HAL
open science

Structural Information in Distributed Computing

David Ilcinkas

To cite this version:

David Ilcinkas. Structural Information in Distributed Computing. Distributed, Parallel, and Cluster Computing [cs.DC]. Université de Bordeaux (UB), 2019. tel-03099722

HAL Id: tel-03099722

https://theses.hal.science/tel-03099722v1

Submitted on 6 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Structural Information in Distributed Computing

David Ilcinkas

Habilitation thesis

Defense date: March 15, 2019

Reviewers

Evangelos Kranakis	Full Professor	Carleton University, Canada
Laurent Viennot	Research Director	INRIA, Paris
Peter Widmayer	Full Professor	ETH Zurich, Switzerland

Jury

Emmanuel Godard	Full Professor	University of Aix-Marseille
Franck Petit	Full Professor	Sorbonne University
Laurent Viennot	Research Director	INRIA, Paris
Pascal Weil	Research Director	CNRS, Bordeaux
Peter Widmayer	Full Professor	ETH Zurich, Switzerland

Abstract.

This manuscript presents a significant part of my research results in the past years, in the field of distributed computing in networks. The document is structured from the point of view of information available to the computing entities. First, I present my results concerning the advice framework, which allows to study quantitative questions about the a priori information given to the computing entities in order to solve a given problem. I address then the geometric setting, where the computing entities are mobile robots that acquire information about their environment through vision. The third part concerns the exploration of dynamic networks by a mobile entity (called agent in this context), and the impact of knowing the future topological changes. The next part also deals with distributed computing by mobile agents and considers various problems and the different ways of storing information on the network. Finally, the last group of presented results focuses on a particular type of information, the routing one, useful in many contexts to efficiently transfer information in a network.

Contents

1 Introduction **7**

2 The advice framework **11**
2.1 Distributed graph coloring 12
2.2 Broadcasting in radio networks 13

3 Geometric settings **15**
3.1 Graph exploration by oblivious robots 15
3.1.1 The model 16
3.1.2 Obtained results 18
3.2 Robots in geometric terrains 19
3.2.1 Exploration 19
3.2.2 Rendezvous 20

4 Dynamic graphs **23**
4.1 Transportation networks 23
4.2 Rings and cactuses 25

5 Marking nodes **27**
5.1 Graph searching using whiteboards 27
5.2 Pointers and alike for fast graph exploration 28
5.3 Black hole search with pebbles 30
5.4 Tweeking the local orderings 31

6 Routing tables **33**
6.1 On routing information 33
6.2 Constructing rooted shortest-path trees 34
6.3 Compact routing 35

7 Conclusion and perspectives **37**

Chapter 1

Introduction

Distributed computing concerns environments consisting of multiple autonomous computing entities that cooperate in order to collectively achieve a common goal. Contrarily to the parallelism field where the use of multiple computing entities is a choice, motivated by gains in performance, the presence of several computing entities is rather considered as a constraint or as the inherent nature of the environment in distributed computing. The goal then is to compute *despite* the distributed nature of the environment, and in particular despite the lack of direct communications between all entities, the asynchrony of the communications, the presence of faults, or the dynamic nature of the systems.

Distributed systems are nowadays really widespread, in numerous contexts and under various forms. The most famous example is the Internet but networks of computers represent only a part of the distributed systems. In fact, even a single computer is usually a distributed system itself in the sense that processors are constituted of several cores that must cooperate (via shared memory or message passing). Also many objects now contain multiple processors, and/or communicate with each other (the famous Internet of Things).

The research community interested in the foundations of distributed computing has investigated various models and frameworks, motivated by the large variety of existing distributed systems and architectures, and the numerous challenges that they pose. One large part of the research effort concerns distributed systems in which the computing entities, or processes, can directly communicate to each other, either via messages or via a shared memory, but in which communication is asynchronous and processes are subject to faults, typically permanent crashes. In these harsh environments, it is in particular not possible to distinguish a slow process from a crashed one. The main challenges therefore are usually to determine what is computable and what is not, and to evaluate the computation power of various communication primitives and assumptions.

Another important field of research in distributed computing concerns environments in which communication is only possible between certain entities. The system is then classically modeled as a graph, in which the nodes represent the computing entities and in which an edge between two processes represents the possibility for these two processes to directly communicate between each other (usually via message passing). Important questions concern how to efficiently communicate and compute in these networks, even when fault tolerance issues are set aside.

Part of the distributed computing community is also interested in distributed computing by mobile entities, typically in networks. In such a setting, the communication network is only a support for these mobile computing entities called agents, that travel from node to node through the edges of the network. When the mobile entities are operating in continuous environments, like the plane, they are naturally called robots.

In all the various different aspects of distributed computing (definitely not all listed above), the difficulty is to deal with uncertainty, caused by the lack of information about the global configuration. Indeed, by default, the computing entities only have local information and they have to communicate (and/or to move for mobile entities) in order to gather sufficient information to solve the global task at hand.

My research, and this document, concern distributed computing in networks and distributed computing by mobile entities, with a special focus on the information available or not to the computing entities. My work thus participates to the research effort aiming at determining the minimal information and/or model assumptions necessary and sufficient to solve distributed computing problems (possibly with some prescribed performance).

Structure of the document

Chapter 2 concerns the advice framework, in which an external oracle, after looking at the whole instance, gives a bit string of advice to the computing entities before the start of the execution. This allows to quantitatively measure the a priori information on the instance given to the computing entities. The development of this framework was started during my PhD Thesis (cf. [**FIP08, FIP10**]) and continued during my postdoc. These later works contain trade-offs between the amount of a priori information and the efficiency of the solutions, for the problems of distributed graph coloring [**FGIP09**] and of distributed broadcasting in radio networks [**IKP10**].

Chapter 3 deals with mobile computing entities, called robots, moving in continuous or discrete environments, and acquiring information about their environment through vision. The first part of this chapter concentrates on the exploration problem by robots with very limited capabilities (except vision), in discrete environments: the lines [**FIPS11**], the rings [**FIPS13**], and the trees [**FIPS10**]. The second part of the chapter considers more powerful robots, operating in continuous environments, having to solve either the exploration problem [**CILP13**], or the rendezvous problem [**CILP11**].

Chapter 4 concerns the exploration of dynamic networks by a mobile entity, called agent, which is provided with various information about the dynamics of the network. The first presented paper [**IW11**] focuses on periodic transportation networks, which are sparse but very regular. The other two papers presented in this chapter deal with constantly connected networks that are subgraphs of rings [**IW18**] or cactuses [**IKW14**].

Chapter 5 addresses various mobile agent computing problems. In all the presented results of this chapter, the agents are marking the network in which they operate in some ways, in order to (better) solve the problem at hand. One of the most powerful way of storing information in the network is by putting a whiteboard at each node, on which the agents can read and write

information. This model of communication is typically used in the context of the graph searching problem [**INS09**]. A less powerful mark is the pointer, which points at an incident edge on a node. This type of marking is used in the rotor router model [**BGH$^+$17**], which is one way of derandomizing the random walk [**CIKK11**]. The weakest way of communicating via marking the nodes is the pebble, which can be carried by an agent and dropped on or retrieved from nodes. It turns out that such a simple mechanism can turn useful for the black hole search problem instead of the more costly whiteboard mechanism [**FIS12**]. Finally, it is even possible to use the local ordering of the edges to encode some information which allows very simple agents to explore a network [**CDG$^+$12**].

Chapter 6 focuses on a particular type of information, the routing one. More precisely, given a node t, the routing information for node u is a pointer such that following the pointers in the network leads to node t. The results in this chapter concern how to route if some pointers are incorrect [**HIKN10**], how many incorrect pointers can be created by topological changes [**GHI16**], how to construct such pointers in a fault-tolerant way [**GHIJ14, DIJ17**], and how to use these pointers to achieve all-to-all routing using limited memory at the nodes [**GGHI13**].

Chapter 2

The advice framework

In distributed systems, the computing entities usually have access only to local information. It is however very common that problems are difficult or impossible to solve without providing the computing entities with additional global information about the instance, see [Lyn89, FR03]. As a consequence, a lot of algorithmic solutions for various distributed computing problems assume some initial global knowledge about the instance, such as the number of nodes, the maximum degree, or the diameter of the network, or the number of participating agents in mobile agent computing.

For example, the directed graph mapping problem by a single mobile agent in polynomial time only needs one pebble (node marker) if an upper bound on the number of nodes is known to the agent, while $\Theta(\log \log n)$ pebbles are necessary otherwise [BFR+02]. Another example could be the wakeup problem in arbitrary networks, in which a source node must wake up all the other nodes by broadcasting a special wakeup message. It turns out that the number of messages necessary and sufficient to solve the problem is a function $\Theta(n^{1+\Theta(1)/\rho})$ of the radius ρ up to which each node initially knows its neighborhood [AGVP90].

With any solution assuming the knowledge of some global parameter comes the question of the necessity of this knowledge. In some cases, such a knowledge is in fact not necessary. This is typically the case for a large family of classical distributed algorithms, as proven in [KSV13]. In many other cases, one can prove that without this specific knowledge, i.e. without any a priori global knowledge, the problem cannot be solved. This however does not prove that the knowledge of another, maybe "simpler", parameter is not sufficient. One way of answering such questions is to look at the *amount* of knowledge necessary to solve a problem, independently of the *type* of this knowledge. This is exactly the purpose of the advice framework, as it allows quantitative rather than qualitative questions about the initial knowledge of the computing entities.

In the advice framework, an external entity called oracle, after looking at the whole instance, gives a bit string called piece of advice, or simply advice, to the computing entities before the start of the execution. These bit strings may or may not be restricted to be the same for all computing entities, and we talk accordingly about uniform or customized advice. For example, providing the total number of nodes to all computing entities is allowed in the uniform advice framework, while designating a leader by giving advice 1 to a node and 0 to all others is not. The quantity of interest is then the size of advice, which

is usually defined as the sum of the lengths of all the pieces of advice given to the different computing entities. Sometimes, the size of advice is defined as the maximum of these lengths.

This framework was developed during my PhD studies, both in classical distributed computing [**FIP10**] and in mobile agent computing [**FIP08**]. In both cases, the goal was to determine the size of advice necessary and sufficient to solve a given problem with a prescribed efficiency. In fact, the advice framework allows for more complex results, giving general trade-offs between the size of advice and the efficiency of a solution. This is partially the case of the two works done during my postdoc on the subject and presented in this chapter, [**FGIP09**] and [**IKP10**]. The literature about the advice framework is now quite rich, both in mobile agent computing [FIP08, NS09, DP12, DKM12, KKKS15, MP15a, MP15b, GP17a] and in classical distributed computing [FGIP09, FIP10, FKL10, IKP10, FP11, GMP15, FPP16, MP16, GP17b].

Note that the advice framework in distributed computing is closely related to the advice complexity framework introduced in [DKP09] for online algorithms. In distributed computing, only local information is available by default and the advice gives global information. In online computing, only information about the past and the present is available and the advice gives information about the future. More information about these two types of advice can be found in the surveys [DKK$^+$13, Kra14]. The advice framework is also related to the informative labeling schemes, whose purpose is to give labels to each node such that queries about a few nodes can be answered efficiently solely based on the labels of the involved nodes. This includes for example distance labeling schemes [GPPR04] and ancestry labeling schemes [AAK$^+$06].

2.1 Distributed graph coloring

In the distributed graph coloring problem, each node of the graph must choose a color in $\{1, 2, \ldots, c\}$ different from its neighbors, given some integer c. We consider here the $\mathcal{LOCAL}$ model [Pel00]. In this model, nodes have unique identifiers, that we consider here to be from the set $\{1, 2, \ldots, n\}$, where n is the number of nodes. The execution proceeds in synchronous rounds in which a node is able to send an arbitrary message to each of its neighbors, receive the messages from all its neighbors, and perform some arbitrary local computation. There are no a priori limitations on the size of the messages or on the length of the local computations as long as they are finite. In t rounds, a node is thus able to gather all the information from its neighborhood up to distance t, and therefore the $\mathcal{LOCAL}$ model really captures the locality of the problem.

The distributed graph coloring problem has received a lot of interest in the past decades, and in particular the coloring in $\Delta + 1$ colors problem, where Δ is the maximum degree in the network. We restrict our attention here to the distributed 3-coloring of the rings and of the trees. Cole and Vishkin [CV86] proved that 3-coloring of the rings can be solved in $O(\log^* n)$ rounds, where $\log^* n$ is the smallest integer i such that $\log^{(i)} n \leq 2$ (where $\log^{(i)}$ is the i-th iterate of the log function). This upper bound is also valid in oriented tree (that is rooted trees in which each node knows the edge leading to its parent, if any). Linial [Lin92] proved that these upper bounds are tight. He also showed that unoriented trees are much more difficult to distributedly color: the d-regular

unoriented tree of radius r needs $\frac{2}{3}r$ rounds to be colored with fewer than $\frac{1}{2}\sqrt{d}$ colors.

In **[FGIP09]**, we focus on the lower bounds on the size of advice that permits to beat these lower bounds for rings and oriented and unoriented trees. First, we prove that, for any constant k, $\Omega(n/\log^{(k)} n)$ bits of advice are needed to beat the $\Omega(\log^* n)$ time lower bound to 3-color the rings and oriented trees. Differently speaking, almost as much a priori information is needed to beat the $\Theta(\log^* n)$ bound than to directly give their colors to the nodes (which would require $\Theta(n)$ bits).

Similarly, we prove the same lower bound of $\Omega(n/\log^{(k)} n)$ bits of advice (for any k) for reaching time $\Theta(\log^* n)$ in unoriented trees. Again, almost $\Omega(n)$ bits are needed to accelerate coloring of unoriented trees to the speed achieved for oriented trees. Note that this implies that orienting a tree (while allowing few rounds of communication) requires to give almost $\Omega(n)$ bits of information to the nodes (in total).

Finally, let us point out that our lower bounds are not only valid for the size of advice but also for the number of informed nodes, that is the number of nodes receiving at least one bit of advice.

2.2 Broadcasting in radio networks

A radio network is modeled as an n-node directed graph, in which the n nodes represent the radio stations, and an arc (u, v) represents the fact that node v is in the transmission range of u. We assume that nodes have unique identifiers (on $O(\log n)$ bits) and that communication is synchronous (i.e., organized in rounds). A node can be either in the transmitting or in the receiving mode. If a node u transmits a message (in the transmitting mode), then only its out-neighbors may receive the message, and such an out-neighbor v does so only if it is in the receiving mode and u is its only in-neighbor transmitting at this round. If more than one in-neighbors of v transmits at this round, then a collision occurs, and node v (in the receiving mode) may or may not be aware of it, depending whether collision detection is assumed or not.

An important subclass of radio networks is the class of geometric radio networks. In this subclass, nodes are positioned in the plane and are aware of their coordinates. Moreover, each node has a transmission range, taken into a constant-size set of positive reals known to all, and an arc (u, v) exists in the geometric radio network if and only if the Euclidean distance between u and v is smaller than the transmission range of u.

In **[IKP10]**, we consider the broadcasting problem in which a designated source node s has to transmit a message m to all other nodes. We are interested in the completion time, that is on the time at which all nodes have received the message m. More precisely, we look at the size of advice which allows one to broadcast very fast. Our results are the following.

First, we focus on radio networks with constant optimal broadcasting time, i.e. on radio networks in which broadcasting can be achieved in constant time provided that all nodes have the full knowledge of the network. Note that broadcasting in these networks, even restricted to geometric radio networks, can require time up to $\Omega(\sqrt{n})$ when each node knows only its label and coordinates. We prove that $O(n)$ bits of advice are sufficient, but $o(n)$ are not, to achieve

constant broadcasting time in these networks. In other words, one needs to provide almost as much information as giving as advice the rounds at which the node must transmit according to a constant optimal broadcasting schedule. Furthermore, if we restrict ourselves to geometric radio networks (still with constant optimal broadcasting time), then only a constant number of bits is sufficient.

Concerning general n-node radio networks of diameter D, we exhibit a trade-off between the size of advice and the achieved broadcasting time by presenting an algorithm using $q \in O(n)$ bits of advice and broadcasting in time $O(\frac{nD}{q} \log^3 n)$. Note that all our upper bounds do not use collision detection while all our lower bounds are also valid with collision detection.

Chapter 3

Geometric settings

In this chapter, we will focus on distributed computing by mobile entities in geometric settings. In such environments, information can be acquired via vision, and the mobile entities are usually called robots rather than agents. Indeed, a natural application concerns the field of robotics, since robots are usually equipped with vision sensors and are mobile.

The distributed computing community investigates this topic in several complementary directions. A major focus concerns the capabilities of the robots.

Part of the literature concentrates on the minimal capabilities the robots may have for solving non-trivial problems. This gave rise to the oblivious robots in the Look-Compute-Move model. In this model, the agents are very weak, except for their ability to see their environment: they are oblivious (no persistent memory), anonymous, identical, unable to directly communicate with each others, and they do not necessarily agree on a common orientation of the space.

Another part of the literature considers stronger and perhaps more natural models where the robots have better computational capabilities. In particular, they have persistent memory (in reasonable amount) and the studied questions are then more concerning the performance than the computability.

Orthogonally, the research community investigates different types of environments. This goes from discrete environments (modeled as graphs) to continuous environments. In the latter case, the space can have one, two, three, or possibly even more dimensions and the environment may, or may not, contain obstacles.

The first section will focus on oblivious robots in the discrete setting, while the second section will consider stronger models of robots in continuous environments.

3.1 Graph exploration by oblivious robots

In this section, we consider weak robots operating under the Look-Compute-Move paradigm. An activated robot starts first by taking an instantaneous snapshot of its environment (the Look phase), then it computes whether and where it wants to move (the Compute phase), and finally it moves to the decided new position (the Move phase). Robots operating under the Look-Compute-Move paradigm are classically considered in continuous environments, usually the plane. The studies on these robots were however recently extended to the

case of discrete environments, modeled as graphs (see [BD11] and [PBRT11] for short surveys on the subject). This section focuses on these discrete environments.

One motivation to consider discrete environments is to get rid of possibly annoying geometric considerations in order to focus on issues directly related to the weaknesses of the robots (anonymity, obliviousness, etc.), to the symmetries of the environment, and to the asynchrony. Another motivation is more practical and comes from the fact that vision sensors do not have an infinite precision. Considering discrete environments thus acknowledges the fact that many vision sensors output digital and thus discrete snapshots of the environment.

We consider in this section the graph exploration problem, in which the robots have to visit every vertex. More precisely, we focus on *terminating exploration*, which requires that, first, each vertex is visited by at least one robot, and second, that eventually all robots stop moving. Another variant of the problem, introduced in [BBMR08], is also studied in the literature, the *exclusive perpetual exploration*, which requires that, first, each robot visits every vertex of the graph infinitely often, and second, that no two robots traverse the same edge at the same time nor visit the same vertex at the same time. Exploring a graph is a fundamental task in mobile robot computing and can be used, for example, to search for specific information, or to discover and list all the services provided by the nodes.

3.1.1 The model

The environment and the robots. We model the environment as a simple undirected connected graph. The graph is assumed to be anonymous: neither vertices nor edges are labeled (or, equivalently, such labels cannot be seen by the robots). The robots are all anonymous and identical, i.e. they all execute the same algorithm. They have no direct means of communication. Also the robots are assumed to be oblivious: they do not have persistent memory. When several robots occupy the same vertex, we say that there is a *tower* on this vertex.

The Look-Compute-Move cycle. The robots operate by repeatedly executing Look-Compute-Move cycles. In the *Look* phase, a robot takes an instantaneous egocentric snapshot of its environment. This includes the structure of the graph around it, and the presence of robots on the seen vertices. The structure of the snapshot will be detailed later. Note however that all robots are perceived on vertices, not on edges. In the *Compute* phase, the robot decides whether to move or not, and in the first case to which neighboring vertex. Since robots are oblivious, this computation is solely based on the last snapshot. Finally, in the *Move* phase, the robot moves to the chosen neighbor, or stays idle if it decided to do so. Moves are considered instantaneous, which is consistent with the fact that robots are seen on vertices in the snapshots.

Timing assumptions. Different levels of asynchrony are classically considered in the literature. In the fully synchronous model FSync, all robots execute their Look-Compute-Move cycles simultaneously. Differently speaking, at each round, every robot executes its full Look-Compute-Move cycle. In the semi-synchronous model SSync, at each round, a non-empty subset of the robots,

chosen by an adversary, execute a full Look-Compute-Move cycle. Finally, in the asynchronous model ASYNC, the timing between the different phases of the Look-Compute-Move cycles performed by the different robots is arbitrary, with the only constraint that, for each robot, the time between two consecutive phases is finite (but possibly unbounded). As a consequence, a move can be performed based on an outdated snapshot in this model.

The snapshot. During the *Look* phase, a robot perceives the structure of the graph and the presence of robots around it within a *visibility radius* ρ given by the model. More precisely, the snapshot taken by a robot consists of the rooted subgraph induced by the vertices at distance at most ρ from the vertex occupied by the robot and, for each seen vertex, of the perceived number of robots on it. The accuracy of the perceived number of robots is given by another model parameter called the *multiplicity detection*. If weak multiplicity detection is assumed, a robot is only able to distinguish between the presence of "zero", "one", or "more than one" robots on a seen vertex. On the contrary, strong multiplicity detection assumes that a robot knows the exact number of robots that are present on a seen vertex. Orthogonally, multiplicity detection can be either local or global. In the case of local multiplicity detection, a robot only knows the multiplicity of the vertex it occupies (whether in the weak or the strong sense), while in the case of global multiplicity detection, a robot knows the multiplicity of all vertices.

Configurations, Views, and Symmetries. The description of the graph, together with the indication of the exact number of robots located on each vertex, constitute a *configuration*. The *view from vertex* u is any rooted graph isomorphic to the subgraph induced by the vertices at distance at most ρ (the visibility radius) from u, and the corresponding perceived number of robots on these vertices (depending on the multiplicity detection assumption). In the Look phase, a robot is given a view from the vertex it is located on. Therefore, symmetries of the graph are somehow still present in the snapshot. Indeed, for example in a ring, if a robot lies on an axis of symmetry of the configuration, then it will not be able to differentiate one direction from the other. Therefore, if it decides to move to a neighboring vertex, the actual move will be to a neighbor chosen by the adversary. More generally, we say that v and v' are *similar* with respect to u if v and v' are indistinguishable for a robot located in u (because of symmetries and taking into account the visibility radius and the multiplicity detection). If a robot decides in the Compute phase to move to a vertex v, then in the Move phase it will actually move to any vertex v' similar to v with respect to the robot's current position, and the choice of v' is made by the adversary.

Terminating exploration. A team of robots solves the problem of *terminating exploration* in a graph family $\mathcal{G}$ if, for any graph $G \in \mathcal{G}$, for any behavior of the adversary controlling the asynchrony and the choices between similar neighbors, and starting from any initial configuration without towers, each vertex of the graph is visited by *at least one* robot and the robots eventually reach a configuration in which no robots ever move.

3.1.2 Obtained results

In the rings.

In [**FIPS13**], the first paper considering the exploration problem in this model, we focus on the case of the rings. We first prove that terminating exploration of an n-vertex ring is not deterministically solvable by a team of k robots, even with full visibility, global strong multiplicity detection, and in the FSync model. This implies that the smallest number of robots able to explore the n-vertex ring is in $\Omega(\log n)$, for infinitely many values of n. This in fact remains true for probabilistic algorithms [DPT13], but only for the asynchronous model ASync. Indeed, in the semi-synchronous model SSync, a constant number of robots, namely 4 probabilistic robots, are necessary and sufficient to solve the terminating exploration problem in any n-vertex ring with $n > 4$, see [DPT13].

Let us now focus on deterministic algorithms. The lower bound 4 still holds in SSync (in FSync, only 3 is a clear lower bound for every n, in particular when n is odd). When n is even, 5 robots are necessary in the SSync model [LPBT10b]. These values are somehow optimal. Indeed, 4 robots can explore the rings of odd size in SSync [LPBT10a], and provided that n is not a multiple of 5, a team of 5 robots can explore the n-vertex ring even in the ASync model [LPBT10b]. As pointed out, the smallest number of robots that can explore the n-vertex ring may be logarithmic in n for infinitely many values of n, but this cannot go worse in the sense that $O(\log n)$ robots are always sufficient. Indeed, for any $k \geq 17$ that is co-prime with n, we prove that a team of k robots can explore the n-vertex ring [**FIPS13**].

Limited visibility has also been considered, for deterministic algorithms, when the visibility radius ρ is 1 [DLLP13], or 2 or 3 [DLLP15].

In the trees.

The trees [**FIPS10**], and the sub-case of the lines [**FIPS11**], have only been considered in the deterministic setting and assuming weak multiplicity detection.

In trees, the absence of port numbers (the anonymous graph assumption) makes empty leaves having the same parent indistinguishable (they are similar with respect to the parent). Therefore, in order to explore sibling leaves despite any choice of the adversary concerning similar vertices, at least one robot must be sent to each leaf attached to a given parent. If a vertex has more than two leaves attached to it and all of them are occupied by robots, then at least two of them are similar, having both either a single robot or a tower. The adversary can thus make these robots merge if the algorithm decides to move them. Therefore, one can prove that $\Omega(n)$ robots are necessary in some trees (at least in complete ternary trees) in the SSync model. Note that this lower bound heavily relies on the weak multiplicity assumption. For trees of maximum degree 3, less robots may be used: we prove that $O(\log n/\log\log n)$ robots are sufficient in such trees, even in the ASync model. This number is actually necessary for some trees because $\Omega(\log n/\log\log n)$ robots are necessary to explore complete binary trees, even with strong multiplicity detection and in the FSync model.

In lines, symmetries are much more limited, and the solvable cases are fully characterized. A team of $k < n$ robots can solve terminating exploration in the n-vertex line if and only if $k = 3$, or $k \geq 5$, or $k = 4$ and n is odd. The lower

bounds are proved in the SSYNC model while the upper bounds hold even in the ASYNC model.

Other results.

The situation for grids [DLP+12] and tori [DLPT15] has later also been investigated, and resembles the situation for lines and rings, in the sense that grids are explorable using less robots (3 in the general case) than necessary for the tori (4 probabilistically and 5 deterministically), which have more symmetries.

The case of arbitrary graphs has been considered in [CFMS10], in the case of asymmetric configurations and in the presence of port numbers locally distinguishing the edges.

3.2 Robots in geometric terrains

This section is devoted to the case of continuous environments. More precisely, we focus on terrains bounded by a polygon, with possibly polygonal obstacles inside. The polygons are considered part of the terrain, which make it closed in the topological sense. As a consequence, shortest paths between two points of the terrain are well defined, although they may not be unique.

3.2.1 Exploration

In **[CILP13]**, we study exploration of a priori unknown bounded terrains with obstacles by a single robot. We consider two scenarios concerning the capabilities of the robot. In the unlimited vision scenario, a robot visiting (positioned at) some node p explores (sees) any point q such that the segment pq is included in the terrain. In the limited vision scenario, we additionally require p and q to be at distance at most 1. Besides, the robot has persistent memory, that it can use to record its trajectory and the explored portion of the terrain. The trajectory of the robot is assumed to be a polygonal line starting from a point chosen by an adversary, and the cost of exploration, which is the quantity to be minimized, is defined as the length of its trajectory when all points of the terrain are explored.

Exploration has been mostly looked at from the competitive ratio point of view, that is the ratio between the cost of exploration by an algorithm and the optimal cost when having full knowledge about the terrain. In the general case, with unlimited vision, the competitive ratio of any algorithm is at least in $\Omega(\sqrt{k})$ [AKS02], and the best known algorithm has competitive ratio in $O(k)$ [DKP91]. For specific cases, better competitive ratio can be achieved. For example, when obstacles are convex and have a constant aspect ratio (i.e. are not arbitrarily thin), the competitive ratio is in $\Theta(\sqrt{k})$ [KP93].

In the considered paper **[CILP13]**, we rather focus on the asymptotic cost of exploration. For unlimited vision, we present an algorithm with cost in $O(P + D\sqrt{k})$, where P is the total perimeter (including those of the obstacles), D is the diameter of the convex hull of the terrain, and k is the number of obstacles. Note that our algorithm does not a priori know these parameters. We additionally show that this cost is worst-case optimal by proving a matching lower bound, holding even when the terrain is known.

For limited vision, we show a lower bound in $\Omega(P + A + \sqrt{Ak})$, where A is the area of the terrain, again holding even with the full knowledge of the terrain. We also exhibit matching upper bounds (via algorithms) in two cases. The first case concerns arbitrary terrains, but with the knowledge of either A or k. The second case is restricted to the c-fat terrains, for some constant c. A terrain is c-fat if we have $R/r \leq c$, where R, resp. r, is the radius of the smallest disc containing the terrain, resp. the largest disc contained in the terrain.

3.2.2 Rendezvous

The last presented result of this chapter **[CILP11]** is about the asynchronous rendezvous of two robots in bounded terrains, possibly with obstacles. Two robots start at different positions chosen by the adversary in a polygonal terrains and have to meet, i.e. be at the exact same point at the same time. We consider here the asynchronous variant of the problem where the robots choose their trajectories but the adversary chooses their speeds, which may vary between the robots and over time.

The robots are assumed to have persistent memory in sufficient amount, but no vision capabilities. Sensing the environment is done in a tactile way. More precisely, the two robots have a unit of length and are equipped with compasses, which have the same chirality (the same notion of clockwise) but may differ otherwise. When on the boundary of the terrain, a robot knows the angle of the polygonal segment(s) it lies on in its own system of coordinates. The trajectory of a robot is defined step by step. In a given step, the robot chooses a direction and a distance d (in its system of coordinates), and then the robot follows this segment for d of its units of distance, unless the segment is blocked by a boundary or the robot reaches a vertex of a polygon, in which case the robot stops for this step.

In **[CILP11]**, the worst-case cost of rendezvous, defined as the sum of the lengths of the trajectories until they meet, is studied in the eight different scenarios formed by the composition of the following three binary factors: (1) obstacles in the terrain are present, or not, (2) compasses of both robots agree, or not, (3) robots have or do not have a map of the terrain with their positions marked. Note that in symmetric terrains for the scenarios with incoherent compasses and the presence of obstacles, the symmetry cannot be broken if robots start in symmetric positions. In these two scenarios (and in those only), we assume that the robots have distinct identifiers. The optimal worst-case asymptotic complexities are summarized in Table 3.1, where D is the initial distance in the terrain between the two starting positions of the robots, P is the total perimeter (including those of the obstacles), x is the largest perimeter of an obstacle, and l and L are respectively the smaller and the larger labels of the robots.

With incoherent compasses and in the presence of obstacles, rendezvous is achieved on a cycle, and the labels are used to break the symmetry. Otherwise, the robots are able to agree on a single point of rendezvous. When no map is available, the robots need to explore a bit the terrain in order to localize the rendezvous point, which explains the larger complexity than in the scenarios where a map is available.

Rendezvous with a map			Rendezvous without a map		
compasses / obstacles	coherent	incoherent	compasses / obstacles	coherent	incoherent
no	D	D	no	$\Theta(P)$	$\Theta(P)$
yes		$\Theta(D\|l\|)$	yes		$\Theta(P + x\|L\|)$

Table 3.1: Summary of results about the asynchronous rendezvous in bounded terrains

Chapter 4

Dynamic graphs

This chapter is based on the work done with Ahmed Wade when he was my PhD student. It concerns the dynamic graphs, which received a lot of attention recently. There exist many different models, used in various scientific fields, which were surveyed in [CFQS12] (see also [KO11, Mic16]). In this chapter, we use the evolving graph model [Fer04], one of the most classical models of dynamic graphs. An *evolving graph* is a pair $\mathcal{G} = (V, \mathcal{E})$, where V is a static set of n vertices, and $\mathcal{E}$ is a function which maps to every integer $t \geq 0$ a set $\mathcal{E}(t)$ of undirected edges on V. The static graph $G = (V, \bigcup_{t=0}^{\infty} \mathcal{E}(t))$ is called the *underlying graph* of $\mathcal{G}$. Conversely, the evolving graph $\mathcal{G}$ is said to be *based* on the static graph G.

The focus of this chapter will be on the dynamic graph exploration problem, in which a mobile agent has to visit every node of the evolving graph. We assume that traversing an edge requires one time unit. The first section will deal with a special case of evolving graphs called the periodically-varying graphs [**IW11**]. The second section will concern constantly connected evolving graphs based on the rings [**IW18**] and on the cactuses [**IKW14**] (which are trees of rings basically).

4.1 Transportation networks

We consider a system $S = \{s_1, \cdots, s_n\}$ of n *sites* among which k *carriers* are moving. Each carrier c has an identifier $\mathrm{Id}(c)$ and follows a finite sequence $R(c) = (s_{i_1}, \ldots, s_{i_{p(c)}})$ of sites, called its *route*, in a periodic manner. The positive integer $p(c)$ is called the *period* of the carrier c. More precisely, the carrier c starts at node s_{i_1} at time 0 and then proceeds along its route, moving to the next site at each time unit, in a cyclic manner (that is, when c is at node $s_{i_{p(c)}}$, it goes back to s_{i_1} and follows the route again and again).

A PV-graph (for periodically-varying graph) is a pair (S, C), where S is a set of sites, and C is a set of carriers operating among these sites. We will usually denote by n, k and p, respectively, the number of sites, the number of carriers and the maximum over the periods of the carriers. A PV-graph is said to be *homogeneous* if and only if all its carriers have the same period. Finally, a PV-graph is *connected* if one can go from any site to any other site, and it is *highly-connected* if one can go from any site to any other site without staying on a site.

This models in particular various types of public transportation systems like bus systems or subway systems for example. It also models low earth orbiting satellite systems, or security systems composed of security guards making tours in the place to be secured.

On the PV-graphs is operating a mobile agent, not a priori knowing the PV-graph, but able to see the carriers present at its current site, and the identifiers of these carriers. It can board a carrier, move with it, change carrier at a site (if both are present at the same site at the same time), leave a carrier, or stay at a site. We do not assume any restriction on the memory of the agent or on its computational capabilities. The goal of the agent, starting at time 0 in the first carrier, is to visit all sites and terminate (i.e., enter a specific terminal state). We are interested in the completion time, and in the number of performed moves (a move is an edge traversal on a carrier).

The first paper considering exploration of the PV-graphs [FMS13] assumes a slightly different model. Namely, it assumes that the agent cannot wait on a site and must always ride a carrier (but it can obviously change carrier when possible). As a consequence, a PV-graph must be highly-connected to be explorable. This restriction makes sense when considering satellite systems for example. The authors prove that some a priori knowledge is necessary for the problem to be solvable, even if sites have unique identifiers and even in our stronger model, and that knowing an upper bound on p is sufficient in any case. To simplify the presentation of the results, we assume that the agent is provided with a linear upper bound on p.

The time and move complexities obtained in [FMS13] and in our work **[IW11]** are summarized in Table 4.1. In each of the four scenarios, the upper part, resp. lower part, corresponds to the results in [FMS13], resp. in **[IW11]**. In particular, in the general case, we prove that waiting at the stations allows the agent to reduce the worst-case optimal number of moves by a multiplicative factor of at least $\Theta(p)$, while the time complexity is reduced to $\Theta(n \cdot p)$. (In any connected PV-graph, we have $n \leq k \cdot p$.) Our lower bounds hold even when sites have unique identifiers, and our upper bounds are proved by exhibiting algorithms which do not use site identifiers and that perform mapping (i.e., they actually output a full map of the PV-graph). Finally, note that these works were somehow extended to the case of harmful sites which destroy any agent entering them, see [FKMS12a, FKMS12b].

	Connected	Highly-connected
General case	Impossible	$\Theta(kp^2)$ moves and time steps
	$\Theta(\min\{kp, np, n^2\})$ moves $\Theta(np)$ time steps	$\Theta(\min\{kp, np, n^2\})$ moves $\Theta(np)$ time steps
Homogeneous	Impossible	$\Theta(kp)$ moves and time steps
	$\Theta(\min\{kp, np, n^2\})$ moves $\Theta(np)$ time steps	$\Theta(\min\{kp, np, n^2\})$ moves $O(np)$ time steps

Table 4.1: Summary of results about the PV-graph exploration

4.2 Rings and cactuses

In this section, we concentrate on constantly connected evolving graphs, that is on evolving graphs $\mathcal{G} = (V, \mathcal{E})$, where for each $t \geq 0$, the static graph $G_t = (V, \mathcal{E}(t))$ is connected. We also need the notion of T-interval-connectivity, from [KLO10]: given an integer $T \geq 1$, an evolving graph is T-interval-connected if, for every window of T consecutive time steps, there exists a connected spanning subgraph that is stable (always present) during this period. Note that the definitions of 1-interval-connectivity and constant connectivity are equivalent.

In **[IW18]**, we consider T-interval-connected evolving graphs based on a ring. Let us first look at the case when the dynamics is not known by the agents. In general, exploration is impossible, so we assume in this particular case that the evolving graph is δ-recurrent, i.e., that each edge of the underlying graph appears at least once every δ time units. We prove that the worst-case time complexity for the exploration problem is $n + \frac{n}{\max\{1, T-1\}}(\delta - 1) \pm \Theta(\delta)$ time units, which corresponds to the trivial algorithm consisting in going in a fixed direction (say clockwise) until exploration is performed. Exploration by several agents has been considered in [LDFS16], where the impact that synchrony, anonymity and topological knowledge have on the computability and complexity of the problem is studied.

In the rest of **[IW18]** and in **[IKW14]**, we assume that the agent knows the dynamics of the graph. In a sense, the exploration problem becomes a centralized problem in this case. Given an evolving graph and a starting position, determining the optimal exploration time is NP-hard, since it is already the case for static graphs, but it is also hard to approximate [EHK15, MS16] in the general case. This is however not the case for evolving graphs based on rings, for which the optimal exploration time can be computed in time $O(n^2)$ [AKM14].

In fact, even computing the asymptotic worst-case exploration time of a family of (constantly connected) evolving graphs is difficult, contrarily to the case of static graphs which are all explorable in time $\Theta(n)$. An obvious lower bound for evolving graphs is $\Omega(n)$, while the most straightforward upper bound is $O(n^2)$, from the fact that the temporal diameter (the maximum time needed to go from one node to another) is at most n. Obtaining better bounds, even in restricted cases, is a challenging problem.

In **[IW18]**, we prove that evolving graphs based on n-node rings can be explored in time at most $2n - 3$. If these evolving graphs are additionally T-interval-connected, for $2 \leq T \leq \lfloor \frac{n+1}{2} \rfloor$, the worst-case optimal exploration time is more precisely $2n - T - 1$.

Still assuming constant connectivity, evolving graphs based on trees are simply static trees, since no edges can be removed without disconnecting the graph. Besides, static trees can be explored efficiently, traversing each edge at most once in each direction. One could therefore think that evolving graphs based on trees of rings, i.e. on the connected graphs in which any two simple cycles have at most one vertex in common, called cactuses, are also efficiently explorable in time $O(n)$. Surprisingly, it is still unknown whether such a claim is true. In **[IKW14]**, we prove this claim for the case of constant-depth or constant-degree trees of rings. For general underlying cactuses, we are able to prove that the worst-case optimal exploration time is sub-quadratic by exhibiting an exploration algorithm performing exploration in time $2^{O(\sqrt{\log n})}n$. We also show a corresponding lower bound of $2^{\Omega(\sqrt{\log n})}n$ for this specific algorithm.

Chapter 5

Marking nodes

In classical distributed computing, where nodes are the computing entities, information is stored on the nodes. In mobile agent computing though, the memory is naturally placed on the computing entities, that is, on the agents. However, marking nodes turns out to be a potentially powerful indirect mean of communication between the agents. Also, marking nodes can be useful to break symmetry, typically in anonymous networks. This chapter considers the most common different ways of marking the nodes.

5.1 Graph searching using whiteboards

The *graph searching* problem consists, for a team of agents often called searchers, in capturing in a finite graph a fugitive which is invisible, arbitrarily fast, and knows the strategy of the searchers [Par78]. Equivalently, the graph searching problem can be seen as the problem of decontaminating a graph from a toxic gas. Intuitively, the contaminated part of the graph represents the part of the graph where the fugitive can be.

The entire graph is contaminated at the beginning. An edge is cleared when it is traversed by a searcher. An edge e is recontaminated as soon as there exists a contaminated edge e' in the graph and a path without searchers between e and e'. The goal is to minimize the number of searchers, called the search number of the graph in centralized settings, and usually to find a corresponding search strategy.

Two important properties of search strategies are monotonicity and connectedness. A search strategy is *monotone* when no recontamination ever occurs. A search strategy is *connected* if the cleared part of the graph is always connected. These properties define, by requiring them or not, four search numbers: $s(G)$, $ms(G)$, $cs(G)$, and $mcs(G)$. Without requiring the connectedness, recontamination does not help [LaP93], i.e., $ms(G) = s(G)$. This is however not true when requiring connectedness: there exist graphs for which $mcs(G) > cs(G)$ [YDA09]. Besides, note that computing these search numbers is NP-hard [MHG+88].

Most of the literature (see the survey [FT08]) concentrates on the centralized setting, in which the search numbers are defined. Differently speaking, $s(G)$ searchers are sufficient to clear a graph if the searchers know in advance the graph, compute a search strategy, and then execute it in a synchronous way, but

more searchers may be required otherwise. Here, we consider the distributed version of the problem in which the searchers all start from a common node called homebase, move asynchronously and communicate through whiteboards. A *whiteboard* is a local memory, present on every node, in which searchers can read, erase and write in fair mutual exclusion.

Most of the works concerning distributed searching either consider specific topologies or assume that the searchers have some a priori knowledge, modeled as advice, about the graph [NS09]. In our paper **[INS09]**, we are interested in distributed graph searching without any a priori knowledge. In [BFNV08], the authors propose an algorithm using $mcs(G)+1$ searchers that clear any unknown graph in a connected way. Their proposed search strategy is not monotone and may use an exponential number of steps.

In **[INS09]**, we are interested in connected search strategies that are also monotone, which implies a polynomial number of steps. We prove that requiring a monotone connected distributed search strategy without a priori knowledge has a cost in terms of the number of searchers. More precisely, we are interested in finding an algorithm minimizing over all graphs the ratio between the number of searchers used by the algorithm and the search number $mcs(G)$. We prove that any algorithm has competitive ratio in $\Omega(\frac{n}{\log n})$, even when restricting ourself to trees with maximum degree 3, and we present an algorithm with competitive ratio in $O(\frac{n}{\log n})$ using $O(n)$ bits per whiteboard and $O(\log n)$ bits of memory per searcher.

5.2 Pointers and alike for fast graph exploration

The whiteboard is a very powerful way of marking the nodes, allowing any kind of information to be stored in the nodes. A less powerful node mark is the pointer, which points at an incident edge, and can thus be simulated by whiteboards with $O(\log n)$ bits in the worst case. It turns out that a pointer is already a powerful mechanism. Indeed, the pointer is what is used in the rotor router mechanism, at the basis of approaches in stabilization of distributed processes [YWB03], load balancing [BKK$^+$15], and graph exploration [GR08].

We will focus here on the latter problem. In this context, the rotor router mechanism proceeds as follows. Each node has a pointer, and edges are locally ordered at every node. When an agent arrives at a node, it leaves the node using the edge indicated by the pointer. Immediately after the agent leaves the node, the pointer is advanced to the next edge in the local ordering of the node. Differently speaking, the pointer is always pointing to the edge that will be used to leave the node by the agent at its next visit. Note that such an agent's algorithm does not require agent's memory.

The whole system (graph, pointers and position of the agent) being finite, such a process eventually enters a periodic behavior. In particular, the trajectory of the agent becomes periodic, and it happens to be an Eulerian traversal of the symmetric directed version of the graph (each undirected edge $\{u, v\}$ is replaced by two arcs (u, v) and (v, u)), see [PDDK96]. The lock-in time (the number of steps until the agent gets locked-in in its Eulerian traversal) is at most $2mD$ in the worst case [YWB03], where m is the number of edges and D is the diameter of the graph.

In **[BGH$^+$17]**, we study in more depth the lock-in time. First, we exam-

ine the influence of the initial configuration of the pointers and of the cyclic orderings on the lock-in time. This is done via introducing a competition between a *player* $\mathcal{P}$ intending to lock-in the agent in an Euler tour as quickly as possible and its *adversary* $\mathcal{A}$ having the counter objective. We show that the lower bound $\Omega(mD)$ on the lock-in time in the worst-case graph from [YWB03] is in fact valid for any graph if the adversary chooses both the pointers and the cyclic orderings, and even the starting node. We also show that the most critical parameter is the choice of the pointers, in the sense that the adversary can always force a lock-in time in $\Theta(mD)$ in some graphs if it chooses the pointers (cases with $\mathcal{A}(\pi)$, and $\mathcal{A}$-*all*), whereas if the player chooses the pointers after the adversary chooses the cycling orderings (case $\mathcal{A}(\circlearrowright)\mathcal{P}(\pi)$), then the lock-in time is always in $\Theta(m)$. The obtained results are summarized in Table 5.1.

Scenario	Lock-in time (worst-case graph)	Lock-in time (best-case graph)
$\mathcal{P}$-*all*	$\Theta(m)$	$\Theta(m)$
$\mathcal{A}(\circlearrowright)\mathcal{P}(\pi)$	$\Theta(m)$	$\Theta(m)$
$\mathcal{P}(\pi)\mathcal{A}(\circlearrowright)$	$\Theta(m \cdot \min\{\log m, D\})$	$\Theta(m)$
$\mathcal{A}(\pi)\mathcal{P}(\circlearrowright)$	$\Theta(m \cdot D)$,	$\Theta(m)$
$\mathcal{P}(\circlearrowright)\mathcal{A}(\pi)$	$\Theta(m \cdot D)$,	$\Theta(m)$, if $D \leq m^{1/3}$
$\mathcal{A}$-*all*	$\Theta(m \cdot D)$, [YWB03]	$\Theta(m + D^2)$, [best starting node]
		$\Theta(m \cdot D)$, [worst starting node]

Table 5.1: The worst- and the best-case lock-in times in considered scenarios for graphs with m edges and diameter D. The asymptotic results hold for any choice of the starting node in the graph, unless otherwise stated.

Still in [**BGH+17**], we study the impact of failures and topological changes on the lock-in time. More precisely, we start with the system in a locked-in configuration, we introduce pointer faults or topological changes, and we look at the lock-in time from this new configuration. If k pointers are modified, then a new Eulerian cycle is established within $\Theta(m \cdot \min\{k, D\})$ steps. If k edges are added to the graph, then a new Eulerian cycle is established within $\Theta(m \cdot \min\{k, D\})$ steps. If an edge is deleted from the graph but the graph remains connected, then a new Eulerian cycle is established within $O(\gamma m)$ steps, where γ is the length of the smallest cycle in the original graph G which contains the deleted edge.

To summarize, this exploration method is simple and has many good properties. Also, it is deterministic and thus can be seen as a derandomization of the random walk. In particular, the rotor router exploration shares two nice properties with the random walk. First, exploration is performed in a (small) polynomial number of steps, and second, exploration is fair regarding the edges in the sense that the frequencies of visit (in a long run) are equal for all the edges. Indeed, the random walk has cover time (expected number of steps to visit all nodes) in $O(mD \log n)$, and the expected frequency of visit of any edge is $1/m$.

The rotor router exploration can also be seen as the exploration using the Oldest-First strategy: at each step, the agent traverses the incident edge for which the most time has elapsed since its last traversal *in this direction*. The rotor router exploration is also a possible execution of a Least-Used-First strategy in which the agent traverses the incident edge that has been traversed the

least number of times in this direction. One can show that any Least-Used-First strategy for directed edges has also cover time in $O(mD)$ and is fair regarding the edges.

In [**CIKK11**], we study the Oldest-First and Least-Used-First strategies for undirected edges, that is when the time of last traversal or the number of traversals so far are considered *in any direction*. We show that the situation changes significantly. Indeed, the undirected Oldest-First strategy can have a cover time in $2^{\Omega(n)}$ in some graphs and there can be an exponentially large ratio of visits between the most often and the least often visited edges. On the other hand, any undirected Least-Used-First strategy is efficient, with a cover time in $O(mD)$, and fair regarding the edges. Moreover, when the exploration starts from a state with non-zero (corrupted) initial values of traversal counts on edges, the cover time is bounded by $O((n+p)m)$, where p is the maximal value of a counter in the initial state.

5.3 Black hole search with pebbles

The black hole search problem consists, for a team of agents starting from a safe homebase, in locating the black hole(s) located in the network. A black hole is a harmful node that destroys any agent entering it, without letting any observable trace. More precisely, at least one agent must survive, and all surviving agents must terminate knowing the location of the black hole(s). The problem exists in its asynchronous [DFPS07] and synchronous [CKMP07] versions. In the synchronous version, the black hole is relatively easy to detect. If two agents are at node u, neighbor of a node v, then they can determine whether v is a black hole or not: one agent is sent as a scout to v and should return immediately to report, and node v is a black hole if and only if the scout does not come back. In the asynchronous setting, such a trick does not work, because one cannot distinguish between a very slow node and a black hole.

In the asynchronous setting, we assume that there exists exactly one black hole, that the network is 2-node-connected, and that the agents know the number n of nodes. The only way to locate the black hole is then to visit safely $n-1$ nodes. The remaining node must be the black hole. In order to avoid too many losses of agents in the black hole, the agents need a way to communicate with each other, typically to indicate that an agent is trying to traverse an edge which may lead to the black hole, preventing the other agents to also die in the black hole through the same edge. Because of asynchrony and the presence of the black hole, the agents cannot wait for each other and meet on a node. They thus communicate through marking the nodes.

The usual indirect communication model used in asynchronous black hole search is the whiteboard model. The whiteboards are used to mark an edge as dangerous before traversing it, and then safe if there wasn't a black hole at the other extremity, but also to exchange topological information or to split the remaining work between the agents. A natural question is to determine the minimum size of whiteboard which is sufficient to solve the problem. Or perhaps weaker types of node marking are sufficient. In particular, marking an incident edge as dangerous could be done by putting a pointer towards this edge.

In fact, several papers consider the black hole search problem using tokens. A *token* is a marker that can be used both as a pointer and as a pebble (a simple

node marker). Indeed, a token is a marker that can be carried by an agent, put on a node (acting as a pebble) or on the port leading to an incident edge (acting as a pointer), or retrieved by the agent. Using pointers, it was proved that the black hole search problem can be solved with the optimal (for whiteboard) number of agents in rings [DKSS06], in unknown arbitrary graphs [DFKS06], and in hypercubes, tori and complete networks [Shi09]. In the first and third of these papers, the number of moves is even optimal.

Note that in all cases, the ability of a token to act either as a pointer, or as a node, is used. Also, in [DKSS06] and [Shi09], several tokens are allowed to be placed at the same position (node or port), while in [DKSS06] and [DFKS06], edges are supposed to be FIFO, preventing the agents to overtake each other on an edge.

In **[FIS12]**, we prove that black hole search can be solved using only pebbles, that is markers that can be placed only on a node, not on a port leading to an incident edge. Moreover, this can be done in arbitrary graphs of known topology, with the optimal number of agents (two), each starting with only one pebble, and with the optimal number of moves ($O(n \log n)$). Moreover, we do not assume the links to be FIFO, and any agent or node carries/contains at most one pebble at any time. Differently speaking, the pebble model is as powerful as the whiteboard model for the problem of asynchronous black hole search in arbitrary graphs of known topology.

5.4 Tweeking the local orderings

Anonymous graphs are classically defined by graphs whose nodes have no identifiers but whose edges are locally distinguished by port numbers. Usually, the edges incident to a node v of degree d are assigned different port numbers at v from the set $\{1, \ldots, d\}$. Choosing the port number assignment rather than considering any port numbering is also a way of helping mobile agents operating on the graph.

This is in particular the case for periodic graph exploration by an agent with constant (or even no) memory, which cannot explore all graphs with worst-case port numbering. In [DJSS05], the authors prove that, for any n-node graph, there exists an algorithm assigning the port numbers such that an oblivious (memoryless) agent can perform periodic exploration of the graph with period at most $10n$.

In **[CDG+12]**, we prove that the minimum period for oblivious agents is at least $2.8n$ and at most $(4+\frac{1}{3})n-4$ in n-node graphs. When the agent is allowed to have a constant number of memory bits, an even smaller period of at most $3.5n - 2$ can be achieved.

Chapter 6

Routing tables

This chapter focuses on a particular type of information, the routing one. The purpose of this kind of information is to facilitate the transmission of a message between two nodes. Instead of flooding the network with copies of the message to be sure that it arrives quickly, or instead of using a (rather slow) random walk to propagate the message if one does not want to duplicate the message, the routing information on the traversed nodes can be used to decide on which edge to forward the message. This allows one to deliver a message quickly and without duplicating it.

We will first consider a single destination, and look at a simple type of routing information, the pointer. Indeed, a pointer that points to a neighbor on a shortest path is a small information that already allows to deliver a message as quickly as possible and without flooding the network. In the first section, we will see how to route despite the presence of incorrect pointers [**HIKN10**], and then how many incorrect pointers can be created by topological changes [**GHI16**]. In the second section, we will look at self-stabilizing algorithms able to construct such pointers [**GHIJ14, DIJ17**]. Finally, the third section will consider all-to-all routing and how to efficiently route while using limited memory at the nodes [**GGHI13**]. A significant part of these works was done during the PhD studies of Christian Glacet, who I co-supervised with Nicolas Hanusse.

6.1 On routing information

Let us assume that there is a specified destination t in the network, and that each node u different from t has a pointer that should designate a neighbor lying on a shortest path from u to t. Suppose that exactly k nodes, called liars, have incorrect pointers, which are thus pointing at neighbors which are not on shortest paths to t. We are interested in a routing algorithm that allows a message (or equivalently a mobile agent) to be delivered quickly at t despite these incorrect pointers.

This model has been introduced in [KK99] under the assumption that each node may be a liar with a given probability. The model with a fixed number k of liars has been investigated in [HKK04, HKKK08]. It is known that in some trees of constant maximum degree, the presence of k liars forces the expected time to reach a target at distance d to be in $\Omega(d + 2^{\min\{d,k\}})$, when both d

and k are logarithmic in the number n of nodes [HKK04]. In the same paper, the authors prove that much better times can be achieved in other topologies: in a complete network, a ring, a torus, or an hypercube, the target can be reached in time polynomial in both d and k. In [HKKK08], simple randomized and memoryless algorithms are designed for the case of bounded degree graphs: the agent follows the pointer with some fixed probability $p > 1/2$, or chooses one of the remaining incident edges uniformly at random otherwise. The authors show that the expected time to reach t with such an algorithm is in $\Theta(d + r^k)$, where $r = \frac{p}{1-p}$.

In [**HIKN10**], we study strategies that can only choose at each step between following the pointer, and following a pure unbiased random walk (i.e. choosing an incident edge uniformly at random). We propose a particular strategy, called R/A, which makes use of a timer (step counter) to alternate between phases of following a pure random walk (R) and following the pointers (A) for a certain number of steps. No knowledge of parameters n, d, or k is required, and the agent need not know by which edge it entered the node of its current location. The performance of this strategy is studied for two classes of regular graphs with extremal values of expansion, namely, for rings and for random Δ-regular graphs (an important class of expanders). For the ring, R/A is shown to achieve an expected searching time of $2d + k^{\Theta(1)} + o(d)$ for a worst- case distribution of liars, which is polynomial in both d and k. For random Δ-regular graphs, the expected searching time of the R/A strategy is $O(k^3 \log^3 n)$ asymptotically almost surely.

In [**GHI16**], we assume that initially all nodes in the network have correct pointers, and we determine how many liars may be created by some topological changes. In this paper, we assume that edges are weighted, and that the notion of shortest path uses these weights. We consider two scenarios. In the strong adversary model, the initial pointers and the topological changes are chosen by the adversary in order to maximize the number of liars. In the random adversary model, the initial pointers and the topological changes are chosen uniformly at random among the available choices.

We show that in the strong adversary model, for one node or edge deletion (or addition), the numbers of liars and distance changes can be at least $n-D-O(1)$, even if the graph remains connected, where n is the number of nodes and D the initial diameter. In the random adversary model, the expected number of errors after $\mathcal{M}$ edge deletions is bounded by $O(n\mathcal{M}D/m)$, where m is the initial number of edges. We also show that this bound is tight when $\mathcal{M} = o(n)$. Moreover, for $\mathcal{M}$ node deletions, the expected number of errors is in $O(\mathcal{M}D)$.

6.2 Constructing rooted shortest-path trees

Rooted shortest-path trees, which are implicitly formed by (correct) pointers at each node except the destination t, are the basis of many distance-vector routing protocols. In fault-free environments, such data structures are easy to construct, for example using the Bellman-Ford algorithm [Bel58, For56], which propagates the distance to the destination throughout the network. This algorithm (and others) however does not behave well in the presence of faults or topological changes and may lead to the count-to-infinity problem [LGW04].

One way of dealing with faults or topological changes is to use the self-

stabilization approach [Dij74]. A self-stabilizing system is able to recover in finite time, called the stabilization time, a correct behavior starting from any initial configuration (that may be the result of an arbitrary set of transient faults or topological changes). The stabilization time is usually measured in rounds, which more or less captures the speed of the slowest process, but recent studies also consider the move complexity, which counts the number of state changes of a node, summed over all nodes of the network. This quantity thus measures the amount of computation done in the network until stabilization.

We consider in this section the *Disconnected Components Detection and rooted Shortest-Path tree Maintenance* (DCDSPM) problem. It considers a non-necessarily connected network, modeled as a simple undirected and weighted graph, with a distinguished node called the root. The goal is to design a self-stabilizing algorithm which, in the connected components not containing the root, detects this fact (the nodes enter a special state) and, in the connected component containing the root, computes a shortest-path tree rooted at the root (each node has a pointer to a neighbor on a shortest path to the root).

In [**GHIJ14**], we present the first algorithm solving the DCDSPM problem with a linear number of rounds that does not assume any initial knowledge on the parameters of the network, nor restrict the asynchrony of the system. More precisely, the presented algorithm runs in at most $2n + D$ rounds in a network with n nodes and hop-diameter D, and is silent, that is, it eventually reaches a configuration in which the states of the processes do not change anymore. Unfortunately, the move complexity is shown to be exponential in n in the worst-case.

In [**DIJ17**], we improve the previous result by presenting an algorithm having the same good properties but with a polynomial move complexity. The round complexity is a bit larger, at most $3n + D$, while the move complexity is in $O(W_{\max} n^4)$ in networks with integer edge weights at most $W_{\max}$.

6.3 Compact routing

In this section, we consider the full routing problem. The goal is to design a distributed routing scheme, an algorithm that distributedly constructs, in any weighted network, routing tables and a corresponding routing algorithm such that each node can deliver a message to any other node by using the routing algorithm and the routing tables along the taken route.

We are interested in different performance measures, of theoretical and practical interest: the time and message complexities of the routing scheme, the maximum memory space (which includes the routing table) used at each node, and the stretch of the obtained routing algorithm. The stretch of a routing algorithm is the maximum ratio, taken over all pairs source-destination, between the length of the route used by the algorithm between the source and the destination, and the length of a shortest path between the same nodes.

For centralized routing schemes, there are well-established trade-offs between the stretch and the memory. In particular, given a positive integer k, achieving a stretch $< 2k+1$ requires routing tables of $\Omega((n \log n)^{1/k})$ bits, see [AGM06a]. Conversely, given routing tables of size $O(n^{1/k} \cdot \text{polylog}(n))$, the best stretch one can achieve is in $\Omega(k)$, see [AGM06b]. For small stretch with sub-linear routing tables, a space- and stretch-optimal centralized routing scheme has been

proposed in [AGM+08]. It has stretch at most 3 and uses routing tables of $O(\sqrt{n} \cdot \text{polylog}(n))$ bits.

In **[GGHI13]**, we consider distributed routing schemes and we first show that time $\Omega(D)$ (where D is the hop-diameter) is required for any constant stretch, and that shortest-path routing requires $\Omega(n^2)$ bit-message complexity even on sparse graphs of logarithmic diameter. On the positive side, we propose an asynchronous distributed routing scheme for weighted n-node networks of hop-diameter D. The stretch is 7 (and the round-trip stretch is 5). The time complexity is $O(D)$, with a small hidden constant (< 10). Moreover, at any time during the execution of the algorithm, the working memory space of each node is $O(\sqrt{n} \cdot \text{polylog}(n))$. In particular, the routing tables have size $O(\sqrt{n} \cdot \text{polylog}(n))$. In a synchronous scenario, and in the case of uniform weights, the message complexity is $O(m\sqrt{n} + n^{3/2} \min\{D, \sqrt{n}\})$ up to poly-logarithmic factors. For the realistic case of weighted sparse networks of poly-logarithmic hop-diameter, the message complexity is sub-quadratic, in $O(n^{3/2} \cdot \text{polylog}(n))$. A simple variant of our algorithm shows that, for this same family of networks, we can achieve stretch 5 with sub-linear routing tables and sub-quadratic message complexity.

Chapter 7

Conclusion and perspectives

Many of the papers I have published in the last decade are related to the various information that distributed computing entities can acquire in distributed computing in networks, the computing entities being either the nodes or some mobile entities.

This concerns first a priori information, given at the beginning of the execution. Information can be specific, like the size of the network, or the future dynamics of the edges (Chapter 4). But information can be also given as pieces of advice, in order to study quantitative questions about a priori knowledge (Chapter 2).

This also concerns information that can be acquired throughout the execution. In distributed computing by mobile entities in particular, there exist many ways of sensing the environment, like in geometric settings (Chapter 3), or of interacting with it (Chapter 5). In networks, I specifically studied the routing information and on how to construct and use it, even in fault-prone environments (Chapter 6).

Through all these different aspects and points of view, I have contributed to the important quest of the distributed computing community consisting in determining the minimal assumptions necessary and sufficient to solve given problems, possibly with prescribed performance.

Recent developments in distributed computing

More generally, my research fits into the two main tendencies that can be seen in the research community on distributed computing. The first tendency consists in formalizing, generalizing, and clarifying the theoretical aspects of distributed computing. The second tendency consists in getting closer to real life by considering more realistic models and problems.

In order to better determine what can and cannot be done in distributed systems, numerous theoretical models were developed, each focusing on different aspects of the distributed setting. For example, the $\mathcal{LOCAL}$ model focuses on the locality of a problem, that is on how far a node must collect information around it in order to solve the problem. It does so by putting aside many of the limiting factors that may be present in distributed systems: the system is fault-free and synchronous, the processes have unique identifiers, and no particular

constraints concerning their memory or their computational capabilities are assumed. By limiting the size of the messages, one obtain the $\mathcal{CONGEST}$ model, suited for studying the impact of limited bandwidth. Focusing even more on the communication complexity aspect, there is the congested clique model, in which any process can directly communicate with any other process, still with messages of bounded size. An even more extreme model concerning communication is the beeping model, which can be viewed as a model in which all messages are identical. For anonymous networks, one of the appropriate models is the local computation model which allows to concentrate on the symmetry problems induced by the anonymity of the nodes.

There exist in fact many more models in distributed computing and some efforts have been made to clarify their respective power. One direction of research consists in exhibiting simulations in which one model is simulated in another. This may concern the number of faults or processes, like in the BG simulation, the asynchrony assumptions, via the synchronizers, the types of communication primitives, or even the types of computing entities (mobile agents versus nodes). Such a clarification effort has also been made by developing computational complexity theories for distributed computing. For example, concerning the $\mathcal{LOCAL}$ model, different complexity and/or computability classes were introduced to study, in particular, the power of randomization and non-determinism. Similar theories were also developed for other models in distributed computing, like in mobile agent computing for example. Among my works not covered in this document, two are concerning these computability/complexity theories in classical [**AFIM14**] and mobile agent [**BI18**] distributed computing.

Besides, the distributed computing community naturally tends also towards increasing difficulty, trying to better capture the complexity of real life scenarios. This has been done first by considering more and more performance measures. For fault-tolerant systems with shared memory for instance, the main concern gradually switched from the computability to the complexity, by considering the number of registers or the number of remote memory references for example. For self-stabilizing algorithms, more general scenarios about the asynchrony were considered, giving rise to the alternate and complementary move and step complexities. More generally, space complexity received a lot of interest in the last decade or so, and it has been one of the main focus in my own work as well.

In parallel to the already mentioned relatively simple models dedicated to study particular aspects of distributed computing, more realistic and complex models and frameworks were also developed. In particular, there has been recently a rapidly growing interest into taking into account the dynamic nature of the distributed systems, and much still needs to be discovered with this respect. Models and/or algorithms somehow inspired by nature are also developing, ranging from algorithms inspired by animal societies, swarm computing, or even programmable matter and quantum distributed computing. In mobile agent computing, the complexity has increased by considering more elaborate tasks, involving more and more agents, and possibly heterogeneous ones. Among my works not covered in this document, two in particular concern different variants of the search problem by heterogeneous mobile agents [**BCIK15, BCG$^+$16**].

On the confidence in distributed computing

These two tendencies, toward a better understanding of the foundations of distributed computing and toward more realistic models and problems, are both natural and worthwhile exploring. However, for any real progress to be made, one needs to be sure that the obtained results are sound and reliable, and I don't think this is really the case. Indeed, one particularity of distributed computing is that even algorithms of just a few lines may be tricky to prove. The main reason is the intrinsic nature of distributed computing, which involves multiple computing entities, and this phenomenon is exacerbated by the nondeterminism of the executions caused by the asynchrony and/or the faults. On this subject, Leslie Lamport noted in [Lam12]:

> [Concurrent (multiprocess)] algorithms can be quite subtle and hard to get right; their correctness proofs require a degree of precision and rigor unknown to most mathematicians (and many computer scientists). A missing hypothesis, such as that a set must be nonempty, which is a trivial omission in a mathematical theorem, can mean a serious bug in an algorithm.

In fact, not only algorithms are concerned, but also impossibility results and lower bounds. In these, it is rather common to argue about what a computing entity does or does not know about the instance, but reasoning about knowledge and communication tends to be really tricky and prone to incorrect arguments. Whether as authors or reviewers, probably most researchers in distributed computing experienced the difficulty of obtaining proofs without incorrect arguments or at least without holes. (This is anyway my case.) The question is then to determine what confidence we can have in all the claimed results in distributed computing, and more generally in all the distributed systems used more or less directly in our life. The related question is to figure out what researchers can do to improve the situation.

One first issue is that proofs in research papers tend to be written in prose, without always a lot of rigor. Hence, already in the 90s, Leslie Lamport [Lam95] was advocating for a more rigorous way of writing proofs in mathematics and computer science. The recommended method is to write very detailed proofs presented in a hierarchical way in order for the reader to be able to parse them more easily and to catch the different ideas used in the proof at different levels of details. As far as I know, this way of writing a proof was not adopted.

However, model checking did develop in distributed computing, especially for fault-tolerant distributed systems, but non only. Model checking is particularly useful for finding bugs, by completely verifying the algorithms in rather small instances. In distributed computing by mobile robots in networks for example, two algorithms for two different fundamental problems in the field were recently proved incorrect using model checking [BLM$^+$16, DBO16, DBO18]. Proving that algorithms or impossibility results are correct is however much more difficult, because of the combinatorial explosion of the state space or even undecidability issues. Using some tricks, the problem may still be tractable for approximate versions, or for limited cases. Keeping the same example, Bérard et al. [BLM$^+$16] proved correct the main phase of the algorithm from **[FIPS13]** presented in Chapter 3, but only for a limited number of nodes (at most 22).

A complementary method for verifying results is to use a proof assistant like Coq. The certification is not automated anymore, but it is in general not subject anymore to undecidability or combinatorial explosion issues. Several projects using Coq are currently under development concerning various parts of distributed computing. Non exhaustively, the VERDI project considers fault-tolerant distributed systems [WWP+15], the PADEC project considers self-stabilizing algorithms [DCA17], the PACTOLE project considers mobile robots systems [BCR+17], and the LOCO project considers the local computations model [CF11].

I am convinced that this confidence issue in distributed computing is one of the main challenge the field will have to face in the next years, because of the increasingly complex algorithms and results that are and will be developed, and the growing use of distributed algorithms in potentially critical systems. I particularly believe in the certification via proof assistants approach, and I already started and will continue to investigate this aspect in the future.

Bibliography

[AAK+06] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, and T. Rauhe. Compact Labeling Scheme for Ancestor Queries. *SIAM Journal on Computing*, 35(6):1295–1309, January 2006.

[AFIM14] Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. Distributedly Testing Cycle-Freeness. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 15–28. Springer International Publishing, 2014.

[AGM06a] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On Space-stretch Trade-offs: Lower Bounds. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 207–216, New York, NY, USA, 2006. ACM.

[AGM06b] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On Space-stretch Trade-offs: Upper Bounds. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 217–224, New York, NY, USA, 2006. ACM.

[AGM+08] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact Name-independent Routing with Minimum Stretch. *ACM Trans. Algorithms*, 4(3):37:1–37:12, July 2008.

[AGVP90] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A Trade-off Between Information and Communication in Broadcast Protocols. *J. ACM*, 37(2):238–256, April 1990.

[AKM14] Eric Aaron, Danny Krizanc, and Elliot Meyerson. DMVP: Foremost Waypoint Coverage of Time-Varying Graphs. In *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 29–41. Springer, Cham, June 2014.

[AKS02] Albers, Kursawe, and Schuierer. Exploring Unknown Environments with Obstacles. *Algorithmica*, 32(1):123–143, January 2002.

[BBMR08] Roberto Baldoni, François Bonnet, Alessia Milani, and Michel Raynal. Anonymous graph exploration without collision by mobile robots. *Information Processing Letters*, 109(2):98–103, December 2008.

[BCG+16] Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, Ralf Klasing, Tomasz Kociumaka, and Dominik Pajak. Linear Search by a Pair of Distinct-Speed Robots. In Jukka Suomela, editor, *Structural Information and Communication Complexity*, Lecture Notes in Computer Science, pages 195–211. Springer International Publishing, 2016.

[BCIK15] Evangelos Bampas, Jurek Czyzowicz, David Ilcinkas, and Ralf Klasing. Beachcombing on Strips and Islands. In *Algorithms for Sensor Systems*, Lecture Notes in Computer Science, pages 155–168. Springer International Publishing, 2015.

[BCR+17] Thibaut Balabonski, Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Certified Gathering of Oblivious Mobile Robots: Survey of Recent Results and Open Problems. In Laure Petrucci, Cristina Seceleanu, and Ana Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification*, Lecture Notes in Computer Science, pages 165–181. Springer International Publishing, 2017.

[BD11] F. Bonnet and X. Defago. Exploration and Surveillance in Multi-robots Networks. In *2011 Second International Conference on Networking and Computing*, pages 342–344, November 2011.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[BFNV08] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. *Theoretical Computer Science*, 399(1):12–37, June 2008.

[BFR+02] Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. *Information and Computation*, 176(1):1–21, July 2002.

[BGH+17] Evangelos Bampas, Leszek Gasieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, Adrian Kosowski, and Tomasz Radzik. Robustness of the Rotor–Router Mechanism. *Algorithmica*, 78(3):869–895, July 2017.

[BI18] Evangelos Bampas and David Ilcinkas. On mobile agent verifiable problems. *Information and Computation*, 260:51–71, June 2018.

[BKK+15] Petra Berenbrink, Ralf Klasing, Adrian Kosowski, Frederik Mallmann-Trenn, and Przemyslaw Uznanski. Improved Analysis of Deterministic Load-Balancing Schemes. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 301–310, New York, NY, USA, 2015. ACM.

[BLM+16] Béatrice Bérard, Pascal Lafourcade, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, and Sébastien Tixeuil. Formal verification of mobile robot protocols. *Distributed Computing*, 29(6):459–487, November 2016.

[CDG+12] Jurek Czyzowicz, Stefan Dobrev, Leszek Gasieniec, David Ilcinkas, Jesper Jansson, Ralf Klasing, Ioannis Lignos, Russell Martin, Kunihiko Sadakane, and Wing-Kin Sung. More efficient periodic traversal in anonymous undirected graphs. *Theoretical Computer Science*, 444:60–76, July 2012.

[CF11] Pierre Castéran and Vincent Filou. Tasks, types and tactics for local computation systems. *Stud. Inform. Univ.*, 9(1):39–86, 2011.

[CFMS10] Jérémie Chalopin, Paola Flocchini, Bernard Mans, and Nicola Santoro. Network Exploration by Silent and Oblivious Robots. In *Graph Theoretic Concepts in Computer Science*, pages 208–219. Springer, Berlin, Heidelberg, June 2010.

[CFQS12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, October 2012.

[CIKK11] Colin Cooper, David Ilcinkas, Ralf Klasing, and Adrian Kosowski. Derandomizing random walks in undirected graphs using locally fair exploration strategies. *Distributed Computing*, 24(2):91, October 2011.

[CILP11] Jurek Czyzowicz, David Ilcinkas, Arnaud Labourel, and Andrzej Pelc. Asynchronous deterministic rendezvous in bounded terrains. *Theoretical Computer Science*, 412(50):6926–6937, November 2011.

[CILP13] Jurek Czyzowicz, David Ilcinkas, Arnaud Labourel, and Andrzej Pelc. Worst-case optimal exploration of terrains with obstacles. *Information and Computation*, 225:16–28, April 2013.

[CKMP07] Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a Black Hole in Synchronous Tree Networks. *Combinatorics, Probability and Computing*, 16(4):595–619, July 2007.

[CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, July 1986.

[DBO16] Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. Model Checking of a Mobile Robots Perpetual Exploration Algorithm. In *Structured Object-Oriented Formal Language and Method*, pages 201–219. Springer, Cham, November 2016.

[DBO18] Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. Model Checking of Robot Gathering. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, *21st International Conference on Principles of Distributed Systems (OPODIS 2017)*, volume 95 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[DCA17] Stephane Devismes, Pierre Corbineau, and Karine Altisen. A Framework for Certified Self-Stabilization. *Logical Methods in Computer Science*, Volume 13, Issue 4, November 2017.

[DFKS06] Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring an Unknown Graph to Locate a Black Hole Using Tokens. In *Fourth IFIP International Conference on Theoretical Computer Science- TCS 2006*, IFIP International Federation for Information Processing, pages 131–150. Springer, Boston, MA, 2006.

[DFPS07] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile Search for a Black Hole in an Anonymous Ring. *Algorithmica*, 48(1):67–90, May 2007.

[Dij74] Edsger W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, 17(11):643–644, November 1974.

[DIJ17] Stéphane Devismes, David Ilcinkas, and Colette Johnen. Self-Stabilizing Disconnected Components Detection and Rooted Shortest-Path Tree Maintenance in Polynomial Steps. *Discrete Mathematics & Theoretical Computer Science*, Vol. 19 no. 3, November 2017.

[DJSS05] Stefan Dobrev, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Finding Short Right-Hand-on-the-Wall Walks in Graphs. In Andrzej Pelc and Michel Raynal, editors, *Structural Information and Communication Complexity*, Lecture Notes in Computer Science, pages 127–139. Springer Berlin Heidelberg, 2005.

[DKK+13] Stefan Dobrev, Rastislav Kralovic, Richard Kralovic, The Distributed Computing Column, and by P. Fatourou. Computing with Advice: when Knowledge Helps. *Bulletin of EATCS*, 2(110), August 2013.

[DKM12] Stefan Dobrev, Rastislav Královič, and Euripides Markou. Online Graph Exploration with Advice. In *SpringerLink*, pages 267–278. Springer, Berlin, Heidelberg, June 2012.

[DKP91] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 298–303, October 1991.

[DKP09] Stefan Dobrev, Rastislav Kralovic, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theoretical Informatics and Applications*, 43(3):585–613, July 2009.

[DKSS06] S. Dobrev, R. Královič, N. Santoro, and W. Shi. Black Hole Search in Asynchronous Rings Using Tokens. In *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 139–150. Springer, Berlin, Heidelberg, May 2006.

[DLLP13] A. K. Datta, A. Lamani, L. L. Larmore, and F. Petit. Ring Exploration by Oblivious Agents with Local Vision. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 347–356, July 2013.

[DLLP15] A. K. Datta, A. Lamani, L. L. Larmore, and F. Petit. Enabling Ring Exploration with Myopic Oblivious Robots. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 490–499, May 2015.

[DLP+12] Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Optimal Grid Exploration by Asynchronous Oblivious Robots. In *Stabilization, Safety, and Security of Distributed Systems*, pages 64–76. Springer, Berlin, Heidelberg, October 2012.

[DLPT15] Stéphane Devismes, Anissa Lamani, Franck Petit, and Sébastien Tixeuil. Optimal Torus Exploration by Oblivious Robots. In *Networked Systems*, pages 183–199. Springer, Cham, May 2015.

[DP12] Dariusz Dereniowski and Andrzej Pelc. Drawing maps with advice. *Journal of Parallel and Distributed Computing*, 72(2):132–143, February 2012.

[DPT13] Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Theoretical Computer Science*, 498:10–27, August 2013.

[EHK15] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On Temporal Graph Exploration. In *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 444–455. Springer, Berlin, Heidelberg, July 2015.

[Fer04] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, September 2004.

[FGIP09] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 21(6):395–403, March 2009.

[FIP08] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with advice. *Information and Computation*, 206(11):1276–1287, November 2008.

[FIP10] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Communication algorithms with advice. *Journal of Computer and System Sciences*, 76(3):222–232, May 2010.

[FIPS10] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14):1583–1598, March 2010.

[FIPS11] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. How many oblivious robots can explore a line. *Information Processing Letters*, 111(20):1027–1031, October 2011.

[FIPS13] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing Without Communicating: Ring Exploration by Asynchronous Oblivious Robots. *Algorithmica*, 65(3):562–583, March 2013.

[FIS12] Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pebbles. *Algorithmica*, 62(3-4):1006–1033, April 2012.

[FKL10] Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local MST Computation with Short Advice. *Theory of Computing Systems*, 47(4):920–933, November 2010.

[FKMS12a] Paola Flocchini, Matthew Kellett, Peter C. Mason, and Nicola Santoro. Finding Good Coffee in Paris. In *Fun with Algorithms*, Lecture Notes in Computer Science, pages 154–165. Springer, Berlin, Heidelberg, June 2012.

[FKMS12b] Paola Flocchini, Matthew Kellett, Peter C. Mason, and Nicola Santoro. Searching for Black Holes in Subways. *Theory of Computing Systems*, 50(1):158–184, January 2012.

[FMS13] Paola Flocchini, Bernard Mans, and Nicola Santoro. On the exploration of time-varying networks. *Theoretical Computer Science*, 469:53–68, January 2013.

[For56] L.R. Ford. *Network Flow Theory.* Paper P-923. Rand Corporation, 1956.

[FP11] Emanuele G. Fusco and Andrzej Pelc. Trade-offs Between the Size of Advice and Broadcasting Time in Trees. *Algorithmica*, 60(4):719–734, August 2011.

[FPP16] Emanuele G. Fusco, Andrzej Pelc, and Rossella Petreschi. Topology recognition with advice. *Information and Computation*, 247:254–265, April 2016.

[FR03] Faith Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3):121–163, September 2003.

[FT08] Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, June 2008.

[GGHI13] Cyril Gavoille, Christian Glacet, Nicolas Hanusse, and David Ilcinkas. On the Communication Complexity of Distributed Name-Independent Routing Schemes. In *Distributed Computing*, Lecture Notes in Computer Science, pages 418–432. Springer, Berlin, Heidelberg, October 2013.

[GHI16] Christian Glacet, Nicolas Hanusse, and David Ilcinkas. The impact of dynamic events on the number of errors in networks. *Theoretical Computer Science*, 627:1–12, May 2016.

[GHIJ14] Christian Glacet, Nicolas Hanusse, David Ilcinkas, and Colette Johnen. Disconnected Components Detection and Rooted Shortest-Path Tree Maintenance in Networks. In *Stabilization, Safety, and Security of Distributed Systems*, Lecture Notes in Computer Science, pages 120–134. Springer, Cham, September 2014.

[GMP15] C. Glacet, A. Miller, and A. Pelc. Time vs. Information Tradeoffs for Leader Election in Anonymous Trees. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 600–609. Society for Industrial and Applied Mathematics, December 2015.

[GP17a] Barun Gorain and Andrzej Pelc. Deterministic Graph Exploration with Advice. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 132:1–132:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[GP17b] Barun Gorain and Andrzej Pelc. Leader Election in Trees with Customized Advice. *arXiv:1702.03534 [cs]*, February 2017. arXiv: 1702.03534.

[GPPR04] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, October 2004.

[GR08] Leszek Gasieniec and Tomasz Radzik. Memory Efficient Anonymous Graph Exploration. In *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 14–29. Springer, Berlin, Heidelberg, June 2008.

[HIKN10] Nicolas Hanusse, David Ilcinkas, Adrian Kosowski, and Nicolas Nisse. Locating a Target with an Agent Guided by Unreliable Local Advice: How to Beat the Random Walk when You Have a Clock? In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 355–364, New York, NY, USA, 2010. ACM.

[HKK04] Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, February 2004.

[HKKK08] Nicolas Hanusse, Dimitris Kavvadias, Evangelos Kranakis, and Danny Krizanc. Memoryless search algorithms in a network with faulty advice. *Theoretical Computer Science*, 402(2):190–198, August 2008.

[IKP10] David Ilcinkas, Dariusz R. Kowalski, and Andrzej Pelc. Fast radio broadcasting with advice. *Theoretical Computer Science*, 411(14):1544–1557, March 2010.

[IKW14] David Ilcinkas, Ralf Klasing, and Ahmed Mouhamadou Wade. Exploration of Constantly Connected Dynamic Graphs Based on Cactuses. In *Structural Information and Communication Complexity*, Lecture Notes in Computer Science, pages 250–262. Springer, Cham, July 2014.

[INS09] David Ilcinkas, Nicolas Nisse, and David Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, 22(2):117–127, October 2009.

[IW11] David Ilcinkas and Ahmed Mouhamadou Wade. On the Power of Waiting When Exploring Public Transportation Systems. In *Principles of Distributed Systems*, Lecture Notes in Computer Science, pages 451–464. Springer, Berlin, Heidelberg, December 2011.

[IW18] David Ilcinkas and Ahmed M. Wade. Exploration of the T-Interval-Connected Dynamic Graphs: the Case of the Ring. *Theory of Computing Systems*, 62(5):1144–1160, July 2018.

[KK99] Evangelos Kranakis and Danny Krizanc. Searching with uncertainty. In Cyril Gavoille, Jean-Claude Bermond, and André Raspaud, editors, *SIROCCO'99, 6th International Colloquium on Structural Information & Communication Complexity, Lacanau-Ocean, France, 1-3 July, 1999*, pages 194–203. Carleton Scientific, 1999.

[KKKS15] Dennis Komm, Rastislav Královič, Richard Královič, and Jasmin Smula. Treasure Hunt with Advice. In *Structural Information and Communication Complexity*, Lecture Notes in Computer Science, pages 328–341. Springer, Cham, July 2015.

[KLO10] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 513–522, New York, NY, USA, 2010. ACM.

[KO11] Fabian Kuhn and Rotem Oshman. Dynamic Networks: Models and Algorithms. *SIGACT News*, 42(1):82–96, March 2011.

[KP93] Bala Kalyanasundaram and Kirk Pruhs. A competitive analysis of algorithms for searching unknown scenes. *Computational Geometry*, 3(3):139–155, August 1993.

[Kra14] Rastislav Kralovic. Advice Complexity: Quantitative Approach to A-Priori Information. In *SOFSEM 2014: Theory and Practice of Computer Science*, Lecture Notes in Computer Science, pages 21–29. Springer, Cham, January 2014.

[KSV13] Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. *Distributed Computing*, 26(5-6):289–308, October 2013.

[Lam95] Leslie Lamport. How to write a proof. *The American mathematical monthly*, 102(7):600–608, 1995.

[Lam12] Leslie Lamport. How to write a 21 st century proof. *Journal of fixed point theory and applications*, 11(1):43–63, 2012.

[LaP93] Andrea S. LaPaugh. Recontamination Does Not Help to Search a Graph. *J. ACM*, 40(2):224–245, April 1993.

[LDFS16] G. A. D. Luna, S. Dobrev, P. Flocchini, and N. Santoro. Live Exploration of Dynamic Rings. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 570–579, June 2016.

[LGW04] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks*. McGraw-Hill, Inc., New York, NY, USA, 2 edition, 2004.

[Lin92] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, February 1992.

[LPBT10a] Anissa Lamani, Maria Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic ring exploration with oblivious asynchronous robots. *arXiv:0910.0832 [cs]*, 6058:183–196, 2010. arXiv: 0910.0832.

[LPBT10b] Anissa Lamani, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal Deterministic Ring Exploration with Oblivious Asynchronous Robots. In *Structural Information and Communication Complexity*, pages 183–196. Springer, Berlin, Heidelberg, June 2010.

[Lyn89] N. Lynch. A Hundred Impossibility Proofs for Distributed Computing. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC '89, pages 1–28, New York, NY, USA, 1989. ACM.

[MHG+88] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The Complexity of Searching a Graph. *J. ACM*, 35(1):18–44, January 1988.

[Mic16] Othon Michail. An Introduction to Temporal Graphs: An Algorithmic Perspective. *Internet Mathematics*, 12(4):239–280, July 2016.

[MP15a] Avery Miller and Andrzej Pelc. Fast rendezvous with advice. *Theoretical Computer Science*, 608:190–198, December 2015.

[MP15b] Avery Miller and Andrzej Pelc. Tradeoffs between cost and information for rendezvous and treasure hunt. *Journal of Parallel and Distributed Computing*, 83:159–167, September 2015.

[MP16] Avery Miller and Andrzej Pelc. Election vs. Selection: How Much Advice is Needed to Find the Largest Node in a Graph? In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pages 377–386, New York, NY, USA, 2016. ACM.

[MS16] Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, June 2016.

[NS09] Nicolas Nisse and David Soguet. Graph searching with advice. *Theoretical Computer Science*, 410(14):1307–1318, March 2009.

[Par78] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*, Lecture Notes in Mathematics, pages 426–441. Springer, Berlin, Heidelberg, 1978.

[PBRT11] M. Potop-Butucaru, M. Raynal, and S. Tixeuil. Distributed Computing with Mobile Robots: An Introductory Survey. In *2011 14th International Conference on Network-Based Information Systems*, pages 318–324, September 2011.

[PDDK96] V. B. Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian Walkers as a Model of Self-Organized Criticality. *Physical Review Letters*, 77(25):5079–5082, December 1996.

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

[Shi09] Wei Shi. Black Hole Search with Tokens in Interconnected Networks. In *Stabilization, Safety, and Security of Distributed Systems*, Lecture Notes in Computer Science, pages 670–682. Springer, Berlin, Heidelberg, November 2009.

[WWP+15] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas Anderson. Verdi: A Framework for Implementing and Formally Verifying Distributed Systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15, pages 357–368, New York, NY, USA, 2015. ACM.

[YDA09] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, September 2009.

[YWB03] Vladimir Yanovski, Israel A. Wagner, and Alfred M. Bruckstein. A Distributed Ant Algorithm for\protect Efficiently Patrolling a Network. *Algorithmica*, 37(3):165–186, November 2003.