CHAPTER 16

# Graph Explorations with Mobile Agents

Shantanu Das

Aix-Marseille Univ., LIS, CNRS, France. <shantanu.das@lis-lab.fr>

**Abstract.** The basic primitive for a mobile agent is the ability to visit all the nodes of the graph in a systematic manner. This chapter considers the exploration of unknown graphs in full detail, for the specific mobile agent model considered in this book. The graph is considered to be finite, undirected and connected. Other than this fact, no prior knowledge of the graph is assumed. Several exploration techniques are introduced and explained for either a single agent, or multiple agents, exploring either labelled or unlabelled graphs. We focus on the efficiency of exploration and consider three different complexity measures, the time taken, the amount of memory used by the agents and the storage needed at each node of the graph. For exploration by multiple agents, we consider collaborative exploration by a team of colocated agents as well as distributed exploration by agents scattered in a graph. The concluding section presents some brief ideas and references on more advanced topics on graph exploration that are not covered in this chapter.

**Keywords:** mobile agents, graph exploration, undirected graph, deterministic, anonymous

## 1 Introduction

Most tasks for mobile agents require them to navigate in a graph, visiting all the nodes in a systematic manner. We call the task of visiting all the nodes of an unknown graph as the *Exploration* problem. We will only consider finite and connected graphs in this chapter. Consider a single mobile agent located in one of the nodes of such a graph. The agent can move only along the edges of the graph. The task of Exploration requires the agent to visit each node of the graph at least once. Depending on the objective, further requirements for the agent may be to terminate after visiting all nodes, or to return to the starting node. The former is called "Exploration with stop" while the latter is referred to as "Exploration with return". As an example, if the agent is required to search for some resource (or information) among the nodes of a graph, it must be able to determine whether the target resource is actually present in the graph and if not return back to the starting node and report failure. In such case, termination of the exploration is important. On the hand, if the agent is required to monitor the nodes of the graph to prevent intruders (e.g. guards patrolling an art gallery),

in such cases, we do not require the agent to terminate the exploration but rather continue visiting all nodes repeatedly. Such an exploration where each node needs to be visited infinitely often is called "Perpetual Exploration" of the graph.

Throughout this chapter we assume that the graph to be explored is initially unknown to the mobile agent. Further an agent has only local visibility restricted to the current node where it is located. So, the agent starting at a node $v$ sees only the node $v$ and the incident edges to it; the agent can not even see the neighboring nodes at the other end of these edges. As the agent visits more nodes, it may store in its memory the history of what it has explored so far. The problem of reconstructing the graph from this information obtained in course of the exploration, is called the *Map Construction* problem.

We distinguish the mobile agent exploration problem from the graph traversal problem in the context of the graph data structures (as in [38]). Graph traversal in that context does not need to be continuous as it is possible to return to any previously visited vertex in the data-structure. However, in the context of mobile agent exploration, the agent must always follow a path of consecutive edges in the graphs (i.e. jumps are not allowed). Another important difference is that the agent may not always be able to distinguish between the nodes of the graph during the exploration. We will study both exploration of both unlabelled graphs (where the nodes are anonymous) and labelled graphs (where each node is assigned a unique identifier that is visible to the agent visiting that node). Even for labelled graphs, the ability of the agent to recognize previously visited nodes depends on its capabilities, e.g. the size of its memory.

**The Model:** We assume the usual mobile agent model, with an undirected connected graph $G$ where the edges of the graph are labelled with port numbering $\lambda$ so that an agent at any node $v$ can deterministically choose to traverse one of the adjacent edges, $e$, by selecting the port number $\lambda_e$ of that edge. In particular we assume that a *proper port numbering* which assigns the labels $0, 1, 2, \ldots d-1$ to the edges incident to a node of degree $d$. For any such edge-labelled graph $G$, we define the corresponding digraph $\hat{G}$ by replacing each edge $e = (u, v)$ by two directed arcs - one from $u$ to $v$ and the other from $v$ to $u$, marked with the port labels $(\lambda_u(e), \lambda_v(e))$ and $(\lambda_u(e), \lambda_v(e))$ respectively. The nodes of the graph $G$ may be labelled or unlabelled. Throughout this chapter, we will use $n$, $m$, $D$ and $\Delta$ to denote respectively, the number of nodes, the number of edges, the diameter, and the maximum degree of any node, of $G$.

We consider different models for communication and interaction of the agent with the environment, including the whiteboard model, the token model and the face-to-face model. The graph is assumed to be static and does not change during the execution of the algorithm (in particular, there are no failures of any kind). Exploration of dynamic graphs is fully investigated in Chapter 20. This chapter considers only *deterministic* algorithm for exploration of *static* graphs.

**Complexity:** We are interested in efficient algorithms for exploration. We measure the efficiency of an algorithm using three different criteria.

– **[Moves] or [Time]:** The *moves complexity* of an algorithm is the total number of edge traversals performed by all agents, where any single edge traversed by a single agent counts as one move. For exploration by only one agent, the moves complexity is same as the *time* complexity, assuming each edge traversal takes one unit of time (we assume that the graph is unweighted and all edges are similar). When there are multiple agents, we may be interested in the total energy consumption (for exploration by physical robots) or the bandwidth consumption (for software agents exploring communication networks); the moves complexity captures these notions.
– **[Storage]:** The maximum amount of information stored at any node of the graph during the course of the exploration, is called the *storage* complexity of the algorithm. Under the whiteboard model of communication, storage is counted as the minimum size (in bits) of the whiteboard needed at each node of the graph. Under the token model, we count storage as the maximum number of tokens at any node. When the agents do not have the ability to mark the nodes of the graph, we say that the algorithm is a *zero-storage* algorithm.
– **[Memory]:** The *memory* complexity of an exploration algorithm is the maximum size of persistent memory needed by any agent during the exploration. Here *persistent memory* refers to the size of the information carried by the agent when moving from one node to another; usually this does not include the working memory used by the agent while performing computations at any node. If $S$ is the set of states of the agents, then the memory complexity is equal to $\log(|S|)$ bits. If the cardinality of $S$ is a constant, independent of the size of the graph or any other parameters, then the agents are called *finite state automata* and we say that the algorithm is a *constant-memory* algorithm. If the algorithm can be executed by agents having no persistent memory then we say that the algorithm is a *zero-memory* algorithm and that the agents are *oblivious*.

### Bibliographical Notes

The problem of exploring an unknown graph started with the study of mazes, labyrinths and caves, and the need for devising algorithmic strategies to traverse such environments. An early as 1951, Shannon [37] studied exploration by a finite state automaton (a mechanical mouse) moving in a two dimensional maze. Later Budach [8] performed a more rigorous study showing that no automata can explore all mazes. Blum and Kozen [6] showed that either one automaton with just 2 pebbles, or a team of two automata can explore all mazes. These results strongly utilize the orientation information for exploring mazes, thus the results do not hold for arbitrary graphs where it is not possible to use a compass to orientate. Graphs are indeed more difficult to explore than mazes as it was shown by Rollik [36] that any finite team of finite state automata cannot explore all graphs. A tight lower bound of $\Omega(logn)$ on the memory complexity of graph exploration was given by Fraigniaud et al. [30], while a matching upper bound was provided by the result of Reingold [35] who gave a log-space exploration

algorithm for all graphs. The space complexity of exploration can be reduced to $O(\log \log n)$, when the agent is provided with $O(\log \log n)$ distinct tokens; such an algorithm was provided recently by Disser et al. [24] whose idea was to use a sequence of explored nodes as a tape of the Turing machine and store information by writing on the tape using the tokens.

In terms of time complexity of exploration, the fastest exploration for arbitrary labelled graphs was given by Panaite and Pelc [34] which makes $m + O(n)$ moves thus having only a linear penalty with respect to any optimal traversal algorithm which knows the graph in advance. There has been a lot of interest on achieving fast exploration of graphs using a zero-memory agent, by assigning specific port numbering to the edges of the graph, in order to guide the memoryless agent. This is known as *label-guided exploration*. The fastest such exploration makes $4n - 2$ moves [31], while a lower bound of $2.8n - 2$ has been proved [13]. For a constant memory agent, [13] provided a faster exploration with $3.5n$ steps while only a trivial lower bound $2n - 2$ is known for this case. There are schemes for labelling the nodes of a graph for guiding the exploration by a $O(1)$ memory agent. The most storage efficient scheme, given by Cohen et al. [11] provides one bit labels to nodes (i.e. the nodes are colored in black or white) in such a way that an agent with constant memory can explore the graph in $O(m)$ time. When no preprocessing of the graph is allowed, the same article provided a 2 bit labelling scheme such that the exploring agent can itself assign the labels during the exploration, while still achieving an exploration time linear in the number of edges.

There has been some studies on exploration of specific families of graphs, including trees [23], unoriented grids and tori [4], edge-labelled hypercubes [28] and other interconnection graphs [25]. Exploration of graphs with *sense of direction* labelling has been investigated by Flocchini et al. [3].

Exploring directed graphs (digraphs) is more difficult than exploring undirected graphs due to the impossibility of backtracking. For labelled directed graphs that are strongly connected, Deng et al. [20] showed that the time complexity of exploration depends on the so-called *deficiency* of the graph which is the number of edges that need to be added to make the graph Eulerian. The exploration of unlabelled directed graphs was studied by Bender et al. [5] and the best known algorithm for mapping a digraph using a single pebble has a time cost of $O(n^8\Delta^2)$, when the agent knows the size of the graph (or an upper bound). When the agent does not know any upper bound on the size of the graph, at least $\Omega(\log \log n)$ pebbles are necessary, as shown in the above paper.

### Outline of this Chapter

We first look at some basic techniques for exploration of unknown graphs. We then consider algorithms for single agent exploration based on the optimization criteria : time efficiency, moves efficiency, storage efficiency and memory efficiency. We then consider the problem of map construction when exploring unlabelled graphs. Finally we investigate multi-agent algorithms for exploration as well as map construction and show how the latter is related to problems

of gathering and leader election. We conclude with some observations concerning more advanced topics on exploration that have not been considered in this chapter.

## 2 Basic Techniques

*Depth-First-Search (DFS)* : The most standard technique for exploration, called depth-first search, is based on a simple rule: At each node $v$, an agent chooses a unexplored edge (if any) and traverses it; if the agent reaches an already visited node, the agent returns back to node $v$. If there are no unexplored edges at $v$ then the agent backtracks to the node visited prior to arriving at $v$ for the first time. The pseudo-code for this algorithm is given in Algorithm 1.

**Algorithm 1** Algorithm `DFS` (Depth-First Search)

```
if current node v has unexplored edges then
    Traverse the unexplored edge with lowest port number;
    if reached node is already visited then
        Return to previous node v;
else
    if current node v is not the starting node then
        Traverse the edge used to reach v for first time;
```

Some properties of DFS exploration:

- The agent needs to distinguish nodes visited for the first time from nodes previously visited during the exploration.
- For each visited node $v$, the agent needs to know (1) which incident edges at $v$ are still unexplored, and (2) which incident edge was used to enter $v$ for the first time.
- The exploration makes exactly $2m$ moves for graphs with $m$ edges. Thus, this is an asymptotically time optimal algorithm.

*Right-hand-on-the-Wall (RHW)* : The RHW algorithm is a technique used by explorers lost in dark caves, where the idea is to follow the boundary of the cave by feeling it with one hand (assuming the cave is too dark to see anything). A similar strategy can be used for exploring certain port-labelled graphs; the exploration starts at a node $v$ by taking the edge with lowest port number (i.e. port number 0) and at each subsequent step, the agent arriving at a node $u$ through port number $x$, leaves that node by the next port number (either port $x + 1$ or port 0, if $x$ equals the degree of $u$). The algorithm is shown below (Algorithm 2).

Some properties of RHW exploration:

- The algorithm `RHW` performs perpetual exploration on any tree starting from any vertex. This is a zero memory and zero storage algorithm.

**Algorithm 2** Algorithm `RHW` (Right Hand on the Wall)

p := port number of arrival to current node (initially 0);
d := degree of current node;
Traverse the edge with port number $(p+1) \mod d$;

- The exploration on the tree can be terminated on reaching the starting vertex from the port number $d(v)$. Thus, exploration with stop can be performed if the agent has the capability to recognize the starting vertex (e.g. by marking with a pebble).
- The exploration with stop makes exactly $2m$ moves for a tree with $m$ edges ($m = n - 1$).
- On cyclic graphs having nodes of degree $\geq 3$, labelled with arbitrary port numbering, the algorithm may not visit all nodes.

One way of exploring arbitrary connected graphs using the above technique is to perform a preprocessing on the graph $G$ to assign specific port numbering to the edges, that allows an agent performing `RHW` to explore all nodes of this edge-labelled graph. Consider any spanning tree of the graph $G$. For each node $v$ of $G$, we assign port numbers greater than $\deg(v)$ to the non-tree edges and assign port numbers $0, 1, 2, \ldots d-1$ to the tree edges adjacent to $v$, where $d$ is the degree of node $v$ in the spanning tree. An execution of Algorithm `RHW` on this graph[1] would visit all edges of the spanning tree and thus visit all nodes of $G$. A more elegant scheme for assigning a proper port numbering to the edges of any graph to allow the `RHW` algorithm to explore the graph, was provided in [13]. However, note that the preprocessing step of assigning port-numbers requires the intervention of some central authority having prior knowledge of the graph $G$ or at least of a spanning tree of $G$.

*Universal Exploration Sequences (UXS)* : The idea of RHW exploration can be adapted to arbitrary graphs with arbitrary port numbering using the concept of *universal exploration sequences* which is defined below. Given any node $u \in G$, we define the $i$th successor of $u$, denoted by $succ(u, i)$ as 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 universal exploration sequence (UXS) for this graph. A UXS for a family of graphs is a UXS for all graphs in this family. The following result on the existence of UXS is important for us.

[1] Note that since this is not a proper port labelling, the algorithm should have a default action of traversing port number 0 whenever there is no edge with the required port number.

*Property 1.* For any positive integers $n$, there exists a UXS of length $Poly(n)$ for the family of all connected graphs with at most $n$ nodes. Further such a sequence can be computed in polynomial time using a deterministic algorithm.

The above result implies an algorithm for exploration of unlabelled graphs, without the need for any storage at the nodes, provided that the size of the graph is known. The construction for the UXS of polynomial size was given by Reingold [35]. We will call the algorithm that applies the Reingold's UXS for exploration as `RUXS` algorithm. This algorithm has the following properties:

- This is a zero storage algorithm for exploration of unlabelled graphs.
- The exploration requires knowledge of the size of the graph, $n$, or at least an upper bound on $n$.
- The algorithm performs exploration in polynomial time using $O(\log n)$ bits of agent memory.

*Rotor-Router (RR)* : Another technique for exploration by a memoryless agent uses a pointer saved in each node of the graph, which points to the next incident edge to be explored. The agent arriving at a node $v$ simply leaves node $v$ by the edge pointed to by the pointer and at that point the pointer is incremented (modulo the degree) to point to the next incident edge. Such a system is called *Rotor router* (also sometime called *Propp machine*). Given any connected graph $G$ and some port numbering $\lambda$ on $G$, if all pointer values are properly initialized (in a preprocessing step), then the rotor-router process moves the agent on an Eulerian tour of the corresponding digraph $\hat{G}$ obtained from $G$ by replacing each edge with two directed arcs; thus the agent periodically explores all edges of the graph with a period of at most $2m$ moves. On the other hand, if the pointer values have arbitrary initial values, the process takes some time to stabilize and after this stabilization time, the agent follows an Euler tour of the digraph $\hat{G}$ as before. It was shown by Yanovski et al. [40] that the stabilization time is no more than $2mD$ for any graph of diameter $D$. Thus the rotor-router can perform an exploration of the graph in $O(m \cdot D)$ moves, using zero agent memory and $O(\log \Delta)$ storage space per node. This algorithm is *self stabilizing* as both the agent and the nodes of the graph may be in arbitrary states at the start of the algorithm (no initialization is required).

**Algorithm 3** Algorithm `RRA` (Rotor Router Agent)

Let $p$ be the pointer at current node and $e$ be the edge pointed to by $p$
$d :=$ degree of current node;
Set pointer $p := (p + 1) \mod d$;
Traverse the edge $e$;

Some properties of the rotor router algorithm :

- Algorithm `RRA` is a zero memory algorithm for unlabelled graphs.

– The algorithm performs perpetual exploration visiting each node with a period of at most $2m$, after stabilization.
– The algorithm requires $O(\log \Delta)$ bits of storage space per node and it is a self stabilizing algorithm.

## 3 Single Agent Explorations

### 3.1 Time Efficient Explorations

Any algorithm for exploring an unknown graph must visit all edges of the graph. Thus the time complexity or move complexity of exploration is at least $m$. The algorithm `DFS` makes $2m$ moves in total for exploring graphs of $m$ edges. Thus algorithm `DFS` is asymptotically optimal. There has been attempts at reducing further the exploration time for unknown graphs. Panaite et al. [34] gave an exploration algorithm that takes $m + O(n)$ steps in the worst case for arbitrary undirected graphs, using a modified version of `DFS` that reduces the number of times the agent backtracks.

Although the `DFS` algorithm and its variants are asymptotically optimal in time and moves, the algorithm may not be optimal in terms of agent memory and storage, depending on the implementation. Note that the algorithm requires the agent to distinguish the visited nodes from unvisited ones. In labelled graphs, each node has a unique identifier and the agent simply needs to memorize the identifiers of all visited nodes, in order to recognize them. Thus the agent requires $O(n \log n)$ memory but no storage is required. On the other hand if the graph is unlabelled, the agent needs to mark each node that it visits (by writing on the whiteboard, or placing a token) to recognize it as a visited node; this requires $O(1)$ bits of storage per node. In both cases, however, the agent still needs memory to remember the paths already visited to allow it to backtrack; In particular the algorithm remembers a spanning tree of the visited subgraph constructed during the exploration (this is often called the DFS tree). So, the algorithm requires $O(n \log n)$ bits of agent memory. We state the following result based on the standard DFS algorithm [38].

**Theorem 1.** *There is an optimal time algorithm for exploration with stop in unlabelled graphs, that requires $O(1)$ bits of storage per node and $O(n \log n)$ bits of agent memory.*

### 3.2 Storage Efficient Explorations

When the nodes of the graph $G$ are labelled with unique identifiers and these are visible to the exploring agent, it is possible to explore the graph using the `DFS` algorithm using zero storage. This is optimal in terms of storage complexity. However, when the graph is unlabelled, the `DFS` algorithm requires $O(1)$ bits or 1 pebble per node, and thus $n$ pebbles in total. So, the question is whether it is possible to explore unlabelled graphs without marking the nodes. The following result (from folklore) gives a negative answer.

**Theorem 2.** *There is no zero storage algorithm for exploration with stop of unlabelled graphs irrespective of the agent memory.*

*Proof.* Consider two unlabelled graphs: Let $G_1$ be a simple ring of $n_1 = 3$ nodes and let $G_2$ be a line of $n_2$ nodes. Suppose each edge of the line is labelled with port numbers $(1, 2)$ in a consistent manner (e.g. from left to right) and each edge of the ring is labelled with port numbers $(1, 2)$ in the clockwise direction. With this port numbering, all nodes of $G_2$, except the end-points, look exactly like the nodes of $G_1$ to any exploring agent. If there was an algorithm for exploration with stop, consider the execution of this algorithm on $G_1$; the execution must terminates after a finite number of steps $t$. Now if we take $n_2 = 2t + 2$ and we place an agent at the mid-point of the line $G_2$, then the execution of the same algorithm for $t$ steps could visit only nodes at distance at most $t$ from the starting node, thus the agent would never reach either end-point of the line during this time, thus visiting only the nodes which look identical to the nodes of the ring. Thus the algorithm would terminate without visiting all nodes of $G_2$ — a contradiction to the correctness of the algorithm.

Note that the above impossibility is due to the fact that the agent has no prior knowledge of the graph. If the agent knows the size of the graph (either $n$ or $D$, or some upper bound) then it is always possible to perform exploration with stop, without the need to mark the nodes. For example, if the agent known the diameter $D$ of the graph $G$, then an agent starting at node $v$ could systematically traverse all paths of length $D$ starting at node $v$, thus visiting every node of $G$. This is equivalent to traversing the *view* of node $v$ in $G$ to a depth of $D$. When the agent does not know the diameter, the value of $n$ could be used as an upper bound on $D$ and the same procedure would perform an exploration with stop. The time or moves complexity of such an algorithm would be $O(\Delta^n)$ in the worst case, thus this algorithm is exponential in terms of moves/time. It is possible to have a polynomial time exploration with stop using the `RUXS` algorithm as discussed before. The following result follows from the properties of the `RUXS` algorithm.

**Theorem 3.** *[35] There is a polynomial time, zero storage algorithm for exploration with stop for all graphs when the value of n is known a priori.*

*Exploration with no knowledge* : In general, we assume that the agent has no information about the graph that is exploring. When the value of $n$ is not known, or can not be determined, then the agent can repetitively execute the above algorithm, with increasing values of $\hat{n} = 2, 4, 8, \ldots$, where $\hat{n}$ is a guessed value for $n$; when $\hat{n} > n$ all the nodes of the graph would have been visited (without the agent having the knowledge of this fact). This is gives an algorithm for exploration *without termination.*

We now consider storage efficient algorithms for exploration with stop in unlabelled graphs when no prior information about the graph is available. Due to the impossibility result from Theorem 2 we know that that the agent needs to store some information on the nodes. In fact, an agent with a single pebble is capable of performing exploration with stop.

**Theorem 4.** *There is an $O(m \cdot n)$ time and $O(nlogn)$ memory algorithm, using one pebble, for exploration with stop in unknown graphs.*

The algorithm that achieves the above result is a modified version of the `DFS` algorithm. Since the agent has only one pebble, the pebble needs to be reused for marking each new node that is visited. As before, the agent stores in its memory the DFS-tree $T$ which is a spanning tree of the subgraph already explored by the agent. Whenever the agent explores any unexplored edge to reach some node $v$, it places the token on $v$ and performs a full traversal of $T$ - if the token is encountered during the traversal then node $v$ already belongs to $T$ (so, it's not a new node); otherwise $v$ must be a new node. Now, the agent can return to node $v$, recover the token, and continue the exploration. Thus, the agent makes an additional $O(n)$ moves for each edge of $G$, which gives a complexity of $O(m \cdot n)$ moves.

The above result is tight with respect the storage complexity, so we know that zero storage algorithms are possible only for labelled graphs and impossible for unlabelled graphs, while 1 bit of storage (or one pebble) in total suffices to explore unlabelled graphs. Surprisingly, it is possible to perform zero storage exploration, even if at least one of the nodes of the graph is uniquely labelled (and the rest of the nodes are unlabelled). We say that the unique node $v$ is a *landmark*, any agent arriving at $v$ can recognize it immediately as the landmark node. We now present an algorithm for zero-storage exploration when the agent starts at the landmark node and explores the whole graph.

*Exploration with a landmark* : The algorithm is based on the fact that each node $w$ can be uniquely identified using an edge-label sequence $P(v, w)$ corresponding to a path from the landmark $v$ to node $w$. On reaching any node $u$ the agent can detect whether or not the node $u$ is distinct from node $w$ by applying the sequence $P(v, w)$ at node $u$ and checking if it leads to the landmark. Thus, the algorithm maintains a set of so-called *Root-paths*, one for each new node discovered during the algorithm. For each edge explored by the algorithm, the agent needs to detect if the node reached has been already visited - this requires performing the checking procedure mentioned before for each path in the set of stored Root-paths. Thus the agent makes at most $O(n^2)$ moves for each new edge explored by the agent. The exploration proceeds in a breadth-first manner, visiting all nodes at depth $h$ from the landmark, before visiting any node at depth $h + 1$. The algorithm requires $O(n \log n)$ bits of agent memory to store the tree containing all the Root-paths (See [9] for more details).

**Theorem 5.** *There is an algorithm taking $O(m \cdot n^2)$ time and using $O(n \log n)$ bits of agent memory, for exploration with stop in unlabelled graphs containing one landmark node.*

Note that the above algorithm can be implemented using one *unmovable* token (i.e. a one-time use token) that can be placed on the starting node to create the landmark. This requires storage facility at only the starting node of $G$. In contrast, the previous algorithm uses one moveable (i.e. reusable) token

which requires some storage at each node of the graph at some point during the algorithm.

### 3.3 Memory Efficient Explorations

The optimal algorithm for exploration in terms of memory is a zero-memory algorithm, for example, the `RRA` algorithm presented before. The algorithm `RRA` can be used to perform exploration with stop, by adding $O(1)$ bits of the storage at the starting node to distinguish this node and by initializing the pointers at each node to zero on the first visit to the node. The agent can terminate the algorithm when it returns to the starting node and finds that the pointer value is zero. This algorithm requires $O(\log \Delta)$ storage at each node and a single bit of agent memory (to distinguish the starting state). However, if termination is not required then it is possible to have a zero-memory algorithm taking $O(m \cdot D)$ moves in the worst case. Zero-memory agents are like tokens that are moved around by the system and such systems corresponding to many natural physical system (e.g. chip firing games). Thus, there has been a lot of investigations on the properties of such systems. The following result shows the optimality of the `RRA` algorithm in terms of storage and moves.

**Theorem 6.** *[32] Any zero memory algorithm for exploration requires $\Omega(\log \Delta)$ bits of storage per node and makes $\Omega(n^3)$ moves.*

There has been a lot of interest on the minimal memory required to explore unlabelled graphs without any storage. In particular there have been several studies on exploration of unlabelled graphs by constant memory agents (i.e. agents with O(1) bits memory, sometime called finite automata). Whether such agents can explore graphs of arbitrary size (possibly without termination) was an open question for the long time, until the question was answered negatively in [36]. The following result by Fraigniaud et al. [30] gives an exact lower bound for the memory requirement of a single agent exploring unknown graphs without marking.

**Theorem 7.** *[30] Any zero storage algorithm for exploration requires $\Omega(\log n)$ bits of memory for exploration of all graphs of size $n$, even for constant degree graphs.*

The proof of this theorem is based on exploration of regular graphs of degree $\Delta = 3$. Note that in such a graph every node look the same to a visiting agent, thus the action taken by the agent is only a function of its current state. Any agent having $\log k$ bits of memory can be in $k$ distinct states. If such an agent is exploring a graph of $n > k$ nodes then it must enter two distinct nodes of $G$ in the same state, and thus it must exit the node by the same port number in both cases. Based on this fact, it is possible to construct a regular graph $G$ of degree 3 and $k + 1$ nodes where the agent forever moves in a cycle of size less than $k$, thus never visiting the rest of the nodes of $G$.

The lower bound from the above theorem is matched by the algorithm `RUXS` which can explore unlabelled graphs using $O(\log n)$ bits of memory and zero storage. The algorithm requires the knowledge of $n$ or an upper bound, which is necessary in unlabelled graphs due to the impossibility result from Theorem 2. However if the nodes of the graph are labelled with unique identifiers, then the knowledge of $n$ is no longer necessary and the algorithm can be adapted to work without any prior knowledge of the graph, while still using $O(\log n)$ bits of memory.

The only remaining question at this stage, is what is the memory complexity of exploration, when the storage space per node is a small constant. Recall that for storage space of $\Omega(\log \Delta)$, it is already possible to have a zero memory algorithm. So, it is natural to ask if there are algorithms using both constant memory and constant storage per node. If fact, it was shown that having only 2 bits of storage per node is sufficient to circumvent the lower bound of Theorem 6 and explore all graphs using constant memory agents.

**Theorem 8.** *[11] There is a polynomial time algorithm for exploring all graphs using $O(1)$ bits of agent memory and only 2 bits of storage per node, without any prior knowledge of the graph.*

The idea of the algorithm is to preprocess the graph assigning labels from a set of three colors to all the nodes of the graph, such that nodes that are at the same depth from the root (i.e. the starting node) are assigned the same color, which is distinct from the color assigned to nodes one level below and one level above. This coloring of the nodes enables the agent to determine after each move whether it moved closer or further, or remained at the same distance from the starting node. The algorithm enables the agent to traverse a spanning tree of the graph in a depth first search manner, with some additional edge traversals at each node, thus having an overall time complexity of $O(m)$ steps. The labelling of the nodes can be done by the agent during the exploration, provided that each node is initialized (i.e. uncolored) at the beginning. This is the only known algorithm that uses both constant memory and constant storage for exploring unknown graphs.

## 4 Map Construction while Exploration

The problem of *Map construction* requires the agent to output a copy of the graph including all port labels, at the end of the exploration. This immediately implies that: (i) The exploration must terminate, and (ii) the agent must have enough memory to store a copy of the graph (i.e. $\Omega(m \log n)$ bits of memory).

When the nodes of the graph have unique labels, exploration is equivalent to map construction; if the agent can remember a full history of all edges traversed by it, this information is sufficient to reconstruct a map of the graph. This is because each node (and each edge) can be uniquely identified on each visit. However when the graph is unlabelled, this is not always the case. For unlabelled graphs, the algorithm *Exploration-with-a-landmark* from Section 3.2,

can be used to build a map of the graph, since each node can be uniquely identified using its root-path. Thus if the agent is able to mark its hombase with a token, it is possible to solve the problem of map construction. However when marking of nodes is not allowed (i.e. in the Face-to-Face model), it is not always possible to construct a map of the graph, even though it is always possible to perform exploration with stop. In other words, there exists graphs which are not recognizable by an agent even after traversing every edge of the graph and even if the agent has an unbounded amount of memory allowing it to remember everything that it has seen during the exploration.

**Theorem 9.** *[39] There is no zero-storage algorithm for Map-construction in unlabelled graphs, even if the agent has unlimited memory and knows the exact size $n$ of the graph.*

First, consider an agent in a ring of size $n$ where each edge is consistently labelled with port numbers $(0, 1)$ in the clockwise direction. An agent moving in such ring networks cannot distinguish a ring of size $n = 3$ from a ring of size $n = 4$. Thus, clearly map construction is not solvable with only the knowledge of an upper bound on the size of the graph, although this knowledge suffices for exploration as we saw in Section 3.2. Even when the agent knows the exact size $n$, there exists graphs of same size that are indistinguishable. This can be explained using the concept of *graph coverings*. We say that a graph $G$ covers a graph $H$ if there is a homomorphism $\varphi$ mapping nodes and edges of $G$ to nodes and edges of $H$ such that for any edge $e$ between adjacent vertices $u, v \in G$, there is an edge connecting $\varphi(u)$ to $\varphi(v)$ in $H$ ($H$ could potentially be a multi-graph). The quotient graph of any graph $G$ is the smallest multi-graph $B$ such that $G$ covers $B$, under an edge-label preserving graph homomorphism (c.f. Chapter 2). If graphs $G_1$ and $G_2$ cover the same quotient graph $B$ then $G_1$ and $G_2$ are indistinguishable to any mobile agent, as all the information that the agent can gather can be represented by the quotient graph $B$. For example, the Figure 1 below show two graphs of size $n = 16$ which are non-isomorphic, but have the same quotient graph $B$.

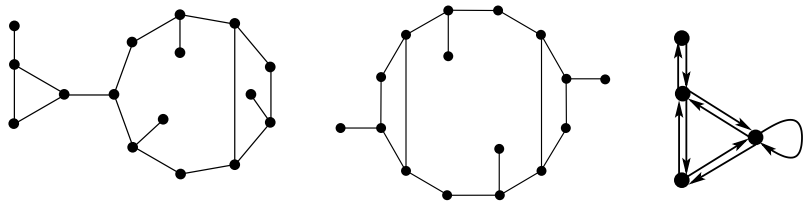

Fig. 1: (i) Graphs $G_1$ and $G_2$ that are indistinguishable to a mobile agent (ii) Graph $B$ that is the quotient graph

It is always possible to construct the quotient graph of any graph, given the knowledge of an upper bound $\hat{n}$ on the size of the graph. For example, an agent can apply the algorithm for *view* construction from Section 3.2 and collapse the view into the quotient graph by merging all 'similar' vertices (i.e. vertices that have the same view up to a depth of $\hat{n}$). Whenever the graph $G$ is identical to its quotient graph then it is possible to construct a map of the graph. Such graphs are said to be *covering minimal*.

**Theorem 10.** *There exists a zero-storage algorithm for Map-construction of any graph $G$ with port-labelling $\lambda$ if (i) the graph $(G, \lambda)$ is covering minimal, and (ii) the agent knows an upper bound on the size $n$ of the graph.*

There exists a more exact characterization of the class of graphs which allow map construction without marking, provided in [39]. We also remark here that it is possible to have a more efficient algorithm for map construction of covering minimal graphs, using the concept of *signatures*, introduced in [14], to identify the nodes. Each node $v$ in such graphs, can be uniquely identified by a sequence of edge-labels encountered by performing by following a UXS path of sufficient length on the graph starting from node $v$. Thus, it is possible to perform `DFS` type exploration with a check procedure for each visited node that computes its signature, and compares it with the signatures of the previously visited vertices.

## 5 Multi-Agent Explorations

When there are multiple agents available, they can together explore the graph to reduce the time taken for exploration. However this requires coordination and communication between the agents, making the task more difficult than single agent exploration. On the positive side, multi-agent exploration can be robust against failures of some agents.

We consider explorations with either colocated agents, or, with agents initially dispersed in the graph. We denote by $k$ the number of agents present in the graph.

### 5.1 Collective Exploration

Collective exploration requires a team of $k$ agents that start from the same location, to explore together all the nodes of the graph, such that each node is visited by at least one of the agents. The agents are assumed to have distinct identifiers such that each agent can be assigned a distinct path to explore. Assuming that all agents move with the same speed (i.e. they are synchronized), the main objective is to minimize the time needed for exploration. When the graph is known in advance, it is possible to devise a strategy to divide the task among the agents such that each agent travels on a distinct tour and they together span the nodes of the graph. We call this an *offline* strategy for exploration; finding the optimal offline strategy that minimizes the maximum tour length of any agent for a given graph $G$ and team size $k$ is known to be an NP-hard problem even for trees [29].

However, we consider the graph to be a priori unknown and the agents need to design and adapt their strategy in an online fashion as they discover new parts of the graph.

Any optimal exploration algorithm using $k$ agent for exploring a graph of diameter $D$ must take at least $O(D + n/k)$ time. When $G$ is tree, Fraigniaud et al.[29] provided a collective algorithm for exploration in $O(D + n/\log k)$ time. The algorithm has a simple strategy, at each node $v$, the available agents are distributed in a round robin manner among the unexplored edges; whenever a subtree has been explored completely all agents in that subtree move to the parent node. The algorithm uses the whiteboard model for communication, thus any agent arriving at a node $v$ can obtain knowledge about the current distribution of agents in the subtree rooted at $v$. The algorithm achieves a competitive ratio of $k/\log k$ over any optimal offline exploration strategy. The best known lower bound for the competitive ratio of any collective exploration algorithm using $k < \sqrt{n}$ agents, is $\Omega(\log k/\log\log k)$, even with global communication between the agents [27].

For graphs of small diameter, fast exploration by small teams of agents can be achieved by a `DFS` based algorithm presented in [7] which has a time complexity of $O(n/k + D^{k-1})$. On the other hand, for large teams of agents, there exists an optimal algorithm for exploring general graphs in $O(D)$ time [21], when the number of agents $k$ is at least $D \cdot n^{1+\epsilon}$ for any $\epsilon > 0$. This algorithm does not require whiteboards for communication and works even in the face-to-face model. The basic strategy is to deploy a fixed number of agents from root at each time step. Exploration of the special class of grid graphs with rectangular holes was studied in [33], which presented a collective exploration algorithm with competitive ratio of $O(log^2 D)$ for such graphs.

### 5.2 Distributed Exploration

When multiple mobile agents start from distinct nodes of the graph, coordination among the agents is more difficult. The task of exploration starting from dispersed locations of the graph is called *Distributed Exploration*. Since the agents do not have a common reference point, direct communication is inutile in this situation. Instead we assume that the agents communicate by writing on the nodes (i.e. the whiteboard model of communication). Notice that if the agents have distinct identities, each agent can individually explore the complete graph (e.g. using the `DFS` algorithm) while marking each visited node with its identifier.

On the other hand if the agents are identical then the marks left on a node by an agent would not be distinguishable from those of another agent. In this case some cooperation between the agents seems necessary. A simple strategy is to use a distributed version of the `DFS` algorithm which we call the distributed depth-first search (`DDFS`) [17].

*Distributed Depth-First Search* : Each agent $a$ performs `DFS` algorithm marking the nodes that it visits (unless they are already marked) and labelling them with a counter that it increments. Each node marked by the agent and the edge used

to reach it are added to DFS-tree stored in the memory of the agent. Note that the agent treat nodes marked by any agent as visited nodes. Thus, whenever the agent reaches an already marked node, it backtracks to the previous node, as in the original algorithm. The tree obtained at the end of the traversal is called the territory $T_a$ of the agent $a$. It was shown in [17] that when all agents have completed the algorithm, 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 have explored all nodes of the graph. The exploration requires $O(m)$ moves in total (instead of $O(m)$ moves per agent if the agents individually explored the graph).

### 5.3 Collision free exploration

When each node of the graph can host at most one agent at any time, then any multi-agent exploration algorithm must prevent collisions (i.e. two agents moving to the same node at the same time). The problem of exploring every node by every agent while ensuring that no node is occupied by more than one agent at any time, is called *Collision-free exploration*. [12] provides such an exploration strategy in labelled graphs for $k$ mobile agents when the mobile agents have 1-hop visibility. The algorithm uses the concept of universal exploration sequences and thus the time complexity is proportional to the length of the UXS. For trees, the paper provides a faster exploration taking $O(n^2)$ time. The algorithms require the agents to start at the same time and always move synchronously so that agents on neighboring nodes swap places without collision.

### 5.4 Map Construction and Leader Election

In Section 4 we saw that Map Construction is possible by a single agent that is allowed to mark the nodes of the graph during exploration. However when there are multiple agents dispersed in the graph, then Map Construction is not always possible, since multiple agents mark several distinct node simultaneously, making it difficult to uniquely identify the nodes. In this case, the possibility of map construction depends on the presence of symmetries in the graph as well as the initial location of the agents. We denote by $b : V \to \{0, 1\}$ a bi-coloring of the graph $G(V, E)$ such $b(v) = 1$ if node $v$ is the homebase of an agent and $b(v) = 0$ otherwise.

**Theorem 11.** *It is possible to solve Map-construction in any graph $G$ with $k$ dispersed agents if $(G, \lambda, b)$ is covering minimal with respect to label preserving and color preserving coverings.*

We now briefly describe an algorithm for map construction by multiple agents in graphs where the above condition is satisfied (see [18] for more details). The map construction algorithm proceeds in two phases. In the first phase, each agent performs the distributed depth-first search (`DDFS`) algorithm discussed previously. At the end of this phase the graph is partitioned into a spanning forest

and each agent $a$ has a map of its DFS-tree $T_a$. The agent obtains its *territory* by adding to this tree, all outgoing edges that are incident to any node of $T_a$. The territory of an agent (including all edge labels) is encoded as an integer $l_a$ that is used by the agent in the subsequent part of the algorithm. The second phase of the algorithm is a competition between neighboring agents, by comparison of the encoded territories. Each losing agent merges its territory with the corresponding winning agent and terminates the algorithm, while each winning agent updates its territory and the same process is repeated with only the winner agents. If the conditions of Theorem 11 hold then there would eventually be a single winner – *the leader* and the territory of this agent would be a spanning tree of the graph. As a final step, nodes of this spanning tree are assigned unique labels (based on the unique path from the root) and thus all non-tree edges can be identified and added to the map. The main complication in this algorithm is the process of synchronizing the agents during each round of the competition phase. There can be at most $O(\log k)$ such rounds in any successful execution of the algorithm and the overall complexity of the algorithm $O(m \log k)$ moves in total. The algorithm fails to construct a map only if the conditions of Theorem 11 do not hold and in those cases, the agent can detect failure after at most $k - 1$ rounds of the competition phase.

The above algorithm also elects a leader among the agents, which is another fundamental task in distributed computing with agents. In fact the problems of the leader election, gathering and map construction in a distributed setting are almost equivalent, with the only exception in the case of *symmetric* trees where leader election may be impossible but map construction is still possible.

## 6 Ongoing Research and Future Directions

This chapter surveyed the main techniques and algorithms for exploration by one or more agents when the agents are deterministic, fault-free and have no constraints on their movements. Similarly, we also assumed that the environment explored is stable and failure-free, allowing any agent to move in any direction on every edge of the graph. This situation is idealistic although in reality several of these assumptions may not hold. In such cases, the task of exploration may become more difficult. We present some directions for further research on the exploration problem, which have been partially investigated.

### 6.1 Constrained Explorations

When mobile agents have constraints on their movements they may not be able to complete the task of exploration in a single attempt. One typical constraint is the budget constraint (i.e. having limited energy for movement) allowing any mobile agent to traverse at most $B$ edges. The problem of *piece-meal exploration* requires an agent to return to its homebase after at most every $B$ edge-traversals, in order to refuel and continue again. In this case, we need to assume that no nodes are at a distance larger than $B$ from the starting location of the agent.

Even with this assumption, the usual techniques for exploration cannot explore the graph efficiently. An algorithm for piecemeal exploration was provided in [1] based on the idea of exploring strips of increasing depths from the starting node, and performing a depth-first search in each strip. This algorithm was later improved upon in [26], achieving an optimal time complexity of $O(m)$ moves in total. However, both these algorithms explore the graph to a depth of $r < B$ and the time complexity increases drastically as $r$ approaches $B$. For piecemeal exploration of graphs of depth $r \leq B$, no efficient algorithm are known. For the family of tree, a piecemeal version of `DFS` algorithm was presented in [16], and shown to be constant competitive (i.e. the algorithm is at most 10 times worse than any optimal offline piecemeal exploration of the same tree).

When the agents are not able to refuel, it is possible to perform constrained exploration using many agents each having a budget of $B$ edge traversals. For any fixed $k$, the best online algorithm for tree exploration using $k$ agents [27] achieves a competitive ratio of $4-2/k$ on the value of $B$ required for exploration. On the other hand, given any fixed $B$, exploration of all trees of depth at most $B$ can be achieved with a competitive ratio of $O(logB)$ for the value of $k$, and this was shown to be asymptotically optimal, at least in the case of local (face-to-face) communication between agents [15]. When both $k$ and $B$ are fixed, it is not possible to completely explore an unknown tree; in this case the problem of *maximal exploration*, which maximizes the number of nodes visited, has been studied [2]. The algorithm in [2] has a competitive ratio of 3, while a lower bound of 2.17 on the competitive ratio was shown in the same paper.

### 6.2 Fault-Tolerant Explorations

While most of the known results are for exploration in fault-free environments, there have been some preliminary investigations on fault tolerant algorithms for exploration. Exploration with faulty tokens that disappear, have been studied in the context of the gathering problem for many agents where each agent uses one token to mark its homebase [19]. Single agent exploration with Byzantine tokens has also been recently studied in [22]. Here, the tokens are unmovable but may sometime be invisible to the agent. If at least one token is fault-free and the agent knows the total number of tokens, there is an exploration algorithm for any unknown and unlabelled graph. Exploration and map construction of *dangerous* graphs containing black holes and black links, can be performed when there are sufficiently many agents [10] using an extended version of the distributed `DFS` based algorithm discussed previously. Under the assumption that the faults do not disconnect the graph, the algorithm builds a map of the fault-free part of the graph whenever it is theoretically possible. Other techniques for exploration of dangerous graphs are covered in Chapter 18.

## References

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

2. E. Bampas, J. Chalopin, S. Das, J. Hackfeld, and C. Karousatou. Maximal exploration of trees with energy-constrained agents. *CoRR*, abs/1802.06636, 2018.
3. L. Barrière, P. Flocchini, P. Fraigniaud, and N.Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40(2):143–162, 2007.
4. H. Becha and P. Flocchini. Optimal construction of sense of direction in a torus by a mobile agent. *Int. J. Found. Comput. Sci.*, 18(3):529–546, 2007.
5. 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'98)*, pages 269–287, 1998.
6. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Symposium on Foundations of Computer Science (FOCS'78)*, pages 132–142, 1978.
7. P. Brass, F. Cabrera-Mora, A. Gasparri, and J. Xiao. Multirobot tree and graph exploration. *IEEE Trans. Robotics*, 27(4):707–717, 2011.
8. L. Budach. Automata and labyrinths. *Math. Nachrichten*, pages 195–282, 1978.
9. J. Chalopin, S. Das, and A. Kosowski. Constructing a map of an anonymous graph: Applications of universal sequences. In *14th International Conference on Principles of Distributed Systems (OPODIS)*, volume 6490 of *LNCS*, pages 119–134, 2010.
10. J. Chalopin, S. Das, and N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *Proceedings of 21st International Symposium on Distributed Computing (DISC)*, pages 108–122, 2007.
11. R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. Algorithms*, 4(4):42:1–42:18, 2008.
12. J. Czyzowicz, D. Dereniowski, L. Gasieniec, R. Klasing, A. Kosowski, and D. Pajak. Collision-free network exploration. *J. Comput. Syst. Sci.*, 86:70–81, 2017.
13. J. Czyzowicz, S. Dobrev, L. Gasieniec, D. Ilcinkas, J. Jansson, R. Klasing, I. Lignos, R. Martin, K. Sadakane, and W. Sung. More efficient periodic traversal in anonymous undirected graphs. *Theor. Comput. Sci.*, 444:60–76, 2012.
14. J. Czyzowicz, A. Kosowski, and A. Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25(2):165–178, 2012.
15. S. Das, D. Dereniowski, and C. Karousatou. Collaborative exploration of trees by energy-constrained mobile robots. *Theory Comput. Syst.*, 62(5):1223–1240, 2018.
16. S. Das, D. Dereniowski, and P. Uznanski. Energy constrained depth first search. *CoRR*, abs/1709.10146, 2017.
17. S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1-3):34–48, 2007.
18. S. Das, P. Flocchini, A. Nayak, and N. Santoro. Effective elections for anonymous mobile agents. In *Proceeding of 17th International Symposium on Algorithms and Computation (ISAAC)*, pages 732–743, 2006.
19. S. Das, M. Mihalák, R. Srámek, E. Vicari, and P. Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *Proc. 12th International Conference on Principles of Distributed Systems (OPODIS)*, pages 463–480, 2008.
20. X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.
21. D. Dereniowski, Y. Disser, A. Kosowski, D. Pajak, and P. Uznański. Fast collaborative graph exploration. *Information and Computation*, 243:37–49, 2015.
22. Y. Dieudonne, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1, 2014.

23. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.
24. Y. Disser, J. Hackfeld, and M. Klimm. Undirected graph exploration with $\ominus(\log \log n)$ pebbles. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 25–39, 2016.
25. S. Dobrev, P. Flocchini, R. Kralovic, P. Ruzicka, G. Prencipe, and N. Santoro. Black hole search in common interconnection networks. *Networks*, 47(2):61–71, 2006.
26. C.A Duncan, S.G. Kobourov, and V.S.A Kumar. Optimal constrained graph exploration. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 807–814, 2001.
27. M. Dynia, J. Lopuszanski, and C. Schindelhauer. Why robots need maps. In *Proc. 14th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 41–50, 2007.
28. P. Flocchini, M. J. Huang, and F. L. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, 2008.
29. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.
30. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.
31. A. Kosowski and A. Navarra. Graph decomposition for memoryless periodic exploration. *Algorithmica*, 63(1-2):26–38, 2012.
32. A. Menc, D. Pajak, and P. Uznanski. Time and space optimality of rotor-router graph exploration. *Inf. Process. Lett.*, 127:17–20, 2017.
33. C. Ortolf and C. Schindelhauer. Online multi-robot exploration of grid graphs with rectangular obstacles. In *24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 27–36, 2012.
34. P. Panaite and A. Pelc. Exploring unknown undirected graphs. *J. Algorithms*, 33:281–295, 1999.
35. O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, September 2008.
36. H. A. Rollik. Automaten in planaren graphen. *Acta Informatica*, 13(3):287–298, 1980.
37. CL. E. Shannon. Presentation of a maze-solving machine. In *8th Conference of the Josiah Macy Jr. Found. (Cybernetics)*, pages 173–180, 1951.
38. R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
39. M. Yamashita and T. Kameda. Computing on anonymous networks: Part i-characterizing the solvable cases. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):69–89, 1996.
40. V. Yanovski, I. A. Wagner, and A. M. Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.