marked node. To this end, the algorithm maintains a data structure, called ROOT_PATHS, that stores the label-sequence for each path P in

T going from any node $u \in T$ to the root r. For such a stored path P, Start(P) refers to the node u. During the process of building the BFS-tree T, the algorithm also constructs a map that contains all the cross-edges discovered by the agent, in addition to the tree edges belonging to T. The algorithm completes the map construction in $O(n^3\Delta)$ moves.

4 Exploration with Small Memory

As mentioned before, a mobile agent requires $\Omega(\log n)$ bits of memory in order to explore all graphs of n nodes. It was an open question for a long time whether exploration of arbitrary graphs can be performed by an agent having logarithmic memory, until Reingold [31] gave a positive answer. Note that an agent having $\Theta(\log n)$ can remember only a few node identifiers in a labelled network. Thus, the conventional DFS or BFS algorithms cannot be used for exploration. It is possible to use a UXS to perform the exploration, due to the results of Reingold [31] who gave a log-space constructible UXS. This however, requires the prior knowledge of an upper bound on n. When such knowledge is not available, we can use the following strategy for exploration. The agent has enough memory to remember the identifier of the starting location (which we call the source). The idea is that the agent guesses a value N as an upper bound for n and performs a traversal using UXS for graphs of size N. During this traversal whenever it reaches a vertex v, the agent performs a check operation and if this operation returns false, then the agent aborts the current traversal, doubles the value of N, and starts the whole procedure again using the new value of N and the current vertex as source. The checking operation at a vertex v, simply visits each neighbor w of v, remembers the identifier of w, returns to the source and performs another traversal to check if the traversal visits w. If not, then the checking operation returns false. Otherwise it continues to check the other neighbors of v and finally return true. If the agent completes the traversal without having to increase N anymore then this traversal has visited all nodes of G. So the agent can stop. Thus the algorithm performs exploration with stop, without the need for any prior knowledge of the network size.

In case of anonymous networks, some upper bound on the value of n is always necessary for exploration, unless the network is a tree. For anonymous tree networks, Gasieniec et al.[23] gave an algorithm for an agent with $O(\log n)$ memory to explore the network and return to its starting location, without any prior knowledge of the network size. Earlier Diks et al. [17] showed that $\Omega(\log\log\log n)$ memory is necessary for tree exploration with stop and $\Omega(\log n)$ memory is required if the agent needs to return to the starting location. Note that in a tree network, exploration without stop can be performed even by an agent having no

memory, using the simple strategy of always leaving a node by the next port (in cyclic order) from the one through which it arrived. This is often called the *right-hand-on-the-wall* strategy.

The exploration of arbitrary graphs with agents having O(1) memory (i.e. finite state automata), has also been well investigated. In this case there must be some additional mechanism to help the agent perform its task of exploration. For example the nodes or edges of the network may be labelled in such a way as to allow the finite state agent to complete the exploration. Dobrev et al. [18] considered the problem of assigning port numbers to the edge of the graph in such a way that an agent with no memory performing a right-hand-on-the-wall walk can periodically visit all nodes of the graph. The objective is to obtain a small period of traversal and the above paper achieved a period of $10n$ for graphs of size n. Eventually this was improved to a period of $(4 + 1/3)n - 4$ in a more recent paper by Czyzowicz et al. [12]. For an agent with $O(1)$ bits of memory, the authors presented an algorithm and a corresponding port assignment that allows periodic traversal with a period of at most $3.5n$. In contrast, there exists a trivial lower bound of $2n - 2$ for the period of traversal by any agent (irrespective of memory size). For the case of the agent with no memory, a stronger lower bound of $2.8n$ was shown in the same paper. Thus there still exists a small gap between the best known lower and upper bounds for the problem.

5 Network Exploration by Multiple Agents

We have so far considered the exploration problem for a single agent. When there are multiple agents starting from the same node of the network, they may collaborate with each-other to explore the network collectively. Note that if the agents are all identical and follow the same deterministic algorithm, then all the agents would move together and we would not gain anything from having multiple agents instead of one. Thus, there must be some means of breaking symmetry between the agents. For example, if the agents have distinct names then each agent can follow a different path and thus explore different parts of the network concurrently.

Another possible scenario is when multiple agents start from distinct nodes of the network. In this case each agent may independently explore the network using the techniques discussed in the previous sections. However if the agents mark the nodes during the exploration then there needs to be some agreement between the agents so that an agent does not get confused when it encounters a node marked by some other agent. For example, Algorithm 1 presented in the previous section, assumes a uniquely marked homebase; this algorithm would fail when there are multiple agents starting from distinct nodes. The problem

of distributed exploration with multiple agents initially dispersed in the graph, is considered in Section 5.2.

5.1 Collaborative Exploration

The collaborative exploration of a network by a team of collocated mobile agents was studied by Fraigniaud et al. [20]. They considered tree networks where a team of k synchronous mobile agents labelled as $1, 2, \ldots k$ located at the root of the tree, need to explore all nodes and return to the starting node. The paper provided an algorithm that takes $O(D + n/\log k)$ time for the exploration by k agents, thus providing an improvement by a factor of $(\log k)$ over single agent exploration. The agents do not know the exact topology or, even the size of the network. The main idea of the algorithm is the following. At any stage of the algorithm, the agents available at node v are distributed among its subtrees in such a way that the number of agents in any two (unexplored) subtrees does not differ by more than one. Whenever a subtree has been explored completely all agents in that subtree move to its parent. The authors show how to implement this strategy in a distributed manner, where the communication between agents is achieved by reading and writing information on whiteboards available at each node.

Note that any algorithm for collaborative exploration by k agents must require $\Omega(D + n/k)$ time. Thus there is an overhead of $k/\log k$ in the above algorithm and it is not known whether this overhead can be reduced. A lower bound of $2 - 1/k$ was shown for the overhead of any collaborative exploration algorithm using k agents.

Collaborative exploration of arbitrary graphs has been considered in the context of the black hole search problem [11]. The objective, in this case, is to locate harmful nodes (called *black holes*) in the network and any agent arriving at such a node dies and cannot continue with the exploration. The above paper provided an exploration algorithm for k agents that takes $O(n/k)$ time when the number of black holes is at most $k/2$ and $k = O(\sqrt{n})$. However this algorithm assumes that the network topology is known in advance by the mobile agents.

5.2 Distributed Exploration

When there are multiple mobile agents initially dispersed among the nodes of an anonymous network it is possible to collectively explore the network if the agents can communicate by leaving marks. In particular we will assume the whiteboard model of communication where agents can read and write information on public whiteboards at each node that the agent visits. If the agents have distinct identities, each agent can individually explore the complete network by performing a DFS marking each visited node with its identifier. On the other hand if the agents are

identical (i.e. no distinct names) then the marks made by an agent would not be distinguishable from those of another agent. In this case some cooperation between the agents seems necessary. A distributed exploration of the network can be performed by the following algorithm [13], which we call a distributed depth-first search (DDFS).

Procedure DDFS: An agent A starts from its homebase a depth-first search traversal marking the nodes that it visits (unless they are already marked) and labelling them with numbers $1, 2, 3$, and so on. Each node marked by the agent and the edge used to reach it are added to a tree data structure stored in the memory of the agent. If the agent reaches an already marked node, it backtracks to the previous node and tries the other edges incident to the node. The agent stops when there are no unexplored edges incident to the nodes of its tree. The tree obtained at the end of the traversal is called the territory T_A of the agent.

It can be easily shown that the territories obtained by the agents in the above process, forms a spanning forest of the graph G. Thus, each node of the agent is visited by some agent and the agents together complete the exploration of the graph. The exploration requires $O(m)$ moves in total. If the agents are required to perform periodic traversals of the graph, then any subsequent traversal of G can be performed using only $(2n - 2k)$ moves in total (by restricting each agent to traverse its own territory).

The above technique was extended in [14] to perform map construction in those cases when G is covering-minimal (i.e. when it is possible to construct a map of G). The map construction algorithm proceeds in two phases. In the first phase, the agents simply perform procedure DDFS. At the end of this procedure there is exactly one agent in each tree in the forest and each agent a has a map of the tree T_a that it belongs to. The second phase of the algorithm is a competition between neighboring agents, during which each losing agent merges its territory with the corresponding winning agent. This process is repeated with the objective of eventually forming a single tree spanning the graph G. As a final step, nodes of the tree are assigned unique labels and then all non-tree edges are added to obtain a complete map of G. The main complication in the algorithm in the competition phase which proceeds in several rounds (at most k rounds when there are k agents). Overall, the algorithm has a cost of $O(mk)$ moves in total.

6 Conclusions

We considered the exploration of unknown networks (graphs) by mobile agents that can autonomously move along the edges of the graph. Graph exploration is a basic subtask in most mobile agent algorithms and efficient techniques for

exploration can help to speed up the computation by mobile agents. We also studied techniques for constructing a map of an initially unknown network. Map construction algorithms help us to execute algorithms for known topologies in unknown environments (or, in dynamic environments where it is not possible to have a-priori knowledge of the network topology).

The scope of this article is restricted in many ways. We consider mainly the exploration of undirected graphs instead of the more difficult task of exploring strongly connected directed graphs. In fact, it is not always possible to explore arbitrary digraphs if the nodes are not uniquely labelled and there is no marking device. Secondly, the algorithms presented in this survey are for the fault-free scenario and it is possible to consider the exploration problem in the presence of various failures, such as node failures, agent crashes, and the failure of marking devices. Finally, this article excludes many interesting results on randomized algorithms for exploration e.g. based on random walks by mobile agents.

References

[1] R. Aleliunas, R. M. Karp, R.J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In 20th Annual Symposium on Foundations of Computer Science (FOCS), pp. 218–223, 1979.

[2] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*. Kluwer, 2003.

[3] B. Awerbuch, M. Betke, and M. Singh. Piecemeal graph learning by a mobile robot. *Information and Computation*, 152:155–172, 1999.

[4] B. Awerbuch and S.G. Kobourov. Polylogarithmic-Overhead Piecemeal Graph Exploration. In Proc. llth Annual Conference on Computational Learning Theory (COLT), 280-286, 1998.

[5] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40(2):143–162, 2007.

[6] M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In Proc. 30th ACM Symp. on Theory of Computing (STOC), pages 269–287, 1998.

[7] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, 2002.

[8] J. Chalopin, S. Das, and A. Kosowski. Constructing a Map of an Anonymous Graph: Applications of Universal Sequences. In Proc. 14th International Conference on Principles of Distributed Systems(OPODIS), pages 119-134, 2010.

[9] J. Chalopin, S. Das, and P. Widmayer. Deterministic Rendezvous in Arbitrary Graphs: Overcoming Anonymity, Failures and Uncertainty, *Search Theory: A Game Theoretic Perspective*, Springer, 2013.

[10] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agents algorithms versus message passing algorithms. In Proc. of the 10th Int. Conference on Principles of Distributed Systems (OPODIS'06), LNCS 4305, pages 187–201, 2006.

[11] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. In Proc. 10th Int. Conf. On Principles of Distributed Systems (OPODIS), pages 320-332, 2006.

[12] J. Czyzowicz, S. Dobrev, L. Gasieniec, D. Ilcinkas, J. Jansson, R. Klasing, I. Lignos, R. Martin, K. Sadakane, W.K. Sung. More efficient periodic traversal in anonymous undirected graphs. *Theoretical Computer Science* 444: 60-76, 2012.

[13] S. Das, P. Flocchini, A. Nayak, S. Kutten, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1-3):34–48, 2007.

[14] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Effective elections for anonymous mobile agents. In Proc. 17th Symp. on Algorithms and Computation (ISAAC), pages 732-743, 2006.

[15] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In Proc. 14th Coll. on Structural Information and Communication Complexity (SIROCCO), LNCS 4474, pages 289–303, 2007.

[16] S. Das, S. Kutten, Z. Lotker. Distributed Verification using Mobile Agents. In Proc. 14th International Conference on Distributed Computing and Networking (ICDCN), pages 330-347, 2013.

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

[18] S. Dobrev, J. Jansson, K. Sadakane, and W.-K. Sung. Finding short right-hand-on-the-wall walks in graphs. In Proc. 12th Colloquium on Structural Information and Communication Complexity (SIROCCO), LNCS 3499, 127–139, 2005.

[19] C.A. Duncan, S. G. Kobourov, and V.S. Anil Kumar. Optimal constrained graph exploration. In Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 807-814, 2001.

[20] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3): 166-177, 2006.

[21] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2Ű3):331Ű344, 2005.

[22] M. Fukuda, L.F. Bic, M.B. Dillencourt, and J.M. Cahill. Messages versus messengers in distributed programming. *Journal of Parallel and Distributed Computing*, 57(2):188–211, 1999.

[23] L. Gasieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. In Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 585-594, 2007.

[24] M. Koucký, Universal traversal sequences with backtracking, *J. Comput. Syst. Sci.* 65:717-726, 2002.

[25] E. Korach, S. Kutten, S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems*, 12(1):84–101, 1990.

[26] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communcations of the ACM*, 42(3):88–89, 1999.

[27] E. Markou. Identifying Hostile Nodes in Networks Using Mobile Agents. Bulletin of the *European Association for Theoretical Computer Science*, 108:93-129, 2012.

[28] R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12):1165 – 1170, 1999.

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

[30] A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, *Networks*, 59(3): 331-347, 2012.

[31] O. Reingold. Undirected ST-connectivity in log-space, In Proc. 37th Annual ACM Symposium on Theory of Computing (STOC), 376-385, 1998.

[32] H.A. Rollik. Automaten in planaren Graphen. *Acta Informatica* 13:287-298, 1980.

[33] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In Proc. of Conf on Mobile Agent Security, LNCS 1419, pages 44–60, 1998.

[34] CL. E. Shannon. Presentation of a maze-solving machine. In 8th Conf. of the Josiah Macy Jr. Foundation (Cybernetics), pages 173–180, 1951.

[35] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 599–608, 2007.

[36] M. Yamashita and T. Kameda. Computing on anonymous networks: Part I–Characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.