Time and Space-Efficient Algorithms for Mobile Agents in an Anonymous Network

Adrian Kosowski

Time and Space-Efficient Algorithms for Mobile Agents in an Anonymous Network

THÈSE D'HABILITATION À DIRIGER DES RECHERCHES
PRESÉNTÉE À L'UNIVERSITÉ DE BORDEAUX PAR

Adrian Kosowski

APRÈS AVIS DES RAPPORTEURS :

M. Sébastien Tixeuil	Professeur, Université Pierre et Marie Curie
M. Laurent Viennot	Directeur de Recherche, Inria
M. Masafumi Yamashita	Professeur, Kyushu University

LA SOUTENANCE A EU LIEU DEVANT LE JURY COMPOSÉ DE :

M. Philippe Duchon	Professeur, Université Bordeaux 1
M. Leszek Gąsieniec	Professeur, University of Liverpool
M. Cyril Gavoille	Professeur, Université Bordeaux 1
M. Ralf Klasing	Directeur de Recherche, CNRS
M. Sébastien Tixeuil	Professeur, Université Pierre et Marie Curie
M. Laurent Viennot	Directeur de Recherche, Inria

Talence, le 26 septembre 2013

Time and Space-Efficient Algorithms for Mobile Agents in an Anonymous Network

This thesis is composed of 8 research papers, preceded by an extended introduction.

[T1] **Kosowski A.**, A $\widetilde{O}(n^2)$ Time-Space Trade-off for Undirected s-t Connectivity, *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ACM (2013), 1873-1883.

[T2] **Kosowski A.**, Navarra A.: Graph Decomposition for Improving Memoryless Periodic Exploration. *Algorithmica* 63 (2012), 26-38.

[T3] Bampas E., Gąsieniec L., Hanusse N., Ilcinkas D., Klasing R., **Kosowski A.**: Euler tour lock-in problem in the rotor-router model. *Proc. 23rd International Symposium on Distributed Computing (DISC)*, LNCS 5805 (2009), 423-435.

[T4] Bampas E., Gąsieniec L., Klasing R., **Kosowski A.**, Radzik T.: Robustness of the rotor-router mechanism. *Proc. 13th International Conference on Priniciples of Distributed Systems (OPODIS)*, LNCS 5923 (2009), 345-358.

[T5] Cooper C., Ilcinkas D., Klasing R., **Kosowski A.**: Derandomizing random walks in undirected graphs using locally fair exploration strategies. *Distributed Computing* 24:2 (2011), 91-99.

[T6] Chalopin J., Das S., **Kosowski A.**: Constructing a Map of an Anonymous Graph: Applications of Universal Sequences. *Proc. 14th International Conference on Priniciples of Distributed Systems (OPODIS)*, LNCS 6490 (2010), 119-134.

[T7] Czyzowicz J., **Kosowski A.**, Pelc A.: How to meet when you forget: logspace rendezvous in arbitrary graphs, *Distributed Computing* 25:2 (2012), 165-178.

[T8] Czyzowicz J., **Kosowski A.**, Pelc A.: Time vs. space trade-offs for rendezvous in trees. *Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, ACM (2012), 1-10.

Preface

Computing with mobile agents is rapidly becoming a topic of mainstream research in the theory of distributed computing. The objective of most studies in the area is to provide models of agent-based computation with solid mathematical and algorithmic foundations, and to improve our understanding of the capabilities of mobile agents, subject to different constraints. Well-known concepts from parallel and distributed computing, such as anonymity, locality, and synchronicity, can be naturally adapted to fit into models using mobile agents as their processing units. At the same time, solutions to many fundamental problems of distributed computing, such as network discovery or process rendezvous, take on a completely new meaning when computations are performed by mobile agents and require the development of new approaches.

This HDR manuscript provides an overview of some of the work I have performed in the period 2008-2013 in the area of mobile-agent computing, focusing on results in variants of the anonymous network model. The main part of the HDR has been published in the form of 8 research papers [T1–T8]. Questions which we ask and address in these papers are usually of the following kind: What problems admit a solution on an anonymous network? How quickly can an agent solve a given problem? Can an agent solve a problem given only a limited amount of state memory? For the case of a single mobile agent in a graph, we consider the complexity of the tasks of exploring an unknown network, and of learning its topology. The former task is studied due to its immense practical and theoretical significance, while the latter is known to be a form of a "universal problem" for a single agent in an anonymous network, to which all other feasible tasks can be reduced. We then extend our attention to the case of more than one agent, considering the most fundamental coordination routine, namely, the rendezvous problem for a pair of agents.

I have decided to open this HDR with an introductory manuscript, organized into five chapters. Its primary purpose is to set the research contribution of the papers [T1–T8], which form the main part of this HDR, into a broader context. Whereas the literature of mobile agent computing has recently gained at least two valuable monographs [88, 130], there is clearly still room for a separate survey work, devoted specifically to the topic of agent-based computation in anonymous networks. So far, this topic has only been covered in a brief survey article [63] and in several invited conference talks [98, 153], serving mainly as comprehensive literature references. I have written this text with the hope that it may, in the future, be extended and revised into the form of a short monograph on the topic of mobile agent computing in anonymous networks.

One of the difficulties which arise when surveying the literature of mobile agents in anonymous networks is the clear lack of agreement among different authors as to the precise formulation of the studied model. Apparent subtleties of definition, such as deciding whether an agent should be modeled by a Mealy automaton or a Moore automaton, or how to count the size of the memory used by an agent, mean that different results advertised in the literature are not comparable at face value. In some cases, results published in the context of centralized computation (e.g., in models of Jumping Automata for Graphs) have direct corollaries in the anonymous model of mobile agent computing. For the sake of clarity, I have made an attempt to "translate" all of the relevant results from the literature into one consistent model, used for the better part of the manuscript. As a side effect of this process, the reader may find that some of the cited results are stated in this manuscript in a slightly different form than that which appears in the referenced original work.

In terms of advancing research directions, I would consider the general contributions of the thesis to be twofold. Firstly, in the context of graph exploration, we have provided a new description and analysis for exploration strategies which may be seen as viable alternatives to the random walk process. In particular, we have carried out investigations into variants of the rotor-router model and of the Metropolis-Hastings walk, contributing significantly to our understanding of their behaviour in aspects such as: tradeoffs between running time and memory space, regularity of exploration, resilience to network faults [T1–T5]. Secondly, in the context of the theory of computability in anonymous graph, we have provided general routines which can be used by the agent to manipulate the so-called quotient graph, which is a form of a "map" of the anonymous network. This has led us to new efficient approaches to several fundamental problems of agent-based computing (such as synchronous rendezvous and leader election), including the first known agent-based algorithms for solving these tasks which run in logarithmic space, as well as to improved time-space tradeoffs for the rendezvous problem [T6–T8]. To allow the reader to more easily appreciate the original contribution of this HDR, the specific technical contributions of the papers [T1–T8] are summarized in Section 1.4.

Whereas each of chapters of this text is largely stand-alone, the presentation is organized so as to develop one central theme. We start Chapter 2 with a discussion of variants of random walks in exploration problem. From random walks, we move on to universal exploration sequences and other approaches which are built by derandomizing random walks. In Chapter 3, we then use the same universal sequences as the key building-block for quotient graph construction algorithms. Next, in Chapter 4, such quotient graph construction routines are applied to solve the rendezvous problem, moving to the domain of multiple agents in the graph. Other multi-agent problems are briefly covered in Chapter 5, which outlines perspectives for future study. When working on this manuscript in the future, I would like to add one or two full chapters on multi-agent exploration and coordination, taking into account some of our most recent results on these topics [55, 67, 118] and current work-in-progress.

Acknowledgments. Most of the work covered in this thesis was performed while I was working at the LaBRI in Bordeaux, and many of the results are the product of research visits to Liverpool, Ottawa, Marseille, Perugia, and Gdańsk. I would like to thank all of my co-authors and collaborators who were involved directly or indirectly in the work referenced in this HDR, for inspiring research discussions and pleasant scientific visits. In alphabetical order, my sincere thanks go to: Evangelos Bampas, Jérémie Chalopin, Colin Cooper, Jurek Czyzowicz, Shantanu Das, Dariusz Dereniowski, Yann Disser, Leszek Gąsieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, Evangelos Kranakis, Alfredo Navarra, Dominik Pająk, Cristina Pinotti, Andrzej Pelc, Tomek Radzik, Thomas Sauerwald, and Przemysław Uznański.

I am particularly grateful to the Referees, Professors Sébastien Tixeuil, Laurent Viennot, and Masafumi Yamashita, for agreeing to take on their commitment and for reading this HDR manuscript. Some of their comments have already been taken into account when preparing the current revision of the text. I would also like to thank Marek Kubale and Dominik Pająk for valuable suggestions directly related to this manuscript.

Last but not least, I would like to thank my parents and my wife Zuzanna for their constant encouragement and unfailing support.

I gratefully acknowledge the financial support received from Inria (through the project team CEPAGE in Bordeaux), from l'Agence Nationale de la Recherche (projects ALADDIN and DISPLEXITY), from Narodowe Centrum Nauki, Poland (through grant contracts N N206 491738 and NCN DEC-2011/02/A/ST6/00201), the Royal Society, UK (Bordeaux-Liverpool joint project on mobile agent computing), from Consiglio Nazionale delle Ricerche, Italy (Invited Researcher Program), and from the universities in Bordeaux, Marseille, Gatineau, and Gdańsk.

Contents

1 Introduction

The concept of "mobile agent computing" has appeared in theoretical studies and engineering practice, in varied and often completely unrelated contexts. It has been used with respect to designs of autonomous and migrating software units, in the theory of robotics, and even in research into interacting economic agents. Comparing the multifarious definitions of mobile-agent-based designs, the reader may discover that the common elements of these designs may only be expressed in the form of a number of very general features or characteristics. These usually include (cf. e.g. [130]): providing a decentralized description of the system from the perspective of individual agents operating within it, the existence of clearly set out rules governing the mobility of agents and their interaction with other elements of the system, autonomy of actions taken by each of the agents, and sometimes an expectation of protocol resilience after a fault arises in the system.

Such a property-driven description of a mobile-agent system provides us with valuable hints on how to model an agent mathematically. From a theoretical perspective, the agent behaves like an automaton, whose actions at a given moment of time depend on its current state (which represents information known to the agent), metadata relating the agent to the environment (unknown to the agent), and the state of the system in the vicinity in which the agent is operating. The agent occupies a fixed location in its environment, such as a node of a network, and has the ability to move to an adjacent location in the same environment. The model takes its precise shape when we take into account the topology of the operating environment of the agent, and our expectations of the computational capabilities of the agent's processing unit. Sometimes a single mathematical model may be used to describe agents with diverse physical characteristics but behaving similarly at a high level of abstraction, for example, a migrant maintenance process on a computer network, a physical robot exploring a maze, and a data token circulating around a distributed network of processors. On the other hand, a team of robots working in an environment consisting of interconnected passageways, and a team of robots operating in the open plane, will usually be described by completely different models, with the model of the former taking into account the topology of the environment, while the model of the latter relying on its geometric properties.

This work focuses on agents operating in a network. It is set within the framework of agent-based computation which has become prevalent in the literature of Distributed Computing theory of the last decade (cf. e.g. survey papers [98, 153]). The emergence of this field is a direct consequence of the growth of computer networks at a global scale,

which has given rise to new algorithmic challenges related to the search, processing, and dissemination of information. In the studied context, agent-based computing may be perceived as an alternative to local models of distributed computation with processors placed on nodes of a graph [154]. In our considerations, the burden of computation is shifted away from processors located on nodes onto agents equipped with computational capabilities, moving around the nodes of graph. At any given moment of time, each agent is located at a network node, and has the ability to traverse network links, carrying over its state memory to the new location. Such an agent-based description is applicable for many maintenance processes operating in the background in a physical computer network, in a distributed system. More indirectly, in a logical sense, agents can be used to describe background processes which crawl, index, and organize webs of information, such as the world wide web or a social network.

Within the framework of networked agent-based computing, different models take into account various characteristics of the agent. By imposing constraints on some of these parameters and allowing the agent more freedom in others, we can focus on different aspects of the agents' behaviour in real-world scenarios. Such model-dependent constraints may concern any of the following features:

- *the computational capabilities of the agent*, and in particular the size of the state memory of the agent which persists when the agent moves from node to node along a network link,
- *the initial knowledge of the agent at the time of deployment*, which may include a full or partial map of the network environment, knowledge of the agent's location, knowledge of values of global parameters such as the size of the network, etc.
- *the agent's capabilities of perception of the environment*, such as the ability to distinguish nodes or edges, or the view of the environment available to the agent which may include the entire graph or only the neighbourhood of the agent's location
- *the ability of the agent to interact with the network and with other agents* through markers or tokens placed at nodes, information written on whiteboards at nodes, etc.
- *differences between agents operating in the network*, manifested by the presence or absence of unique identifiers of the agents, differences in the speed of traversal of edges, or the potential asynchrony of the clocks governing the operation of different agents.

The precise mathematical model of the system which we consider in this work is that of *anonymous port-labeled networks*. Such networks were introduced in the more general context of distributed computation by Yamashita and Kameda [172], and only later

adapted to include agent-based computation [131]. The key characteristics of this network model include: a lack of *a priori* global knowledge of the network topology, perception of the environment by the agent restricted to only a local view of the network in the neighbourhood of the agent's current location, and *anonymity of nodes* — a requirement that computations performed by the agent cannot make use of any identifying information specific to the node at which the agent is located. The latter assumption, which is arguably the most restrictive, focuses our attention on mobile agent algorithms which behave "uniformly" over all nodes of the network. One classical example of such an algorithm is the *random walk* on graphs, which owes much of its popularity to its efficiency in many practical applications, simplicity of formulation, location-independence, and resilience to minor network faults. Most other algorithmic approaches considered in this work are, at least to some degree, inspired by the goal to alleviate some of the known weaknesses of the random walk, while retaining its desirable properties. Before proceeding to a formal definition of the studied model and providing an overview of our contributions in Sections 1.3 and 1.4, we first give a slightly broader perspective of mobile agent models in theory and practice.

1.1 Research context

The field of agent-based computing in networks takes its roots from several research areas of theoretical computer science, distributed computing, network optimization, and robotics.

The origins of exploration problems in mobile agent computing can be traced back at least a century, to the problem of finding lost treasure in a maze (cf. [45] for a historical survey). The topic became particularly relevant to mainstream theoretical computer science in the 1980s, which saw an intensification of study into solutions to reachability problems. At the focus of attention was the seminal problem of st-Connectivity, which consists in deciding whether two given nodes of a graph, s and t, belong to the same connected component. This problem, which admits a simple solution by strategies such as Depth First Search, becomes considerably more involved when a restriction is imposed on the amount of memory available for computation. The work of Aleliunas *et al.* [7] provided the first efficient logarithmic-space solution to the problem with bounded error, establishing random walks as a viable method for graph exploration problems, as well as introducing the notion of deterministic traversal sequences for graph exploration. A quarter of a century later, Reingold [156] showed the first deterministic graph exploration strategy which works in logarithmic space, resolving in the affirmative the equality of log-space and symmetric log-space complexity classes ($L = SL$). These results, as well most of the literature on the topic, are set in the centralized setting of computation in the RAM model. However, they can also be re-formulated in the JAG model of computation (jumping automata on graphs), introduced by Cook and Rackoff

in 1980 [50]. The main distinction between the RAM and JAG models is that in a JAG, identifiers of nodes may not be used as operands of arbitrary arithmetic operations or comparisons. The JAG model, initially designed as a convenient tool for proving lower bounds and impossibility results, admits a natural interpretation of a JAG as a team of stateful pebbles, placed on nodes of a graph, such that each pebble can be moved along an edge incident to its current location, or teleported to the location of another pebble. Consequently, it is easy to reformulate the JAG model in terms of a team of communicating mobile agents moving around a graph, located at its nodes and traversing its edges in successive time steps. As a matter of fact, many algorithmic results, including the aforementioned works of Aleliunas *et al.* and Reingold, can also be shown to hold when using a JAG with just one pebble moving along the edges of a graph, which means that these problems can be solved by deploying a single mobile agent in a graph.

Considerations of mobile agents from a distributed perspective have led to additional concerns, resulting from the need to coordinate a team of agents. In this context, agents admit properties analogous to those of processors in a distributed system. Agents can either operate synchronously, moving among the network nodes in synchronous rounds governed by a central clock, or asynchronously, with each agent setting its own pace. The concept of faults in the system may be applied both to agents (leading to notions such as "Byzantine agents" [101]), as well as to nodes or links, which may possibly damage the agent they are hosting (so-called "black holes" [74, 120]). Likewise, the lack of unique identifiers is a concern both in the context of agents (which may, but need not have unique identifiers), and locations in the network. In particular, the model of an anonymous network, formalized by Yamashita and Kameda [172] in the context of distributed computing, provides a convenient setting for studying location-oblivious algorithms for mobile agents, which is central to this study. The anonymous network model restricts the agent's capabilities by imposing constraints of anonymity and locality of view, but at the same time it guarantees some form of persistence of information, since the agent can carry its memory state over network links. The size of this memory can be restricted, but even then, the agent is usually assumed to be capable of basic navigation, e.g, of reverting the last move it has performed.

Some related models of mobile agents seek to capture the inherent possibility of loss of state information by the agent as a result of system faults. This has led to an interesting line of study of models in which agents are oblivious, and their knowledge is restricted to a (possibly delayed) perception of the state of the system, known as a *snapshot*. The typical life cycle of such an agent is described by an iterated sequence of three phases: a *Look* phase when the agent performs a snapshot, a *Compute* phase when it considers its next action, and a *Move* phase during which it is transported to an adjacent node. Such approaches are a natural extension of self-stabilizing algorithms in distributed computing, since it is expected that an acceptable final configuration can be reached from any initial starting configuration. Results in the area concerns such

problems as gathering at a single point [112, 117, 119] or perpetual exploration of the graph [70, 135], and have been obtained for both geometric and networked scenarios and in models differing in synchronicity and atomicity of operations [62].

Many challenges faced by mobile agents in networks have their counterparts in geometric settings. Geometric agents, moving around a continuous terrain, are more frequently referred to by the term *robots*, and have been the object of study in robotics, autonomous system design, operations research, and distributed computing. For a single robot, tasks which have been studied include exploring an unknown environment (e.g. with the goal of constructing a map or locating an unknown target) and patrolling the interior or the perimeter of a known terrain. Exploration and patrolling tasks naturally generalize to scenarios with a team of multiple cooperating robots; such multi-robot coverage tasks are a promising and important direction of future research (cf. Chapter 5). The application of multiple robots has also led to the design of algorithms for performing numerous coordination routines, such as establishing visual contact among a team of robots, as well as the task of gathering geometric robots at a single point [49,89], or more generally, formation of a geometric pattern with a group of robots [90,163,174]. Some geometric scenarios can be related to a networked setting by discretizing the terrain in which the robots operate, which usually takes the form of a polygon with holes or obstacles, and modeling it as a grid-type graph, in a process known as skeletonisation. In this way, for example, some geometric patrolling problems can be modeled as problems of vehicle routing in graphs or the graphic Traveling Salesman Problem [8], and problems of approximate meeting of two robots on the Euclidean plane can sometimes be solved using rendezvous algorithms for networks [60]. Nevertheless, models of geometric robots may include characteristics which cannot be easily captured in a networked setting. The observation capabilities of robots on the plane may correspond to the line of sight of the robot (and possibly a bounded angle or range of vision, depending on its heading). A physical robot may have non-zero dimensions, potentially influencing the rules of its own motion, and obscuring vision and obstructing movements of other robots [59]. On the other hand, when the dimension of the robot are treated as negligible, tasks such as exploration of a two-dimensional terrain or meeting with another robot in the absence of orientation points can only be solved in an approximate manner [60]. Whereas computations in networked models involve problems related to symmetries of the graph, the sense of direction of a robot on the plane is usually controlled by robots' compasses, which may potentially be faulty or uncoordinated among different robots [162]. Geometric scenarios also call for a continuous description of robot's motion, requiring a different definition of asynchrony than network models, allowing for robots with variable speeds. Some notions, such as limited battery power (or tank capacity) of the robot, incurring the need for refuelling from a depot, have been carried over from geometric scenarios to networked scenarios [3, 78].

Techniques which appear in the analysis of mobile agents can also be found in other

areas of distributed computing and network theory. The most closely connected topics include routing protocols in the presence of extremely limited routing data (e.g., compact routing, and routing with invalid information [103, 104]), as well as protocols related to token circulation in a distributed system. Among the latter, one especially interesting example is the randomized self-stabilizing solution to the mutual exclusion problem proposed by Israeli and Jalfon [110] (cf. also later improvements [107]). In it, tokens circulate in a network of processors following independent random walks, and only nodes hosting a token perform non-empty operations. Such a protocol can effectively be considered to be a deployment of a system of mobile agents over a network of processors.

Finally, we remark on some game-theoretic aspects of mobile agent computing. When more than one mobile agent is operating in the network, and all the agents have a common goal, the behavior of the agents may be viewed as a form of a collaborative game played by the agents. On the other hand, it is easy to imagine contexts in which individual agents or sub-teams of agents have opposing goals. This includes different search games, also known as games of pursuit-evasion or predator-and-prey [12]. The usual form of a search game on a graph is one in which a team of white agents (known as guards or cops) move around the graph in an attempt to find and capture a single black agent (the robber or the intruder). This problem has been studied under various assumptions, concerning e.g. the relative speed and visibility of the agents or requirements on some form of monotonicity of the search strategy. It displays beautiful connections to both structural graph theory, relating to parameters such as pathwidth [116], and to complexity theory in the analysis of games of graphs. In geometric models, it is possible to design interesting game scenarios with only a single predator and a single prey, leading to problems known under colorful names such as “the problem of the princess and the monster”, or “the lion and man problem” posed by Rado in 1925 (cf. [138]).

1.2 Mobile agents in networking practice

Computational models which rely on mobile agents focus on describing the operation of the system from the perspective of individual agents, rather than from the perspective of network nodes. Protocols governed by this “mobile agent paradigm” are usually used as subroutines of more complex tasks, related to network communication, navigation, transportation, and security.

From a modeling perspective, a mobile agent is an entity located at any moment of time at a network node, equipped with some state information which remains intact when the agent traverses network links, and with the ability to modify its own state while located at network nodes. The nodes of the network are treated merely as passive carriers, whose role is limited to propagating the agent or the team of agents operating in the network. As such, a node which does not contain an agent at a given moment

is not permitted to perform any computations at that time. Such a constraint is necessary when defining protocols for networks in which nodes are logical entities with no computational capabilities, e.g., webs of knowledge or social networks, but it may also prove technologically justified by efficiency concerns in physical computer networks.

When considering implementation of an agent-based protocol on a computer network, the agent usually materializes in the form of a piece of software or a packet of data. During deployment, the mobility of an agent among the hosts of the network may be realised in one of two distinct forms. In one case, the agent is a process physically running outside the network it is logically placed in, which merely operates on an external data structure, representing the structure of network. In this case, the agent's algorithm needs to "pull" from the network data structure the data corresponding to the agent's current location. This type of "pull" paradigm lies at the heart of the design of crawling robots for indexing the world wide web [25, 52]. In such a process, a robot running on an external server downloads a web page, parses it, proceeds along a link to download a subsequent web page, etc. — thus realizing mobility through a web of information without ever actually changing the server on which it is running. Webs of interconnected data obviously cannot host an agent physically, hence only pull-type approaches are feasible. This includes, for example, robots for evaluating the relative importance of nodes in a web using PageRank-type algorithms [25, 150], robots proposing new connections in a social network through so-called supervised random walks [24], and robots for finding cuts and clusterings of the network [177]. For implementations related to social networks, a model of operation of a robot called "crawl-and-jump" has recently been introduced [40], where the "crawl" phase corresponds to an agent located at a node of the social network moving along a friendship connection to a neighboring node, while the "jump" phase represents a global move to a randomly chosen node of the social network. Such an approach has been used e.g. to describe algorithms for choosing a representative sample of nodes from among those active in the social network [38], and in particular to evaluate how strongly a node of a social network can influence the opinion of other nodes in the network when propagating a rumour between direct neighbours. A related random-walk based approach is used in tasks of sampling in social networks, in which the goal is to find the value of a metric of the network's population (such as average age, income, or number of friends), by only crawling and polling a subset of the nodes [99, 134, 136]. The crawl-and-jump approach, from a theoretician's perspective, is very closely related to the classical JAG model in centralized computing, or the anonymous network model studied in this manuscript (cf. Section 1.3). At the level of modeling social networks and webs of data, the idea that pieces of information seem to propagate along random walks or paths in the web has proved to be a useful tool, improving our understanding of the flow of information on the web [115, 121].

The second type of mobility of the agent corresponds to situations in which the agent's environment is a network of physical hosts — servers or processors. Such hosts

can “push” the agent from node to node, conveying the agent’s memory state (and perhaps also the code of its algorithm) along the communication links of the network. As a rule, each node, having “processed” an agent, transmits it to precisely one other node, and returns to the same state of operation in which it was before receiving the agent. Nevertheless, situations in which the agent is a complete piece of software, passed on from machine to machine, are relatively rare in practice. Such a paradigm of mobile or migrant software was widely studied in the literature in the 1990’s, but never gained widespread adoption, most notably due to security concerns connected to running mutable software received from other hosts. It is much more common to find applications in which all of the network hosts are pre-installed with the immutable logic of the agent, while only the agent’s mutable memory state is transmitted among nodes. Such a process resembles a routing process, and whether a retransmitted piece of data is treated as a packet or as an agent is sometimes a question of convention. In general, an agent-based approach seems more in place when all nodes treat the agent uniformly, which means that, in particular, they do not apply routing tables specific to their location in the network in order to decide the agent’s next move. Studies of this type of simple token or agent, usually following random walks among the host of the network, have had a major impact on routing and information management protocols in peer-to-peer networks [100]. When a node of a peer-to-peer network needs to discover some information, it may attempt to ask all its neighbours at a time, flooding the network, or release a single request which moves through the network, following a kind of random walk. The performance of latter approach is especially efficient under the assumption of an undirected network with low diameter and good expansion properties, which is usually the case in contemporary peer-to-peer designs for distributed data storage.

Regardless of whether a specific technological problem calls for a “pull” or “push”-type approach to agent mobility, the use of a mobile agent protocol to describe the solution at an abstract level has a number of advantages. Below, we briefly list some of the most important:

- *Economy of resources.* The number of active agents in the system is easy to control, allowing for easy bounding of the amount of communication and the number of messages circulating in the network at any given time. Moreover, nodes which do not host an agent at a given moment of time may remain in a resource-saving “dormant state”.

- *Operation in networks of unknown topology*. The agents operating in the system do not, in general, need to know or store the structure of the entire network. This is an important property, in view of the large scale of the network, and the constant changes in the topology of the network. Quite often, even the exact size of the network is unknown.

- *Operation in dynamic environments*. Agents may be designed so as to success-

fully adapt to a constantly evolving network topology, and to perform local relocation whenever necessary. Such relocation may also at times be necessary if any of the agents is subject to a fault which excludes it from further computations within a team of agents.

- *Independence of implementation details.* Agents are autonomous and may work in heterogenous environments. In many cases, they do not even rely on any information about the host nodes, such as node identifiers. This is the case in the so called *anonymous graph model*, which is the object of study throughout this work.

- *Provable properties of protocols.* Designs based on mobile agents are in many cases easier to analyze, showing provable resilience to network faults, unexpected changes to the network topology, or Byzantine attacks on one or more network nodes.

1.3 Anonymous networks: the theoretical framework

Throughout this manuscript, we will consider problems of mobile agent computing in the following setting. The network in which the agent operates is modeled in the form of a graph $G = (V, E)$. The set of nodes V of the graph describes locations which are capable of hosting a mobile agent, whereas the edges belonging to set E describe communication links, which can be used for transporting the agent and the contents of its local memory from one node to another.

The focus of this study is on network graphs which are *a priori* unknown to the agent, and moreover such that locations in the graph contain no identifying information which would help the agent to tell them apart. This means, in particular, that nodes of the network which have the same number of neighbors are indistinguishable from the perspective of the agents. Such a concept of *anonymous graphs* (or *anonymous networks*) provides the foundations for a model which has found applications in network communication, graph exploration, and stabilization of distributed processes. The basic network model under consideration was introduced by Yamashita and Kameda [172]. It has since been studied in different contexts in distributed computing, particularly in relation to fundamental symmetry-breaking tasks, such as leader election [35, 86, 173]. Various models of system communication have also been adopted, including asynchronous message-passing models [47, 171], scenarios with faults [37], and self-stabilizing approaches [111], as well as agent-based models which are of primary interest to this thesis.

The study of anonymous networks has strong motivation, of both theoretical and practical nature. In principle, due to minimalist assumptions, any solution to a distributed problem that is provided in the anonymous model also constitutes a valid solution in any other model based on a communication graph. The fact that such solution does not depend on node identifiers means that it may be deployed even in environments in which node identifiers may be unknown to the agent (or any other distributed process) operating in the network. Agent-based solutions designed for anonymous networks are likely to have properties of self-healing after changes to network topology, and may be applicable in dynamic environments, as discussed in Section 1.2. Intuitively speaking, an agent-based algorithm which is not allowed to store or distinguish a point of reference in the network, such as the identifier of a previously visited network location, is unlikely to be affected by the future disappearance or corruption of such a point of reference. Some authors also mention scenarios in which node identifiers are simply unavailable, since nodes might potentially refuse to disclose them to agents operating in the system, citing e.g., privacy concerns [T8]. Another important rationale for the study of anonymous graphs is the goal of characterizing the limits of computability for mobile agents in networked scenarios. In the context of agent-based computing, it has to be emphasized that the model does not allow the agent to obtain any information of a topological nature, such as auxiliary routing information or a global sense of direction in the graph.

The considered framework is well-suited for modeling networks of an abstract nature, such as ontologies or webs of interlinked documents, with respect to which the agent is an external process. As such, we will not consider occurrences such as faults of an agent, malicious nodes, or loss of state memory by the agent in between time rounds (obliviousness). In most cases, we will also make the assumption that network nodes have no storage capacities writable by agents, and consequently, that all state information is associated with the agent. Once again, such an assumption is justified when considering logical networks in which a node may represent a static web page or a profile hosted on a remote server. In the absence of this type of helper information, one may either assume that multiple agents can exchange information through a shared (global) block of memory, or consider a more restricted scenario in which communication is only possible among agents occupying the same network node.

Within the anonymous graph model, the way agents operate and the way they collaborate in a team is studied subject to various assumptions, corresponding to different real-world applications. Limitations may be imposed on computational resources available to an agent, in particular, by bounding the amount of state memory carried over edges by an agent). When multiple agents are considered, *clock synchronization* among agents comes into play. We will, for the most part, deal with fully synchronous scenarios, occasionally comparing them to results from the literature concerning the asynchronous setting. Communication between multiple agents will be restricted to the case

when the communicating agents occupy the same node.

For a survey of other models, we refer the interested reader to the recent monograph [130]. In [130], the specific problem of agent rendezvous in the ring topology is used as a "benchmark" for comparing the power of agent-based computations under different scenarios. Models considered therein are often more permissive, allowing the agent to write and read a certain number of bits of information at a node, making use of so-called *white boards*. Alternatively, the agent may be allowed to mark a fixed number of nodes of the network, dropping different types of markers known as *tokens* or *pebbles*. This type of helper information, which we do not allow in the model considered in this work, may prove useful both for orienting a single agent within the network, and for allowing multiple agents to coordinate.

Definition of an anonymous network. We assume that the graph G in which the agent operates is simple, undirected, connected, and anonymous, i.e., the vertices in the graph are neither labeled nor colored. However, while visiting a vertex the robot can distinguish between its adjacent edges. This is achieved by using a predefined local ordering of edges known as a *local orientation*.

The first requirement concerning the local orientation is that it should, at the very least, allow the agent to cyclically iterate through all the edges adjacent to the node. This is realised by providing a unicyclic permutation $\mathit{NextPort}_v$ on the set of edges adjacent to each node v. For an edge e adjacent to v, we will then say that the incident edge $\mathit{NextPort}_v(e)$ is its *right-hand neighbor at* v. The knowledge of the function $\mathit{NextPort}_v$ gives the agent the capability to implement some very simple strategies for walking around the graph. For example, we can conceive an agent which enters node v by edge e and leaves it by edge $\mathit{NextPort}_v(e)$. Such an agent is said to be performing a walk following the *right hand on the wall rule*, also known as the *basic walk*. More advanced strategies are also feasible, since the agent may sometimes choose to exit a node using some other edge, e.g., $\mathit{NextPort}_v(\mathit{NextPort}_v(e))$. In fact, it turns out that careful application of the $\mathit{NextPort}_v$ function is sufficient to define a strategy for the agent which allows it to explore any graph deterministically using only very limited resources regardless of the choice of the *NextPort* function (cf. Section 2.3 on universal exploration sequences and universal tables).

In all further considerations we apply the slightly stronger (though in practice almost equivalent) *labeled port model*. Namely, in addition to a given $\mathit{NextPort}_v$ function, we will assume that for each node there is exactly one distinguished "first" edge leaving this vertex. Equivalently, we allow the agent to rely on an explicit *local port labeling*, in which, for each vertex $v \in V$, there exist consecutive integer labels (starting from 1), called *port numbers*, preassigned to all the edges adjacent to v, next to the endpoint v of each edge. The labels at node v are always distinct and form the discrete interval $[0, \deg(v) - 1]$, where $\deg(v)$ is the degree of v in G, see Fig. 1.1 for an illustration. (A

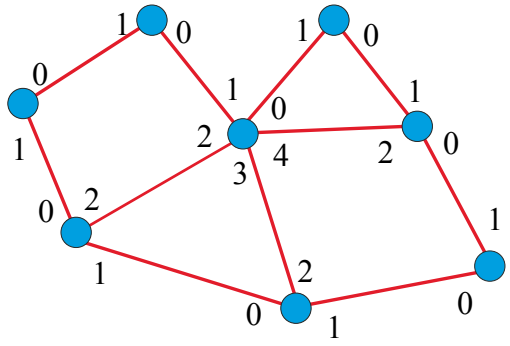

FIGURE 1.1: Example of a port-labeled anonymous graph

discrete interval $[a, b]$ is defined as the set of all integers k such that $a \leq k \leq b$ ($[a, b] = \emptyset$ when $a > b$).) In this way, each edge of the graph is assigned two port numbers, one for each endpoint, and the port numbering is local, i.e., there is no relation between port numbers at u and at v. In such a setting, the *NextPort* function at vertex v can be naturally induced for port l as $\textit{NextPort}_v(l) := [l \bmod \deg(v)]$.

In the sequel, the term "graph" will as a rule refer to a graph equipped with a port labeling described by the above rules, unless otherwise stated. By $succ(v, i)$ we denote the node which is a neighbor of v and linked to it by the edge with port number i at v, and by $end(v, i)$ we denote the port number of the same edge at $succ(v, i)$. For each edge $\{v, w\}$ traversed by using port i at node v ($w = succ(v, i)$), the pair of ports $(i, end(v, i))$ is called the *edge label* of $\{v, w\}$ and denoted by $lab(i, end(v, i))$. We use the natural lexicographic ordering of edge labels: $(i_1, i_2) \prec (j_1, j_2)$, if $i_1 < j_1$, or $i_1 = j_1$ and $i_2 < j_2$. We write $(i_1, i_2) \preceq (j_1, j_2)$, if $(i_1, i_2) \prec (j_1, j_2)$ or $(i_1, i_2) = (j_1, j_2)$.

The graph in which the agent operates is denoted by $G = (V, E)$, with $|V| = n$ and $|E| = m$. The diameter of the graph is denoted by D, its maximum vertex degree by Δ, and its minimum vertex degree by δ. Following conventions from the literature, the degree of a regular graph or an upper bound on the maximum degree of an arbitrary graph which is known to the agent will sometimes be denoted by d. The set of neighbors of a vertex $v \in V$ is denoted by $N(v)$, with $\deg(v) = |N(v)|$. For any $v \in V$, let $E_G(v)$ denote the set of edges of G that are incident to node v.

Definition of a mobile agent. When an agent moves from one node to another, it carries with its own local memory which encodes the state of the agent, based on the information obtained by the agent in previous moves. Note that when the agent is located at any node of the graph, it has access to a read-write memory which can be used for local computation, but we are usually not concerned about the cost of performing local computations at node. Moreover, the agent may never leave behind any marks on the nodes it has visited.

We consider mobile agents traveling among the nodes of an anonymous graph of

unknown topology. Formally, a *mobile agent* is an abstract state machine in Mealy form: $\mathcal{A} = (S, s_0, \pi, \lambda)$, where S is a set of states among which there is a specified state s_0 called the *initial* state, $\pi : S \times \mathbb{Z}^2 \to S$ is the state transition function, and $\lambda : S \times \mathbb{Z}^2 \to \mathbb{Z}$ is the output function. Here, $\mathbb{Z}$ denotes the set of all integers.

Initially, the agent is located at some node $u_0 \in V$ (unknown to the agent), called its *initial position* or *homebase*, in the initial state $s_0 \in S$. The agent performs actions in discrete time rounds, measured by its internal clock. Each action can either be a move to an adjacent node or a null move resulting in the agent remaining in the currently occupied node. Suppose that in time round t, the agent entered a node u by port p in memory state s; we set $p = -1$ if the move performed in round t was a null move, or if $t = 0$. Then, in round $t+1$ the agent will leave node u by some port p' in some memory state s', given by the output function λ and state transition function π, respectively, depending on the port p by which the agent entered u and the degree of u:

$$p' = \lambda(s, p, \deg(u)),$$
$$s' = \pi(s, p, \deg(u)).$$

A value of $p' \in [0, \deg(u) - 1]$ signifies that in round $t + 1$ the agent will move from node u along port p' to node $succ(u, p')$; any other value of p' corresponds to a null move of the agent. The agent continues moving in this way in successive time rounds, possibly forever.

The above definition admits a natural extension to the case of randomized agents, which are described simply as randomized state automata. The output pair (p', s') is no longer described by a deterministic pair of functions (λ, π), but by a pair of not necessarily independent random variable-valued functions $(\boldsymbol{\lambda}, \boldsymbol{\pi})$.

Problems, solutions, and complexity. The notion of a problem and a solution in the context of mobile agent computing is rather informal. When talking about the question of *computability* with a single mobile agent (subject to given constraints on resources), we will apply terms derived from complexity theory. We consider languages which are families of graphs with a single distinguished vertex. A *problem* L will then be understood as a binary question of the form "Does the pair (G, v) belong to the language L?". We will say that this problem can be *solved* by a mobile agent under the given constraints if there exists an agent A satisfying the said constraints such that agent A, when released from node v in graph G, terminates after a finite number of steps in an accepting state if and only if (G, v) belongs to L, and terminates after a finite number of steps in a rejecting state, otherwise.

We will sometimes say that we allow the agent solving problem L to have knowledge of some global parameter $f(G)$ of the graph. Formally, this means that instead of asking about the existence of a single agent A which solves L, we will ask if there exists a

family of agents $\{A_f\}$, such that agent $A_{f(G)}$ deployed in graph G correctly decides problem L for the pair (G, v), $v \in V$.

In some cases, we will think of problems in a generalized sense, closer in spirit to distributed computing. For each valid starting configuration with an agent or set of agents placed in graph G, we will look at the countable sequence of global configurations (snapshots) of the system composed of the graph and the agents located in it, achieved in successive time rounds $t = 0, 1, 2, ...$ A *(generalized) problem* is defined as a set of testable conditions on a sequence of snapshots, and is said to be *solved* by the agent if the sequence produced by the agent meets the constraints of the problem. For example, in the *exploration problem* we will require that all the nodes of the graph are visited by the agent at some point in time, while in the *rendezvous problem* for two agents in the graph — that at some moment in time the two agents are located at the same node. The smallest T such that for all $T' \geq T$ the prefix of the sequence of states of the system in time rounds $t = 0, 1, \ldots, T'$ satisfies the constraints of the problem is referred to as the time required to solve the considered problem, e.g., the exploration time for the exploration problem. For the case of randomized agents, we again extend this notion by taking a probability distribution over all possible execution paths of the algorithm, and describing the expectation of the time until the problem is solved. We will also sometimes impose the requirement of termination of an agent-based algorithm, which means that there must exist a moment of time from which the configuration of the system remains unchanged.

A subtler point concerns the definition of the memory size of the agent. Unless otherwise stated, we will assume that the memory of the agent is defined as the logarithm of the number of its states, $M = \log_2 S$. For this definition, it is important to note that the agent is given as a Mealy machine. If agents were instead to be given as Moore machines, and their memory defined likewise through the logarithm of the number of states, then the memory complexity of some problems might increase by up to $O(\log \Delta)$ if the agent makes use of its input port and the degree information of the current node, which it would have to encode as part of its state. The distinction does not really matter for problems with a memory complexity of $\Omega(\log n)$, hence in most cases the models can be seen as equivalent. In the sequel, whenever we consider computations in sub-logarithmic space, we will be adhering strictly to the Mealy machine model. For example, the *basic walk* strategy in which the agent enters a node by port p and exits it using the right hand rule by port $(p + 1) \mod \deg(v)$, will be considered herein to be memoryless (stateless).

The agent's memory size can also be understood and defined in a slightly different manner. One may perceive the agent not as an automaton, but as a computer program which sends itself (i.e., its machine code and its state) from node to node, lodges in the memory of its new host, and performs computations to determine which host to visit next. In this view, the space consumed by the agent is measured as the maximum number

of bits of a host's memory that the agent ever occupies, throughout execution. Since this measure is relevant from a practical perspective and also has profound theoretical implications, we will occasionally make references to it, using the term *memory of local computation* of the agent.

1.4 Overview of contributions

The main research questions undertaken in this study concern the feasibility of solving fundamental tasks in an anonymous network, subject to limitations on the resources available to the agent. Typical challenges include: exploring a graph by means of an agent with limited memory, discovery of the network topology, attempting to meet with another agent in another network (rendezvous), or dealing with changes to the network topology in time. The constraints imposed on the agent may include the number of moves which the agent is allowed to perform in the network, the amount of state memory available to the agent, the ability of the agent to communicate with other agents, as well as its *a priori* knowledge of the network topology or of global parameters.

The material covered in this manuscript includes most of the contributions of papers [T1–T8], and is organized as follows. This introductory chapter was devoted to a general discussion of mobile agents and to a formal introduction to the anonymous network model. In Chapter 2, we consider the problem of graph exploration, in which a mobile agent is charged with the task of visiting all of the nodes of a the network it is operating in. Since we are working in the anonymous network model, agents cannot immediately recognize previously visited nodes, which renders many search methods known from the centralized model, such as Breadth First Search (BFS) or Depth First Search (DFS), infeasible. In anonymous networks, the basic randomized strategy which is applicable is the random walk, in which the mobile agent moves from the current node to a neighbour picked with uniform probability. The random walk serves as a point of reference for other exploration strategies, both randomized and deterministic, in terms of parameters such as: the time required to visit all nodes of the graph, the memory required by the agent, the regularity of visits to nodes or edges during continued exploration, resilience to faults, performance for multiple agents, etc. It is in such a context that the results on several different exploration strategies, obtained in the papers [T1–T5] are presented.

In [T1], we consider the Metropolis walk, which is a form of biased random walk on graphs, and following [147] we assume that transition probabilities of the Metropolis walk between a pair of neighbouring nodes depend only on their degrees. Our main results in [T1] concern the properties of short Metropolis walks, and stem from the observation that an agent following a Metropolis walk for a certain number of t steps discovers $\Omega(\sqrt{t})$ distinct nodes of the network in expectation, for $d^2 \leq t \leq n^2$, in a graph of maximum degree d and at most n nodes. In [T1], these results are used

to design graph exploration strategies in the centralized RAM model of computation, achieving a tradeoff between execution time and required memory space which is superior to that of previously known approaches, such as DFS, BFS, or the unbiased random walk. In Chapter 2, the results from [T1] are translated to the distributed setting, into the considered anonymous network model. We also point out that the Metropolis walk can be implemented in the anonymous network model to explore all graphs in an expected number of $O(n^2 \log n)$ steps, using an agent equipped with only $O(\log \log n)$ bits of memory. Next, for the sake of completeness, in Chapter 2 we present the state-of-the-art in deterministic and randomized exploration strategies, and corresponding lower bounds. We address the question of how to design deterministic exploration strategies based on the random walks and Metropolis walks. We recall concepts of Universal Traversal Sequences (UTS) and Universal Exploration Sequences (UXS), and describe a generalization of such sequences for non-regular graphs, formalizing the notion of a universal exploration table.

The results presented in [T2] address the question of exploration of anonymous networks in the case when the designer of the network has no control of its topology, but may adjust the ordering of neighbors at each node (the so-called port labelings) to facilitate agent-based exploration. With this assumption, the problem of network exploration can be solved by an agent which deterministically traverses a closed trajectory. Building on a series of results in the area, in [T2] we show that the port labeling can be chosen so that even a memoryless agent, which follows a very simple strategy known as the basic walk, can traverse in any graph of n nodes a closed trajectory of length less than $4n$. This type of result highlights the fact that the difficulty of the exploration problem in anonymous networks lies in the inability of the agent to figure out a sense of direction in the graph when the port labeling is not set specifically to help it in the task. The results presented in [T2] rely on a proof of the existence of a specific kind of tree-based substructure in any graph, which may be of interest in its own right.

Chapter 2 closes with a discussion of variants of the anonymous model in which the agent is memoryless, but is guided by auxiliary counters on the edges of the graph. These considerations cover the material of papers [T3–T5]. We start by presenting the so-called rotor-router model, in which each node maintains a cyclic list of its neighbours and the agent, during successive visits to a node, leaves it by proceeding to neighbours chosen from the list in a round-robin fashion. The results obtained in [T3] include an almost-complete characterization of worst-case exploration time for an agent in the rotor-router system, subject to different models of the adversarial setting (e.g., the adversary may be able to preselect the ordering of the neighbours for each node, or the first neighbour to be visited from a given node). The main result of that paper is a tight lower bound on the number of steps required by the agent to explore the graph. In combination with the result of Yanovski [175], our result from [T3] implies that for any graph of m edges with diameter D, $\Theta(m\,D)$ steps are always sufficient and sometimes required

for an agent following the rotor-router to visit all nodes of the graph, and subsequently to stabilize to a periodic Eulerian traversal of its set of edges. In [T4], we study the behaviour of the rotor-router after changes or faults occur in the system. We show that it is possible to relate the behavior of the rotor router to the BEST theorem (due to de Bruijn, van Aardenne-Ehrenfest, Smith and Tutte), providing a product formula for the number of Eulerian circuits in an orientation of the undirected network graph. This type of analysis allows us to show that, after a single edge addition or port modification, the rotor-router stabilizes to its new limit behaviour within $O(m)$ steps, however, a single edge deletion may result in the need for the process to start stabilizing from scratch, requiring $\Theta(m\,D)$ steps. Finally, the paper [T5] introduces a slightly different exploration model, in which the agent always follows the least often traversed edge adjacent to its current node (breaking ties arbitrarily). We provide a complete analysis of the performance of such a strategy, known as Least-Used-First, showing that it explores the graph in $\Theta(m\,D)$ steps, just as the rotor-router. Interestingly, a related rule in which the agent traverses the edge which has not been visited for the longest time turns out to work in an exponential number of steps in the worst case, even though it might at first glance appear to be only a slight modification of the round-robin rule of the rotor-router. To put all the considered methods into perspective, we provide an overview of the main results discussed in Chapter 2, in the form of Tables 2.1 and 2.2.

Graph exploration can in some sense be seen as the most fundamental practical global problem which can be solved by a single agent in an unknown network. If the agent does not know how to explore the graph efficiently, then most likely it will not be able to e.g. compute values of topology-dependent parameters, solve the problem of locating an item placed on one of the nodes ("treasure hunt"), or to meet with another agent (rendezvous). At the other end of the difficulty spectrum lies the question of deciding a complete or universal problem for the agent, i.e., one to which a whole class of other problems can be reduced. This definition of completeness is somewhat informal and depends on what type of problems are considered. If the goal of the agent is to try to compute some parameter which is a function of the network graph it is operating in, then the basic complete problem consists in collecting all topological information stored in the network which is possible to retrieve. This task, known as map construction, is considered in Chapter 3. We discuss different possible representations of such a map of all retrievable information about the network, recalling the concepts of an agent's view and a quotient graph. The latter representation proves to be efficiently computable by an agent which knows some upper bound n on the number of nodes of the network. We present some of the results from [T6], which imply that an agent can construct the quotient graph of the network in a number of steps which is polynomial in n. Consequently, if any parameter of the network can be computed by an agent, it can be computed in polynomial time. We also present a method introduced in [T7], which can be used by an agent to decide the existence of a specific edge of the quotient graph not only in poly-

nomial time, but also in logarithmic space. Properties which can in this way be tested by an agent with knowledge of an upper bound on the number of nodes and logarithmic memory include, for example, testing if the network is a tree, and if not, finding an edge whose removal does not disconnect the network. All of the discussed methods for quotient graph construction rely in different ways on applications of universal exploration sequences, traversal sequences, or exploration tables, with the log-space approach from [T7] also introducing the new notion of a distinguishing sequence — a sequence whose traversal allows an agent to tell two graphs apart, whenever such a distinction is possible. A comparison of the time and memory requirements of the studied methods is given in Table 3.1.

In Chapter 4, we extend our considerations to problems with more than one agent, focusing on the rendezvous problem, which asks about the feasibility of achieving a situation in which two agents meet. We give most of our attention to the scenario studied in [T7, T8], in which the two agents deployed in the network are completely identical, moving in synchronous time rounds at the same speed, and having exactly the same starting state and no unique identifiers. We recall the characterization of starting situations for which rendezvous in the studied scenario is feasible. We then present the main result of [T7], which states that for all feasible starting situations, there exists an algorithm which achieves rendezvous, using agents with only logarithmic memory. The applied method relies on an extension of the previously discussed approach for map construction. A complementary lower bound from [T7] states that $\Omega(\log n)$ bits of memory are required for rendezvous even when the graph in which the agents operate is a ring. Finally, we present the results of [T8], which characterize the interplay between the time and the space required to achieve rendezvous for the special case when the network is a tree. We prove that for an agent with $k \geq c \log n$ bits of memory, where c is an absolute constant, $\Theta(n^2/k + n)$ steps are always sufficient and occasionally required to achieve rendezvous. This complexity result for rendezvous in trees is a significant improvement with respect previous approaches known from the literature, and is also an isolated example of an important problem for which a tight time space-tradeoff is known for the whole spectrum of memory sizes.

In Chapter 5 we discuss some of the perspectives of the field of mobile agent computing in networked scenarios. In particular, we present a broader perspective of the challenges facing collaborative computations with multiple agents, and briefly outline some of our most recent research results.

2 The exploration problem

In the task of *network exploration*, a mobile agent, initially located at some node of the network, is required to visit all the nodes of the network. If we allow the network to be disconnected, then the agent is expected to visit all the nodes of its connected component of the network. Exploration routines lie at the heart of tasks related to the search of information in the network by an agent, as well as of tasks related to finding or detecting the presence of another agent in the network (see Chapter 4). The agent may also be required to explore a graph periodically, e.g. with the goal of regular patrolling and monitoring of network resources. For a historical overview of questions and results related to network exploration, we refer the reader to [45, 98].

In centralized computing, one of the simplest strategies for exploring a graph is that of Depth First Search (DFS). When performing DFS, the agent traverses a spanning tree of the graph rooted at its starting location, moving at each step to an unvisited neighbouring node in the graph whenever such a node exists, and otherwise backtracking to its parent node in the tree. This guarantees a traversal of all of the n nodes of the graph in optimal $O(n)$ time (a complete traversal of the DFS tree takes $2(n-1)$ time rounds). Unfortunately, DFS-type approaches do not fit into the considered framework of mobile agents in anonymous networks. First of all, DFS relies on the ability to recognize nodes previously visited by the agent, which is not feasible in the anonymous graph model in the absence of node identifiers or whiteboards writable by the agent. Even in models where node identifiers are available, search strategies based on Depth First Search or Breadth First Search often prove insufficient. To run such an algorithm, the process has to maintain a memory stack or queue in its state memory, encoding the identifiers (or at least port labels) along the path in the tree from its root to the current location of the agent, which potentially requires $\Omega(n)$ space, and causes problems if the topology of the graph changes during the execution of the algorithm. Hence, the need arises to design alternative exploration strategies. Most such strategies are inspired, at least to some extent, by considerations of the random walk.

A comparison of the algorithms and approaches considered in this chapter is provided in Table 2.1, with the criteria of comparison being: the determinism of the approach, the exploration time, and the required memory. We will also consider other parameters of exploration strategies, and we start this chapter by introducing a number of additional characteristics useful for comparing exploration strategies in anonymous graphs in Section 2.1. In Section 2.2, we look at randomized graph exploration strategies for anonymous graphs, considering the simple random walk and its biased variants, in par-

ticular, the Metropolis-type walk. The rest of the chapter is devoted to an overview of deterministic exploration strategies for anonymous graphs. We recall known results for deterministic exploration strategies, obtained by derandomizing (biased) random walks in Section 2.3. We then discuss variants of the anonymous graph model, designed with the goal of obtaining faster deterministic exploration algorithms. In Section 2.4 we address the question to what extent one can assist a deterministic exploration strategy by appropriately configuring port numbers in the anonymous network. Finally, in Section 2.5 we consider strategies in which the agent is guided by counters on ports (corresponding to the so-called *rotor-router model*), and present analogous results for models in which the agent is guided by counters on edges.

2.1 Comparing exploration strategies

The agent-based algorithms we consider herein are meant to be light, memory-efficient, and resilient to faults, and to operate in networks without visible node identifiers. To this end, we consider several parameters measuring different aspects of the performance of exploration strategies, the most important of which are discussed below.

Cover time. The *cover time* (or *exploration time*) $C(G)$ of an exploration strategy is defined as the maximum expected length of the time interval during which the mobile agent following the strategy visits all the nodes of the graph, where the maximum is taken over all starting configurations of the system (in particular, over all starting nodes for the agent). When considering algorithms different from the random walk, i.e., when the agent is equipped with state memory, we define $C_t(G)$ as the expected length of the shortest time interval starting at time round t during which all nodes of G are visited at least once. We then distinguish the *first cover time* $C_0(G)$ describing the cover time of the system just after initialization, and the *refresh time* $C_\infty(G) = \limsup_{t\to+\infty} C_t(G)$, describing the eventual cover time of the system in the limit. The latter measure is interesting from the point of view of scenarios with patrolling, in which we expect the agent to periodically visit all nodes of the graph.

Regularity of exploration. From the perspective of the designer of an exploration strategy, it is desirable to ensure that all of the vertices and/or edges of the graph are visited regularly. Let $\mathbf{c}_{s,v}(t)$ be the random variable describing the number of visits to a vertex v within some number of steps t, for an exploration starting at vertex s. The *visit frequency* $f_v(G)$ of vertex v is defined as the frequency of visits to this vertex, given a worst-case starting vertex: $f_v = \min_{s\in V} \mathbb{E} \liminf_{t\to+\infty} \frac{\mathbf{c}_{s,v}(t)}{t}$. An analogous measure, known as the *traversal frequency* $f_e(G)$, is defined for visits to edges. We will say that a strategy visits all nodes (edges) *fairly* or *equitably*, if its traversal frequency is roughly

the same for all nodes (edges), i.e. $f_v(G) \sim 1/n$ ($f_e(G) \sim 1/m$).

Discovery rate. The discovery rate of a strategy gives a lower bound on the number of newly explored nodes or edges in the first t steps of exploration, for small t. A strategy with a high discovery rate is preferable when performing only partial exploration tasks, for instance, when the agent is searching for a resource which is available at some uniformly randomly spread subset of nodes of the network. Informally speaking, for strategies with a high discovery rate we can also hope for smaller cover time when deploying multiple agents, which collaboratively complete the exploration process.

Required resources. Last but not least, the chosen exploration strategy may impose requirements on the capabilities of the agent. Some of the strategies considered in this chapter require the agent to be equipped with a certain amount of state memory, and work differently depending on whether the agent has knowledge of some global parameters, such as a bound on number of nodes of the network. Finally, some of the exploration strategies considered here are randomized, while others are deterministic. Whereas the question of determinism vs. randomization can be regarded in terms of guarantees on performance, the desire to eliminate randomness and to construct deterministic strategies is also motivated by practical constraints. True randomness may prove to be a scarce resource, unavailable to the agent.

2.2 Randomized strategies

2.2.1 A point of reference: the random walk

Classically, a random walk is defined as a Markovian process, starting with an initial probability distribution over the set of nodes of a graph, and with transition probabilities governed at each step by the normalized adjacency matrix of the graph (cf. e.g. [6]). For the purposes of studies of mobile agents moving around the graph, we will choose to use an equivalent frequentist view of probability. In this context, the *simple (unbiased) random walk* is an oblivious exploration strategy for a mobile agent located at a node of the graph. The edge used by the agent to exit its current location is chosen with uniform probability from among all the edges adjacent to the current node. In other works, for a walk located at a node u, outport i is chosen with probability $p_i = \frac{1}{\deg(u)}$.

The random walk is arguably the simplest possible strategy which, in expectation, allows even a memoryless randomized agent to explore any connected graph (in expectation). Moreover, it admits the following desirable properties:

- In expectation, the random walk visits all of the vertices of the graph within polynomial time.

TABLE 2.1: Comparison of exploration algorithms in variants of the anonymous network model.

Agent's algorithm	Agent's memory	Cover time (first cover)	Refresh time (limit behaviour)	Reference
Anonymous graph model				
Random walk *	none	$O(\min\{mn, mD\log n\})$ *	same as cover time	[7]
Metropolis walk *	$O(\log\log n)$	$O(n^2 \log n)$ *	same as cover time	[7, 147, T1]
Deterministic algorithm	$O(\log n)$	$O(n^4 \log^2 n)$	same as cover time	[7, 69, T1]
— lower bound on memory	$\Omega(\log n)$	–	–	[93, 159]
— lower bound on time	–	$\Omega(n^2)$	same as cover time	[39]
Anonymous graph model with ports pre-configured for faster exploration				
A dedicated rule	$O(1)$	$< 3.5n$	same as cover time	[56]
Right-hand rule	none	$< 4n$	same as cover time	[T2]
Exploration controlled by the environment				
The rotor-router rule	none **	$\Theta(mD)$	$O(m)$	[175, T3, T4]
The least-used-first rule	none **	$O(mD)$	$O(m)$	[T5]

* - for randomized algorithms, the provided values of cover time and refresh time represent expectations

** - in models controlled by the environment, there exist auxiliary counters associated with edges of the network

- In expectation, the random walk stabilizes to the steady state within polynomial time, and henceforth all edges are visited with the same frequency.

In these respects, the random walk will be treated as a point of reference for deterministic exploration models, for which we will attempt to achieve similar properties, but in the sense of worst-case performance.

For the interested reader, we provide below a brief overview of the most important properties of the random walk, starting with bounds on the time necessary to explore an anonymous network.

Cover time. Different techniques are used to bound the cover time of random walks in general graphs. Some of the most useful bounds either rely on the relation between cover time and the maximum time required to reach a fixed vertex of the graph (Matthew's bound [143]), or make use of the resistor network analogy, linking the so-called commute times of the random walk with resistances of replacement between pairs of nodes of the resistor network corresponding to the considered graph [48]. A compendium of simple bounds which can be obtained using such techniques is given by the theorem below.

Theorem 2.1. [7, 48, 143] *For the simple random walk process in any graph G, the following bounds on cover time hold:*

- $C(G) < 2m\,D(\ln n + 1)$.
- $C(G) < 2mn$.
- $C(G) < \frac{4}{27}n^3$.
- $C(G) > n \log n$.

All of these bounds are asymptotically tight up to a constant factor in some graph classes. For the first bound, a value of $C(G) \in \Theta(m\,D \log n)$ is achieved in many graphs with small values of diameter, for instance, for the class of stars and for the class of complete binary trees. The bound $C(G) \in \Theta(mn)$ is tight for the class of *lollipop graphs*, which consist of the union of a path and a clique, with one of the vertices of the clique connected by an edge to an endpoint of the path. For a given value of n, an asymptotically largest possible cover time of $C(G) = \frac{4}{27}n^3 - o(n^3)$ is achieved by a lollipop graph which has roughly $n/3$ nodes in the path and roughly $2n/3$ nodes in the clique (cf. [41] for the details of the construction).

For many special network topologies, such as complete graphs, expanders, trees, or grids, more precise bounds on the cover time can be obtained (cf. [6]). One property worth noting is a small cover time of the random walk of $O(n \log n)$ on graphs with good expansion properties, and of $O(n^2)$ on regular graphs, regardless of their degree.

Fairness of traversal. The random walk in the limit visits all edges of the graph equally often, $f_e(G) = 1/m$. In fact, the random walk admits an even stronger property. In any connected undirected non-bipartite graph G, the random variable describing the location of the agent converges to a stationary probability distribution, so that in any step the probability π_v that the walk is located in a given node v is proportional to the degree of the node, i.e.:

$$\pi_v = \frac{\deg(v)}{2m}.$$

For bipartite graphs, the same stationary distribution can be reached by changing the transition probability so that the walk has some small probability of remaining at its current vertex at each step. More precisely, at a vertex u, the probability of choosing port i is $p_i = \frac{1}{\deg(u)+1}$, and the probability of remaining at vertex u is $p_- = \frac{1}{\deg(u)+1}$. Such a walk, known as the *lazy random walk*, has the same asymptotic values of parameters such as cover time.

In expectation, the random walk stabilizes to a fair traversal of the edges very quickly. Several notions for describing the rate of convergence of the random walk to its stationary distribution have been introduced. One of the most studied is that of *blanket time*, which (informally speaking) corresponds to the expected moment at which (for a regular graph) all vertices have been visited a similar number of times, cf. [170]. A very recent result of Ding, Lee, and Peres [73] is that the blanket time is within a constant factor of the cover time, for all graphs.

Irregular behaviour. The fact that the random walk visits the edges of the graph fairly does not directly imply a worst-case bound on the length of the time interval between two successive visits by the agent to a fixed edge e of the graph. Such an interval is, in fact, potentially unbounded. For the random walk, which is a memoryless strategy, the length of this interval may be described by a single random variable $\mathbb{T}_e$. The expected value of $\mathbb{T}_e$ is the same for all edges and is given by:

$$\mathbb{E}\mathbb{T}_e = \frac{1}{f_e} = m.$$

The higher order moments of $\mathbb{T}_e$ depend on the chosen edge and the topology of the graph. In general, the probability tail of $\mathbb{T}_e$ decays exponentially only for time intervals exceeding the edge cover time $C_e(G)$ of the graph. For any even integer k, by applying the Markov bound to a walk on a time interval of length $2C_e(G)$, and considering $k/2$ independent iterations of such a walk, we can bound the tail of distribution $\mathbb{T}_e$ as follows:

$$\Pr[\mathbb{T}_e \geq kC_e(G)] \leq 2^{-k/2}.$$

The irregularity of exploration may still be high for graphs with a large cover time. For example, in the so-called lollipop graph which consists of a path of length $n/3$ connected

at one endpoint to a clique on $2n/3$ nodes, the distribution of visits to the edge e at the end of the path which is not connected to the clique, has a heavy tail up to the order of the cover time of the graph, $\Pr[\mathbb{T}_e \geq \frac{4}{27}n^3] \in \Omega(\frac{1}{n})$. This means, for example that, with constant probability, any time interval of n^4 consecutive rounds includes a sub-interval of $\Omega(n^3)$ rounds in which the considered edge is unvisited. Alleviating this type of weakness of the random walk is one of the goals in the design of alternative randomized and deterministic strategies for graph exploration.

Discovery rate and parallelization. The discovery rate for the random walk is far from simple to analyze. Linial (cf. [113]) conjectured that a random walk of length t visits $\Omega(\sqrt[3]{t})$ nodes in expectation, and this conjecture was later proved by Barnes and Feige [26]. They also showed that a walk of the same length visits $\Omega(\sqrt{t})$ edges in expectation, and more strongly, that the expected value of the product of the distinct number of edges and vertices visited until time t is $\Omega(t)$. All of these results hold up to m visited edges and n visited nodes, and the latter result can be seen as a powerful generalization on the $O(mn)$ bound on the cover time of the random walk.

The so-called *parallel random walk* is achieved by deploying independent agents performing random walks in a graph independently and without any form of coordination. Recent work on the area of parallel random walks [9, 80, 83, 160] contains a characterization of the improvement of the cover time due to the deployment of k independent random walkers with respect to the case with a single walker. It is shown in these works that the achieved speed-up depends on different parameters, such as the mixing time [83] and edge expansion [160] of the graph. The speed-up may sometimes be as low as $\Theta(\log k)$ [9], and sometimes as high as exponential in terms of k [80]. For many classes of graphs the speed-up is linear in terms of k (especially when k is small, $k \in O(\log n)$).

2.2.2 Fast exploration using the Metropolis walk

It turns out that, with respect to the random walk, further improvement of the cover time of the walk is possible by applying so-called "look-ahead" strategies, allowing the agent to obtain some additional information, e.g., about the topology of the neighborhood of the current vertex of the walk. The main difficulty lies in attempting to simulate this type of additional knowledge in a graph exploration strategy without changing the assumptions of the anonymous graph model, in which the mobile agent only has access to information about the current node.

A *biased random walk* on a graph G governed by a *transition matrix* $P : V \times V \rightarrow [0, 1]$ over graph G is defined as a walk in which the probability that an agent located at a node v transits in the next step to a node u is equal to $P(v, u)$. It is assumed in the definition that matrix P is stochastic, i.e. for all v, $\sum_{u \in V} P(v, u) = 1$, and that it reflects

the topology of G in terms of permissable moves of the agent, i.e., $P(v,u) = 0$ for all v, u such that $v \neq u$ and $\{v,u\} \notin E$. The matrix P does not have to be symmetric, allowing the same edge to be chosen with different probability in different directions. The parameters of the biased random walk are denoted here in the same way as for the unbiased walk, but using the matrix P in place of the graph G.

Modifications of the transition matrix decrease the probability of using some edges of the walk, favoring others. For example, one may consider a transition matrix P_T which restricts the agent to a random walk on some fixed spanning tree T of graph G, i.e., putting $P_T(v,u) = 1/\deg_T(v)$ for $\{v,u\} \in E(T)$, and $P_T(v,u) = 0$ for all other edges. Such a transition matrix guarantees a vertex cover time of $C(P_T) = O(n^2)$, regardless of the chosen spanning tree T. Such a cover time is in some sense the best possible for a biased walk: it was shown in [108] that when graph G is a path, any transition matrix P over G has cover time $C(P) = \Omega(n^2)$. However, achieving $O(n^2)$ cover time by means of a transition matrix based on a spanning tree requires global knowledge of the topology of the graph, and is completely infeasible in the anonymous graph model. As pointed out in [108], it is not known if a transition matrix with $O(n^2)$ cover time can be constructed by the agent based on local information, only.

In the current state-of-the-art [108, 147] there do, however, exist two schemes with a cover time of $O(n^2 \log n)$ in which the elements $P(v,u)$ of the transition matrix depend only on the degrees of vertices from the neighborhood of v. The first of these schemes, proposed in [108], relies on a transition matrix in which the walk chooses the next node in the neighborhood of v proportionally to the inverse of the square root of its degree. Again, implementing transitions according to such a transition matrix using a constant number of time steps of the mobile agent seems difficult to achieve. This problem can, however, be alleviated for the latter scheme, proposed in [147], which relies on so-called *Metropolis walks*.

The Metropolis-Hastings-type transition matrix can be defined so as to achieve any given stationarity probability distribution π_v of the walk on the nodes of the graph. Intuitively, the unbiased random walk has a stationary distribution of $\pi_v = \frac{deg(v)}{2m}$, whereas the cover time of the walk can be improved by balancing the stationary distribution towards the uniform distribution $\pi_v = 1/n$.

Theorem 2.2 ([147])**.** *For any normalized probability distribution $\boldsymbol{\pi} = \{\pi_v : v \in V\}$ with $\min \boldsymbol{\pi} > 0$, let $P_{\boldsymbol{\pi}}$ be the transition matrix on graph G given by:*

$$P_{\boldsymbol{\pi}}(v,u) = \begin{cases} \min\left\{\frac{1}{\deg(v)}, \frac{1}{\deg(u)}\frac{\pi_u}{\pi_v}\right\}, & \text{for } \{v,u\} \in E \\ 1 - \sum_{w \in N(v)} P_{\boldsymbol{\pi}}(v,w), & \text{for } v = u \\ 0 & \text{otherwise.} \end{cases}$$

Then, $C(P_{\boldsymbol{\pi}}) = O(f n^2 \log n)$, where $f = \max\{\pi_u/\pi_v : u, v \in V\}$.

We remark on two special cases of the above strategy.

When the probability distribution is given by $\pi_v = \deg(v)/2m$ for all $v \in V$, we obtain $P_{\boldsymbol{\pi}}(v,u) = \frac{1}{\deg(v)}$ for $\{v,u\} \in E$, and $P_{\boldsymbol{\pi}}(v,u) = 0$ otherwise. Thus, the transition matrix describes the unbiased random walk, and the claim of the theorem implies that $C(G) = O(\frac{\Delta}{\delta} n^2 \log n)$.

The second interesting case is given by the uniform probability distribution $\pi_v = 1/n$ for all $v \in V$, for which $f = 1$, and the claim of the theorem yields a bound of $O(n^2 \log n)$ on the cover time.

There exists a simple procedure, due to Metropolis *et al.* [144], for simulating a single step of Metropolis-Hastings walk by sampling a neighbour of the current location uniformly at random, and then deciding whether to accept the new node or remain at the current one. We observe in [T1] that this implies that the Metropolis walk can be used as an efficient strategy for randomized graph exploration, using only logarithmic space.

In the setting of mobile agents in anonymous graph, this procedure is formally stated as Algorithm 1 (assuming for simplicity a uniform distribution on nodes, $\pi_v = 1/n$). In this formulation, the agent is assumed to have the set of states $S = [0, \Delta]$. The special state 0 represents an agent which is looking for a new node to move to. States from the range $s \in [1, \Delta]$ signify that the agent is testing the current node as a potential new location, having arrived from a node of degree s. With probability depending on s and the degree of the new node, it may accepting it, or potentially revert to the original node. (This type of formulation appears to have been first given in 2012, for the JAG model of graph exploration in [T1], and at the same time for the crawl-and-jump model of network sampling in [136]).

Algorithm 1: Metropolis walk strategy for fast randomized exploration with a log-space agent [136, T1]

- When the agent is located at a node $v \in V$ of degree d in state $s = 0$:
  The agent chooses a port $i \in \{0, \ldots, d-1\}$ uniformly at random and moves to the vertex $u = end(v,i)$, changing its state to $s := \deg(v)$. {"Test"}
- When the agent is located at a node $u \in V$ in state $s > 0$:
  With probability $p = \min\{\frac{s}{\deg(u)}, 1\}$ the agent remains at u, performing a null move and changing its state to $s := 0$. {"Accept"}
  Otherwise, the agent reverses the last move, moving along the port by which it entered u, and also changing its state to $s := 0$. {"Revert"}

Since the memory of the agent required in Algorithm 1 can be bounded as $\log |S| \leq \log n$, we obtain the following Corollary of Theorem 2.2 with $f = 1$.

Corollary 2.3 ([T1])**.** *There exists a randomized strategy for a mobile agent which explores any graph w.h.p. in $O(n^2 \log n)$ rounds, requiring $O(\log n)$ state memory.*

In fact, the memory requirement for the agent can be reduced by implementing a Metropolis-type walk which takes into account only an approximation of the value of the degree of the node, rounded up to the nearest power of 2. This can otherwise be seen as adding a certain number of self-loops to each node of the graph, so that its degree becomes a power of 2. The corresponding modification of Algorithm 1 consists in setting s' as $s := \lceil \log \deg(v) \rceil$ in the "test" step, and then accepting the new state with probability $p = \min\{\frac{2^s}{2^{\lceil \log \deg(u) \rceil}}, 1\}$. By performing this modification, we obtain a stationary distribution of the walk which only differs from the uniform distribution by a constant factor, i.e., we can apply Theorem 2.2, knowing that $f < 2$.

Corollary 2.4. *There exists a randomized strategy for a mobile agent which explores any graph w.h.p. in $O(n^2 \log n)$ rounds, requiring $O(\log \log n)$ state memory.*

We close the discussion of the cover time of randomized exploration strategies with an overview of directions of current and future study. The results state above do not yet fully close the question of minimizing the memory required by the agent to retaining a fast cover time, such as $\tilde{O}(n^2)$ (where the $\tilde{O}$ notation disregards polylogarithmic factors). In fact, in current work-in-progress [127], we have designed a strategy with $\tilde{O}(n^2)$ cover time and a state memory requirement of $O(\log \log \log n)$, which we conjecture to be optimal.

Whereas none of the stated approaches require knowledge of global parameters, it is plausible that under the even stricter restrictions of constant-size memory, knowledge of n may affect the feasibility of achieving exploration with small cover time. For example, by rounding the degrees of nodes up to the nearest power of $\sqrt{n}$ (rather than to the nearest power of 2) and then merging the test/revert phases into one, one can design a strategy for graph exploration in $\tilde{O}(n^{2.5})$ time, requiring precisely 1 bit of state memory. This approach relies on the knowledge of a bound or estimate on $\sqrt{n}$. No similar strategy is known for the case of algorithms without global knowledge. In fact, we conjecture that any algorithm with a constant number of bits of memory must have a cover time of $\Omega(n^3)$, i.e., cannot provide an improvement with respect to the random walk. However, proving tight lower bounds on the performance of randomized algorithms is often a challenging task.

We remark that the Metropolis walk is not faster than the random walk in all graph classes. The Metropolis walk visits all the nodes of a graph within $\tilde{O}(n^2)$ steps, which for all but sparse graphs is an improvement with respect to the bound of $O(nm)$ on the cover time of the simple random walk. Nevertheless, applications in sampling tasks show empirically that a simple random walk after erasing null moves is faster (by a constant factor) than Metropolis for some real-world topologies of social networks [136]. Moreover, there exist graph classes for which the random walk has an asymptotically

smaller cover time than the Metropolis walk. A generic example of such a graph, called the *glitter star*, was defined by [147] as a tree on $n = 2l+1$ nodes, with one central node of degree l connected to l nodes of degree 2, which are in turn connected to l leaves. On the glitter star, the cover time of the random walk is $\Theta(n \log n)$, and the cover time of the Metropolis-Hastings walk (with uniform stationary distribution) is $\Theta(n^2)$. To alleviate such undesirable behavior, we propose an approach which combines some of the advantages of the random walk and the Metropolis walk. One way to achieve this is to design an agent which iteratively performs a phase of the random walk, followed by a phase of the uniform Metropolis walk of the same length, doubling the lengths of both walks in each subsequent iteration. Such a walk visits all the nodes of the graph in expected time asymptotically equal to the cover time of the faster of the two walks.

Theorem 2.5 ([T1])**.** *There exists a mobile agent process with $O(\log n)$ state memory which covers any graph G in expected time $C(G) = O(\min\{C_R(G), C_M(G)\})$, where $C_R(G)$ is the cover time of the random walk, and $C_M(G)$ is the cover time of the Metropolis walk in graph G.*

A similar effect can also be achieved by an (almost) Markovian process, outlined in [T1], which consists of a Metropolis-type transition rule between nodes, with transition probabilities being a weighted average of those of the random walk and the uniform Metropolis walk. This leads to an analogous bound on cover time, but requires the knowledge of the average graph degree, $d^* = 2m/n$, by the agent.

Other properties of the Metropolis walk. In our work [T1], we show some interesting properties of the Metropolis walk. First of all, we prove that short Metropolis walks quickly discover many nodes of the network. Metropolis walks of length $t > \Delta^2$ discover new nodes of the graph quickly, with the number of nodes visited being proportional to the square root of the length of the walk. Formally, in [T1] we show that within t steps ($0 < t < 6n^2$), a Metropolis walk will return to its initial location at most $5\sqrt{t} + 2\Delta$ times in expectation. The proof techniques rely on the resistor network analogy for commute times of the walk, and resemble the approach used to study short random walks in regular graphs in [6]. It is then shown that the probability of reaching a node picked uniformly at random within t steps is at least $0.1\sqrt{t}/n$, if $\Delta^2 \leq t < 6n^2$. By taking the union over all n nodes, we have the following corollary.

Theorem 2.6 ([T1])**.** *A Metropolis walk of length $\Delta^2 < t < 6n^2$ visits $\Omega(\sqrt{t})$ distinct nodes of the graph, in expectation.*

By applying the union bound, we describe the cover time of a group of Metropolis walks deployed in the graph; the results from [T1] are rephrased here in mobile agent terminology.

Theorem 2.7 ([T1])**.** *A team of k mobile agents, initially placed at nodes of the network chosen uniformly at random, with each agent performing a Metropolis walk, covers all of the nodes of the graph w.h.p. within $\tilde{O}(\max\{\Delta^2, n^2/k^2\})$ steps (where the $\tilde{O}$-notation disregards polylogarithmic factors).*

The above theorem shows that exploration strategies based on Metropolis walks parallelize well, obtaining a speedup of $\tilde{\Theta}(k^2)$ with respect to the known bounds for the single-agent case, provided that the agents start from nodes which are spread out through the network, and that the degree of the network is not too high.

This type of argument is further exploited in [T1] in combination with so-called landmark distribution schemes due to Broder *et al.* [42] to obtain the main result of [T1], set in the centralized model of computation. We obtain that the classical ST-connectivity problem, which consists in determining if a pair of nodes s, t of a graph given at input belong to the same connected component, can be solved in time T and space S with a time-space tradeoff: $S \cdot T = \tilde{O}(n^2)$, with bounded probability of error. This improves on a series of previous tradeoffs which relied on the application of random walks, with the previously best trade-off being one of $\tilde{O}(nm)$, due to Feige [84].

A comparison of the most important exploration properties of the random walk and the Metropolis walk is provided in Table 2.2.

2.3 Deterministic exploration strategies

Designing algorithms for agents which explore all graphs with certainty requires strictly more resources than the design of algorithms working with bounded probability of error. It is well known that in the absence of any device for marking nodes, no memoryless robot can deterministically explore all anonymous graphs [45]. In [159], this impossibility result was extended to a finite team of robots, showing that they cannot even explore all environments belonging to the class of planar cubic graphs. The result was further strengthened in [50], where the authors introduce a powerful tool, called the Jumping Automaton for Graphs (JAG). A JAG is a finite team of finite automata that permanently cooperate and that can use *teleportation* to move from their current location to the location of any other automaton. However, it turns out that not even JAGs (with constant state memory) can explore all graphs.

The precise requirement on the amount of state memory necessary for an agent to explore all graphs on at most n nodes is $\Theta(\log n)$. The negative result, i.e. a proof that a robot requires at least n states (and thus $\Omega(\log n)$ bits of state memory) to explore all graphs of order n, can be found in [93]. On the other hand, exploration using $O(\log n)$ bits of state memory can be achieved in polynomial time using so-called *universal sequences*.

Rather than treat such universal sequences as a special case of deterministic exploration strategies, we will use universal sequences as a starting point, and then show that

TABLE 2.2: A comparison of exploration algorithms: the random walk, the Metropolis walk, and the rotor-router model. Results which follow from papers [T1, T3, T4] are marked with bullets (•).

Property	Random walk	Metropolis walk	Rotor-router
System model:	Anonymous network	Anonymous network	Anonymous network with a pointer at each node
Determinism:	Randomized	Randomized	Deterministic
Stable state behaviour:	Visits edges equitably	Visits nodes equitably	Visits edges equitably (Eulerian traversal)
Cover time w.r.t. n:	$O(n^3)$	$O(n^2 \log n)$•	$O(n^3)$
Cover time w.r.t. m, D:	$O(mD \log n)$	$O(mD \log n)$•	$\Theta(mD)$•
Limit cover time:	As above (Markovian process)	As above (almost-Markovian process)	Covers graph in $2m$ steps (Eulerian traversal)
Recovery after change in network topology:	None required (Markovian process)	None required (almost-Markovian process)	Stabilizes in $2mf$ steps (f new edges or pointer changes)•; slow for edge deletions•
Worst-case performance:	Unbounded	Unbounded	As in average-case
Graph discovery in t steps:	Discovers $\sim \sqrt{t}$ edges	Discovers $\sim \sqrt{t}$ nodes $(t > \Delta^2)$•	Discovers $\sim \sqrt{t}$ edges•
Cover time using k walks in parallel:	$O(k \log n)$-fold speed-up; $\Omega(k)$-fold speed-up in many graphs (for small k)	Speed-up unknown in general; $\tilde{O}(\Delta^2 + n^2/k^2)$ cover time for uniformly random initialization•	Speed-up unknown; no slow-down possible

an arbitrary deterministic strategy can be viewed as a so-called *universal table*, which generalizes the notion of a universal sequence to take into account the state of the agent.

2.3.1 Universal sequences

Anonymous graphs can be explored using so called *universal traversal sequences* (UTS-s). A UTS describes a sequence of port numbers by which the agent should progress along when leaving its location in successive steps. Formally, let $(a_1, a_2, \ldots, a_t)$ be a sequence of integers. An *application* of this sequence to a graph G at node u is the sequence of nodes $(u_0, \ldots, u_{t+1})$ obtained as follows: $u_0 = u$; for any $0 \leq i \leq t$, $u_{i+1} = succ(u_i, a_i)$ if $a_i < \deg(u_i)$, and $u_{i+1} = u_i$ otherwise. A sequence $(a_1, a_2, \ldots, a_t)$ whose application to a graph G at any node u contains all nodes of this graph is called a traversal sequence for this graph, and a traversal sequence for all graphs in some class $\mathcal{G}$ is called a UTS for $\mathcal{G}$. An agent can traverse a fixed (known) UTS A using only $O(\log |A|)$ state memory, since the agent's move at time t depends only on the value of t, which can be used to compute the current element a_t of the sequence.

Aleliunas et al. [7] observed that there exists UTS-s of length polynomial in n for the class of all graphs of at most n nodes. The applied argument is based on the probabilistic method and thus non-constructive. The length of the obtained UTS for the class of port-labeled graphs $\mathcal{G}$ is given as $O(\max_{G \in \mathcal{G}} C_R(G) \cdot \log |\mathcal{G}|)$, where $C_R(G)$ represents the cover time of the random walk in G. This may be bounded for graphs with at most n nodes and degree at most d as $O(n^3 d^2 \log n)$, since $\max_{G \in \mathcal{G}} C_R(G) = O(n^2 d)$ and $|\mathcal{G}| = O(2^{nd \log n})$ [7]. A mobile agent, given fixed n and d, can always compute a distinguished UTS (e.g., the lexicographically smallest among all shortest UTS-s for the class $\mathcal{G}$) and follow it in the graph.

Proposition 2.8 ([7])**.** *For any positive integers n, d, $d < n$, there exists an algorithm for a mobile agent with $O(\log n)$ state memory, which explores any anonymous graph of at most n nodes and maximum degree at most d in $O(n^3 d^2 \log n)$ steps.*

We remark that although the state memory carried over edges by the agent need only encode its position in the UTS, at every step, the agent has to generate the complete UTS from scratch to be able to perform its next move. No polynomial time routines for finding a UTS of length $O(n^3 d^2 \log n)$ are known to date.

At the cost of increasing the cover time, it is possible, however, to bound the time and memory of local computations of the agent. The approach relies on the application of so-called *Universal Exploration Sequences (UXS-s)* in place of UTS-s. A UXS describes, for each step of the walk, the offset of port by which the agent leaves a node with respect to the port by which it entered the node. Formally, for a given integer d, a sequence of integers $(a_1, a_2, \ldots, a_t)$, $0 \leq a_i < d$, is called a (n, d)-UXS if for any graph $G = (V, E)$ of maximum degree at most d and at most n nodes, all of the nodes of G

are visited by an agent which starts at an arbitrary node $u_0 \in V$ and visits the sequence of nodes $(u_0, \ldots, u_{t+1})$, obtained as follows: $u_1 = succ(u_0, 0)$; for any $1 \leq i \leq t$, $u_{i+1} = succ(u_i, (p_i + a_i) \mod \deg(u_i))$, where p_i is the port by which the agent entered u_i in step i.

In general, the probabilistic method of Aleliunas *et al.* [7] can be used to construct UXS-s of the same length as UTS-s for arbitrary graphs. However, UXS-s have a number of advantages over UTS-s, e.g., an agent following a known UXS can always apply a reversed sequence to return to its initial location [128]. Certain properties of UXS-s may also be exploited to discover the topology of an anonymous graph and help break symmetries between multiple agent; we apply such techniques when considering the rendezvous an mapping problems in Chapters 3 and 4.

A recent result by Reingold [156] implies that a UXS-type exploration procedure can be constructed in logarithmic space of local computation for any graph*, and that this algorithm can even be operated by a single jumping automaton (or equivalently, a single mobile agent moving around the ground). It follows that there exists a deterministic strategy for a mobile agent which explores any graph in a polynomial number of steps, using logarithmic-space (and consequently also polynomial-time) local operations, only. The number of steps of such a sequence is, however, somewhat too large to be of practical significance — on the order of at least n^{100} in Reingold's original implementation.

2.3.2 Universal tables

An extension of the concept of a UTS or UXS is captured by so-called exploration tables. We introduce such tables by generalizing the formulation from [69], in which similar tables were used to derandomize the random walk. An *exploration table* A corresponding to t steps of exploration is an array of indexed pairs of cells of the form $[(a_{i,p,s,\deg}, s'_{i,p,s,\deg})]$, with $1 \leq i \leq t$, $0 \leq p < \deg \leq d$, $s \in S$. Then, an agent following exploration table A is defined so as to have set of states S, and to behave in the i-th step of execution as follows: if the agent entered the current node v in step $i-1$ by port p in state s, then it should leave v in step i by port $(p + a_{i,p,s,\deg(v)}) \mod \deg(v)$ in state $s'_{i,p,s,\deg(v)} \in S$. Here, we choose to define the port of exit as $(p + a_{i,p,s,\deg(v)}) \mod \deg(v)$ rather than $a_{i,p,s,\deg(v)})$ so as to obtain the analogue of a UXS (and not a UTS), giving certain desirable properties, such as the ability of the agent to backtrack its steps knowing only the definition of the table.

We remark that an agent following a universal table is precisely as powerful as an agent which is allowed to follow an *arbitrary* deterministic exploration strategy in an

*While Corollary 5.5 from [156] concerns only regular graphs, the reduction mentioned before this corollary reduces a UXS on arbitrary N-node graphs to a UXS on π-consistently labeled N^2-node graphs of degree 3. Thus the more general result for arbitrary, not necessarily regular graphs, holds as well.

anonymous graph. The difference is purely notational: in a universal table, we emphasize the role of the time step index i, which would normally be concealed as a part of the state of the agent carried over edges. The memory requirement for an agent following an exploration table A for $|A|$ steps is $\log|S|+\log|A|$, where the last $\log|A|$ bits encode the current value of the time step.

Construction. The explicit time dimension in the table proves conceptually helpful when designing a universal table by means of derandomizing any chosen randomized exploration algorithm, such as the random walk or the Metropolis walk, relying on the probabilistic method. We assume that the considered randomized algorithm has a fixed set of states S (corresponding to those in the table), and that the table is designed for an agent operating in an anonymous graph of maximum degree at most d. Intuitively, one considers a set of exploration tables $\mathcal{T}$ such that an algorithm which picks a table $T \in \mathcal{T}$ uniformly at random and then follows exploration according to T is indistinguishable from the original randomized algorithm based on which the set $\mathcal{T}$ is designed (e.g., the random walk or the Metropolis walk). The set $\mathcal{T}$ is designed for an exploration length t corresponding to the cover time of the considered randomized algorithm within the chosen graph class $\mathcal{G}$. Based on this, one obtains by the probabilistic method that there exists a specific table T' of length $t \cdot \log|\mathcal{G}|$, obtained by the concatenation of $\log|\mathcal{G}|$ specifically chosen tables from $\mathcal{T}$, which is a *universal table* for the class $\mathcal{G}$, i.e., an exploration following T' starting from an arbitrary initial node of any graph $G \in \mathcal{G}$ visits all of the nodes of G. Specifically, such an application of the probabilistic method is performed in [69] for the slightly simplified case of a random walk (which is a stateless strategy, hence states s and s' do not need to appear among the indices and values stored in the table). They obtain that for the class of graphs $\mathcal{G}$ with at most n nodes and degree at most d, there exists a universal table, which explores any graph from $\mathcal{G}$ in $O(\max_{G\in\mathcal{G}} C_R(G) \cdot \log|\mathcal{G}|)$ steps, where $C_R(G)$ is the cover time of the random walk. This corresponds precisely to known bounds on the length of a UTS. In fact, one can use exactly the same approach to derandomize the Metropolis walk, obtaining a table with $O(\max_{G\in\mathcal{G}} C_M(G) \cdot \log|\mathcal{G}|)$ steps, where $C_M(G)$ is the cover time of the Metropolis walk. Recalling that $\max_{G\in\mathcal{G}} C_M(G) = O(n^2 \log n)$ by Corollary 2.3 and $|\mathcal{G}| = O(2^{nd\log n})$, we obtain the following theorem.

Theorem 2.9. *For any positive integers n, d, $d < n$, there exists an algorithm for a mobile agent with $O(\log n)$ state memory, which explores any anonymous graph of at most n nodes and maximum degree at most d in $O(n^3 d \log^2 n)$ steps.*

The above algorithm can be designed to operate without knowledge of d: the agent can start with an assumed value of d corresponding the degree of the initial location of the agent, and after the specified number of steps ($O(n^3 d \log^2 n)$) the agent is guaranteed either to have explored the whole graph, or to have found a node of degree higher than

d. In the latter case, the process is restarted with a doubled assumed value of d. The process is guaranteed to terminate within $O(n^3 d \log^3 n)$ steps. Alternatively, one can simply put $d = n$, and run the algorithm in $O(n^4 \log^2 n)$ steps (for simplicity, this is the bound on the exploration time of deterministic algorithms stated in Table 2.1).

As in the case of randomized algorithms, knowledge of an upper bound on n is required to guarantee termination of the exploration process. We note that universal tables based on the Metropolis walk lead to strictly shorter tables than UTS-s.

Lower bounds. The question of the existence of faster deterministic algorithms than those requiring $\tilde{\Theta}(n^3 d)$ steps, or equivalently shorter universal tables, is not well understood. The best currently known lower bound on the length of a universal table follows from a classical lower bound of $\Omega(n^2)$ on the length of a UTS in 3-regular graphs [39]. It is known that UTS-s of length $\Omega(n^2)$ are required to explore all graphs belonging to a special class of labeled 3-regular graphs, which can be chosen so that in each graph the used labeling has the property that for every edge, the two port numbers at the endpoint of each edge are the same. For such graphs, the models of a UTS, a UXS, and a universal table of fixed length have exactly the same "power", i.e., any table or sequence of one type can be converted into another, preserving the behaviour of the agent. Hence, the lower bound of $\Omega(n^2)$ holds for all three models, and so is also a lower bound on the time of deterministic exploration in the anonymous network model.

Most of the research on lower bounds in deterministic exploration has concerned UTS-s, only, with the best currently known lower bound for UTS-s in graphs of maximum degree d being $\Omega(n^{2.51} d^{0.49})$ [61]. This result is the last in a long sequence of incremental improvements based on a proof technique which makes use of constructions of so-called reflecting sequences to design port labelings forcing the application of long UTS-s on the ring, which are then generalized to arbitrary regular graph. As pointed out by Koucky [128], lower bounds of this type do not carry over even to the case of UXS-s, since a sequence of successive port increments of '1' corresponds to the well-known basic walk (right hand rule) strategy, and is sufficient to explore graphs such as rings and trees in linear time. Finding lower bounds for universal tables appears to be even more complicated.

Randomization of universal tables. When designing efficient deterministic exploration algorithms, we obtained a universal table by derandomizing some form of biased random walk. However, it is also possible to perform the process in the opposite direction, adding elements of randomness into a universal table to obtain a randomized exploration strategy. Assuming the agent knows a bound τ on the number of steps of exploration it can perform and on the degree of the graph, we can consider a probability distribution $\rho : \mathcal{T} \to [0, 1]$ where $\mathcal{T}$ is the set of (not necessarily universal) exploration tables of length at most τ. Now, an agent can proceed with the following randomized

exploration algorithm: in the first step, it picks a table $T \in \mathcal{T}$ according to probability distribution ρ, and then deterministically follows table T for τ steps. Such an approach is extremely powerful. In fact, subject to some assumptions on known upper bounds of parameters n, d, and τ, it can be seen as universal: it is well known that any randomized exploration strategy, which relies on some coin tosses, can be converted into one where all the random decisions are decided before the first step of the agent, and the subsequent execution of the procedure is deterministic. Since any deterministic strategy is equivalent to some exploration table, the universality of the adopted approach follows.

The problem of designing a randomized exploration table which is superior to the Metropolis walk is a challenging research question. Intuitively, it might seem that in regular graphs with a port labeling which provides the agent with no additional information about the topology of the graph, the $\Theta(n^2)$ cover time achieved by both the random walk and the Metropolis walk is the best possible. However, none of the known lower bounds of $\Omega(n^2)$ on cover time for specific classes of algorithms extends to the general case of randomized tables.

2.4 Exploration of anonymous networks with preconfigured port numbers

In this section we consider a natural variant of the exploration problem in anonymous networks, in which the designer of the agent's exploration strategy initially sets up the port labels and the ordering of the local port numbers in the network, so as to allow a very simple agent to efficiently explore all nodes of the network. This line of study was initiated in [75].

We will study the exploration problem with preset ports in the context of *periodic* graph exploration. The task of visiting all the vertices of a network periodically is particularly useful in network maintenance, where the status of every vertex has to be checked regularly. It turns out that, due to the ability to preset the port numbers, it is possible to design a memoryless agent which visits all nodes (i.e., an agent which is not equipped with any state information which survives when traversing an edge). It turns out that it is even possible to visit all nodes of the graph with a memoryless robot periodically, once every a number of steps which is linear in n. This shows that the ability to configure ports is extremely powerful in network exploration. We recall that for arbitrary port settings, exploration was only feasible for an agent equipped with $\Omega(\log n)$ bits of state memory (cf. Section 2.3).

Formally, we consider a scenario in which the agent is initially located at some vertex v, and starts the exploration of G by traversing the edge having label 0 at endpoint v. Once it has reached the other endpoint of this edge, say some node u, it reads the associated label, say l, and enters node u. When the agent is memoryless, the port

by which the agent leaves v is a function of the input port. In fact, we will restrict our considerations to agents following the so-called *basic walk* (or the *right-hand rule* [56]), in which vertex u is left by the port labeled *NextPort*(u), i.e., the next port after the port of entry in some cyclic ordering of the ports at u. The basic walk strategy has been shown to be the best possible among all memoryless exploration strategies, for any graph [56]. After performing a certain number of steps according to the rule, the agent will eventually re-enter port 0 at vertex v, and the traversal will proceed periodically from then on. We will say that the agent *explores* the graph if its route goes through each vertex of the graph at least once; from now on, we will only consider port labelings leading to valid explorations. It is known that all graphs admit a port labeling leading to an exploration [75].

For a given port labeling, the *exploration period* π is defined as the total number of steps made by the agent before returning to the initial port (or equivalently, as the total number of arcs of the form (u, v), for $\{u, v\} \in E$, used during the exploration). Herein, we focus on finding labelings which lead to valid explorations of minimum possible period when using a memoryless agent. This immediately leads to the natural definition of the graph parameter $\pi(G)$ known as the *minimum exploration period* of the graph.

The first constructions of port labelings leading to short exploration periods for a memoryless robot were provided in [75], showing that for any graph on n nodes, we have $\pi(G) \leq 10n$. Recently, by applying a clever graph decomposition technique in order to build an appropriate exploration cycle, [56] have improved this bound to $\pi(G) \leq \frac{13}{3}n \approx 4.33n$. They have also shown a strong worst-case lower bound: for arbitrarily large values of n, there exist n-node graphs G_n such that $\pi(G_n) \geq 2.8n - O(1)$. Finally, the following improved bound on the minimum exploration period of any graph was shown in [T2]: $\pi(G) \leq 4n - 2$. The proof of this result is constructive, and in the rest of this section we discuss in more detail how to algorithmically construct a port labeling which guarantees an exploration period of at most $4n - 2$ for the basic walk.

We note that the value of $\pi(G)$, expressed in relation to the number of nodes n, exposes certain interesting structural properties of the graph. For example, we have that $\pi(G) = n$ if and only if G is Hamiltonian, and for a Hamiltonian graph an appropriate labeling can be defined so as to direct ports 0 and 1 of all nodes along the edges of the Hamiltonian cycle (Fig. 2.1a). It is also known that $\pi(G) < 2n$ for all graphs admitting a spanning tree T such that $G \setminus T$ has no isolated vertices [56].

2.4.1 Construction of Port Labeling for the Basic Walk

The port labeling will be constructed in such a way that the agent will perform one period of its traversal within $4n - 2$ steps, and hence will only visit a small number of edges of the traversed graph G. The general approach is to construct a multigraph H on the same set of nodes V as G, adding one copy of an edge between a pair of nodes each time the agent traverses an edge between this pair of nodes in its walk on G. We apply

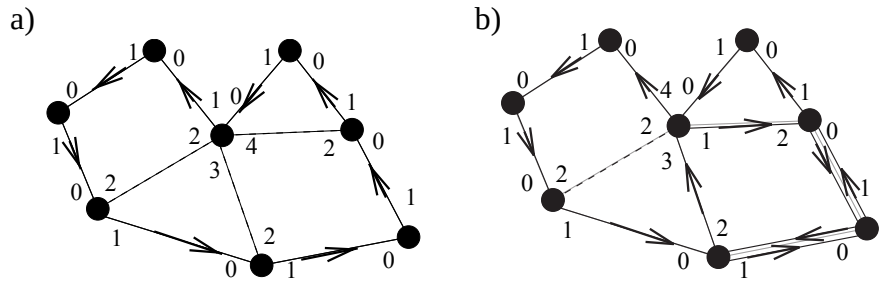

FIGURE 2.1: Exploration cycles obtained for different labelings: (a) a labeling leading to a Hamiltonian cycle, (b) another exemplary labeling.

a method introduced in [56], which consists in proving that there exists a multigraph H on the set of nodes and edges of G satisfying a certain set of properties, and next, that given such a multigraph H, we can design a periodic walk spanning all nodes from V, whose period is at most the number of edges of H. In our case, we will expect the multigraph H to have at most $4n - 2$ edges (counting all parallel edges).

For a multigraph H, we will denote by $V(H)$ its vertex set, by $E(H)$ its edge multiset, and by $|E(H)|$ the number of its edges (including multiple edges). The number of edges adjacent in H to a vertex $v \in V(H)$ is denoted by $\deg_H(v)$. The notation $2e$ denotes 2 copies of an edge e; the notation $2H$ denotes a multigraph with vertex set $V(H)$ and each edge $e \in E(H)$ replaced by $2e$. An edge $e \in H$ is called *double* if $2e$ belongs to H, and *single* otherwise. Throughout the section, we will never consider multigraphs with more than two parallel edges.

Any labeling scheme for graph G uniquely determines an exploration cycle, understood as a sequence of directed edges traversed in a single period of the exploration, i.e., a sequence in which the directed edge (u, v) corresponds to a transition of the robot from some node u to another node v, where $\{u, v\} \in E$. The corresponding *exploration multigraph* H is defined as the undirected submultigraph of $2G$ given by the edges of G traversed by the robot during one exploration cycle. Each edge is included as it is traversed, possibly twice if it is traversed in both directions; note that no edge can be traversed twice in the same direction, since then the memoryless agent would start the next period of its traversal. Let H_2 be the spanning subgraph of H consisting of its double edges only, and let $H_1 = H \setminus H_2$. A vertex $v \in V$ is called *saturated* in H with respect to G if $\deg_H(v) = \deg_{2G}(v)$. The following property of exploration multigraphs is known to hold.

Proposition 2.10 ([56])**.** *Any exploration multigraph $H \subseteq 2G$ has the following properties:*

A. For each vertex $v \in V$, $\deg_H(v)$ is even.

B. Each vertex $v \in V$ having $\deg_{H_1}(v) = 0$ is saturated in H with respect to G.

The converse of the above proposition does not hold in general, but one more additional property can also be formulated.

C. H_2 *is connected.*

Then, the following structural theorem has recently been shown.

Theorem 2.11 ([56])**.** *Any multigraph* $H \subseteq 2G$ *fulfilling properties A, B, and C is a valid exploration multigraph, i.e. induces an exploration cycle on* G *of length at most* $|E(H)|$.

Consequently, for the rest of the section we will concentrate on defining a multigraph $H \subseteq 2G$ which satisfies properties A, B, and C. To achieve this, in graph G we select an arbitrary spanning tree T_0. Let $G' = G \setminus T_0$. Then, in multigraph $2G'$ we find a spanning (not necessarily connected) submultigraph H' satisfying properties analogous to A and B:

A'. *For each vertex* $v \in V$, $\deg_{H'}(v)$ *is even.*

B'. *Each vertex* $v \in V$ *having* $\deg_{H'_1}(v) = 0$ *is saturated in* H' *with respect to* G'.

The final multigraph H is given as $H = H' \cup 2T_0$, thus $2T_0 \subseteq H_2$. It is clear that if H' satisfies properties A' and B', then H satisfies properties A, B, C, and that $|E(H)| = |E(H')|+2(n-1)$. Hence, in order to obtain an exploration cycle with period $\pi(G) \leq 4n-2$, we confine ourselves to constructing an appropriate submultigraph $H' \subseteq 2G'$ with $|E(H')| \leq 2n$.

Note that the construction of H' can be performed independently for each connected component of G'; throughout the rest of the discussion, w.l.o.g. we assume that G' is connected. The existence of an appropriate multigraph H' constitutes the main result of [T2].

Theorem 2.12 ([T2])**.** *For any connected graph* G' *with vertex set* V, $|V| = n$, *there exists a multigraph* $H' \subseteq 2G'$ *such that* $|E(H')| \leq 2n$, *and* H' *satisfies properties* A' *and* B'.

Construction of multigraph H'

In [T2], it is shown that the multigraph H' satisfying properties A' and B', such that $|E(H')| \leq 2n$, can be determined by Algorithm 2. In it, we use the following notation: for a spanning tree rooted at node r in G', we will call a vertex $v \in V$ *tree-saturated* in T if $\deg_T(v) = \deg_{G'}(v)$. For tree T, let $s(T)$ denote the number of tree-saturated vertices in T, and let $s_h(T)$, for $0 \leq h < n$, be the number of tree-saturated vertices in T at height (i.e. distance in tree T from root r to the vertex) not greater than h. This allows us to define a partial order on the set of rooted spanning trees of G'. For a pair of trees T_a, T_b, we will say that $T_a < T_b$ if one of the following conditions is fulfilled.

1. $s(T_a) < s(T_b)$,
2. $s(T_a) = s(T_b)$, and for some h, $0 \leq h < n$, we have $\forall_{0 \leq l < h}\ s_l(T_a) = s_l(T_b)$ and $s_h(T_a) > s_h(T_b)$.

Algorithm 2: Computing multigraph H'

1. Let T be a minimal spanning tree in G' with respect to order $(<)$.
2. Let $\mathcal{S}$ be a subgraph in $G' \setminus T$, whose connected components are stars, such that for each $v \in V$ either v is tree-saturated in T or $\deg_{\mathcal{S}}(v) > 0$.
3. Find a submultigraph $H' \subseteq \mathcal{S} \cup 2T$ fulfilling properties A' and B', such that $|E(H')| \leq 2n$, and return it as output.

It turns out that all of the steps of the above algorithm are well defined. Step (1) requires no comment. For step (2), graph $\mathcal{S}$ is well defined because any graph admits a subgraph which is a set of stars, touching all non-isolated vertices; for graph $G' \setminus T$, the only isolated vertices are those which were tree-saturated in T. The correctness of step (3) is the result of the appropriate choice of spanning tree T. The selection of the edges which are included in H' can be implemented by applying several auxiliary procedures which select edges incident to nodes of tree T, processing these nodes level by level, starting from the leaves and moving towards the root of the tree [T2]. The correctness of this construction of H' implies the following theorem.

Theorem 2.13 ([T2])**.** *For any graph G of size n there exists a port labeling leading to an exploration period of the basic walk: $\pi \leq 4n - 2$.*

It is natural to ask about the runtime of the procedure required to obtain a labeling with such an exploration period, and about the tightness of the obtained bound; we address these questions in the following subsections.

Whereas the construction of the appropriate cycle can always be performed using Algorithm 2 (in finite time), this does not necessarily mean that a solution can be found in polynomial time. The problem consists in computing an appropriate spanning tree, minimal in the sense of order $(<)$, in step (1). In general, finding a spanning tree having a minimum number of saturated vertices is already NP-hard. (The proof of this observation proceeds by reduction from the problem of finding a Hamiltonian path in a 3-regular graph: a 3-regular graph has a spanning tree without saturated vertices if and only if it admits a Hamiltonian path.) However, this problem was alleviated in [T2] by selecting a different partial order on spanning trees, having a less intuitive definition, but satisfying the same essential properties and allowing for faster processing.

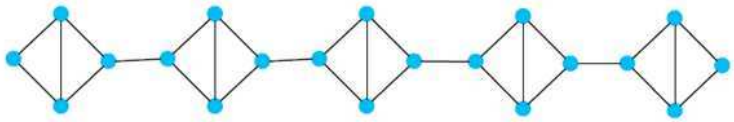

FIGURE 2.2: Example of a class of graphs in which the exploration period of the basic walk is $2.8n - O(1)$. [56]

Theorem 2.14 ([T2])**.** *There exists a polynomial time algorithm which, given a graph G, determines a port labeling leading to an exploration period $\pi \leq 4n - 2$.*

Tightness of the bound

The best known existential lower bound on the length of an exploration cycle is within an additive factor of $2.8n$ [56]. The class of graphs attaining such a bound is obtained by connecting 4-vertex diamond graphs into a chain, as presented in Figure 2.2. This leaves a gap between the lower bound, and the best known labeling with a period of $4n - 2$, given by Theorem 2.13.

However, obtaining exploration periods significantly shorter than $4n$ would require some completely new insight. All of the approaches known to date, including those from [56, T2], rely on constructions of exploration multigraphs in which the set of edges traversed twice during the exploration period touches all of the nodes of the graph (formally, for all $v \in V$, $\deg_{H_2}(v) > 0$). This property would have to be discarded to allow further improvement to the length of the exploration period in view of the following result.

Theorem 2.15. *For all values of $n \geq 3$, there exists a graph G of order n, such that any exploration of G whose exploration multigraph satisfies for all $v \in V$, $\deg_{H_2}(v) > 0$) has $|E(H)| \geq 4n - 8$ edges.*

We remark that in particular, the construction would need to avoid the condition imposed on the double-edge subgraph of multigraph H (property C in Theorem 2.10).

2.4.2 Agents with Small Memory

A variation of the considered problem was proposed in [109], where the robot is equipped with few extra memory bits. We will denote the exploration periods in such a model by π_c. In [109] it is shown how to obtain an exploration period $\pi_c(G) \leq 4n - 2$, regardless of the starting vertex of the robot. The obtained bound has since been improved in [97] to $\pi_c(G) < 3.75n - 2$ by exploiting some particular graph properties, still allowing only constant memory. The constant memory model was also addressed in [56] and the bound was further improved to $\pi_c(G) < 3.5n - 2$ by using a combination of the properties from [97] and the new decomposition technique also used in [97] for the oblivious

case. Interestingly enough, apart from the relation $\pi_c(G_n) \geq 2n-2$ which clearly holds whenever G_n is a tree on n nodes, there are to date no known non-trivial lower bounds on the worst case value of parameter π_c.

2.5 Exploration in network models augmented by edge counters

The exploration of anonymous graphs can be accelerated by providing additional information to the agent at nodes of the environment. In the extreme case of graphs in which nodes have unique identifiers available to the agent, but not known in advance, one obtains an online graph exploration problem which has been studied in the context of minimizing the time of first cover in terms of either the number of moves (edge traversals) or the amount of memory used by the agent. Panaite and Pelc [151] gave an algorithm for exploring labelled undirected graphs that uses $m + O(n)$ moves, improving on the standard Depth-First Search algorithm that takes $2m$ moves. Collaborative versions of the same problem, employing multiple agents, have been considered in [67, 92]. Deng and Papadimitrou [66] as well as Albers and Henzinger [2] studied the exploration of strongly connected directed graphs under the same conditions. There have also been some studies on the efficiency of exploration when some prior information about the graph is available with the agent — for instance, when the agent possesses an unlabelled isomorphic map of the graph [152]. For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [29] proposed the method of dropping a pebble on a node to mark it and showed that any strongly connected directed graph can be explored using just one pebble, if the size of the graph is known and using $O(\log \log n)$ pebbles, otherwise. Dudek *et al.* [77] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [30] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explores new edges.

Herein, we focus on modifications to the anonymous graph model in which the agent is guided at each node by a simple local rule, influenced by the history of the agent's exploration. Specifically, in a *local exploration strategy* the next node to be visited by the agent should depend only on the values of certain parameters associated with the edges adjacent to the current node. Our main goal is to design local exploration strategies which in some sense mimic the behavior of a random walk in a graph, in an attempt to achieve the same properties of cover time and exploration fairness in the deterministic sense of worst-case performance.

In Section 2.5.1 we consider the so-called rotor-router model, in which the agent at each step exits the current node, using its outgoing ports in round-robin fashion. Two other local strategies, known as Least-Used-First and Oldest-First are discussed in

Section 2.5.2. These strategies make use of counters associated with edges, rather than ports of the graph, and strive to preserve some form of fairness of use of edges during exploration. The obtained results relating to the cover time of these strategies are briefly summarized in Table 2.1.

2.5.1 The rotor-router model

The rotor-router mechanism was introduced as a deterministic alternative to the random walk and studied in the context of a wide selection of network problems, including work on load balancing problems in [53,76], graph exploration [1,91], and stabilisation of distributed processes [32,155,175]. The *rotor-router mechanism* is represented by an undirected anonymous graph $G = (V, E)$. As in the previously considered anonymous model, nodes in V bear no names, however, the endpoints of edges in E, called *ports*, are arranged in a *cyclic order* at each node. For the purposes of the rotor router, no explicit port numbers need to be defined; nevertheless, such a port labeling. Instead, each node is additionally equipped with a *pointer* that indicates the current exit port to be adopted by an agent on the conclusion of the next visit to this node. The rotor-router mechanism guarantees that after each consecutive visit at a node its pointer is moved to the next port in the cyclic order. The port labeling and the initial pointer assignment is treated as part of the rotor-router mechanism, configured before the release of the agent.

In this section we consider the efficiency of graph exploration using the rotor-router mechanism, hence, the most interesting questions for us are how quickly the agent explores the whole graph, and then how evenly it keeps traversing the edges of the graph. We then proceed to study the behavior of the rotor-router under modifications to the topology of the graph. A comparative overview of the main properties of exploration using the rotor-router and the random walk is provided in Table 2.2.

Definition of the rotor-router

We consider the rotor-router model (on graph G) with a single agent. The agent moves in discrete steps from node to node along the arcs of graph $\vec{G} = (V, \vec{E})$ which is the directed symmetric version of G, having the set of arcs $\vec{E} = \{(v, u), (u, v) : \{v, u\} \in E\}$.

A *configuration* at the current step is a triple

$$((\rho_v)_{v\in V},\ (\pi_v)_{v\in V},\ r),$$

where ρ_v is a cyclic order of the arcs (in graph $\vec{G}$) outgoing from node v, π_v is an arc outgoing from node v, which is referred to as *the (current) port pointer at node* v, and r is *the current node* – the node where the agent is at the current step.

For each node $v \in V$, the cyclic order ρ_v of the arcs outgoing from v is fixed at the beginning of exploration and does not change in any way from step to step (unless an

edge is dynamically added or deleted as discussed later on in the Section). For the sake of convenience, we will define the cyclic order ρ_v using the concept of an explicit port assignment (labeling).

For an arc (v, u), let *next*(v, u) denote the arc next after arc (v, u) in the cyclic order ρ_v. During the current step, first the port pointer π_r at the current node r is advanced to the next arc outgoing from r (that is, π_r becomes *next*(π_r)), and then the agent moves from node r traversing the arc π_r.

The exploration starts from some initial configuration and then keeps running without ever terminating. The initial configuration is uniquely defined through the port assignment and an initial pointer assignment, defined as follows.

Definition 2.1. An *initial pointer assignment* to the nodes of an undirected graph $G = (V, E)$ is a function $f : V \to E$, such that for all $v \in V$, $f(v) \in E_G(v)$.

Lock-in time of the rotor-router mechanism

Due to the limited number of configurations in a graph G of bounded size, it is intuitive that a walk of the agent controlled by the rotor-router mechanism must be locked-in in a loop eventually. Moreover, this loop must include all the vertices. We formulate this in the form of the following lemma, attributed to folklore.

Lemma 2.16. *The agent following the rotor-router visits each node infinitely many times (thus traverses each arc infinitely many times).*

Rather surprisingly, however, Priezzhev *et al.* [155] proved that an agent traversing a finite graph gets locked-in to an Euler tour based on arcs obtained by replacing each edge in G with two arcs having opposite directions. More precisely, after the initial *stabilisation (lock-in) period*, the agent keeps repeating the same Eulerian cycle of the directed graph $\vec{G}$. We will subsequently refer to the undirected links in graph G as *edges* and to the directed links in graph $\vec{G}$ as *arcs*. We will also keep using an arrow on the top of a symbol, as in $\vec{G}$ and $\vec{E}$, to stress that we refer to directed graphs and arcs.

The result of Priezzhev at al. means that the rotor-router has a highly desirable property of regularity of edge exploration: namely, after the lock-in period, each edge is visited precisely once every $2m$ steps. This is a much stronger property than the regularity achieved by the random walk, which visits every edge with expected frequency $1/2m$. Independently, Wagner *et al.* [167, 168] showed that, starting from an arbitrary configuration (arbitrary cyclic orders of edges, arbitrary initial values of the port pointers and an arbitrary starting node) the agent covers all edges of the graph within $O(nm)$ steps, where n and m are the number of nodes and the number of edges in the graph. In other words, the cover time of the rotor-router is deterministically bounded by $O(nm)$, matching the (expected value of the) cover time achieved by the random walk (cf. Theorem 2.1).

Bhatt *et al.* [32] strengthened and combined the earlier results of Priezzhev *et al.* and Wagner *et al.*, showing that within $O(nm)$ steps the agent not only covers all edges but already locks in to the traversal of its eventual Eulerian tour. This result was improved by Yanovski *et al.* [175] who showed that the agent locks in to an Eulerian cycle within $2m\,D$ steps.

Theorem 2.17 ([175])**.** *For any graph G, any cyclic order ρ_v of the arcs outgoing from each node $v \in V$, and any initial values of the port pointers π_v, $v \in V$, there exists a time step $t_0 \leq 2mD$ such that from step $t_0 + 1$, the agent keeps repeating the same Eulerian cycle of graph $\vec{G}$.*

This $O(m\,D)$ bound on the lock-in time of the rotor router also corresponds asymptotically to the time required for the rotor-router to visit all nodes of the graph. Thus, it can be seen as a counterpart of the $O(m\,D)$ bound on the cover time of a random walk (cf. Theorem 2.1).

A simple proof of the lock-in bound of Yanovski *et al.*is achieved by studying the saturation of subsequent nodes in the initial phases of the walk. We say that a node becomes *saturated* when all its incident edges are traversed in both directions for the first time. Note that when a node becomes saturated, its pointer returns to the initial position for the first time. A slightly stronger version of this claim is given below.

Lemma 2.18 ([175, T3])**.** *If in the current step i the agent leaves the current node r along an arc (r, y), then the first arc traversed for the second time during the period $i, i + 1, \ldots,$ is this arc (r, y).*

The following lemma provides a characterization of the behavior of the rotor-router system during its first exploration of the graph, before its eventual lock-in into an Eulerian traversal.

Lemma 2.19 ([32])**.** *Let $G = (V, E)$ be a graph with a starting node $s \in V$, an assignment of ports and pointers. The Euler tour lock-in in G is performed in phases $\{P_i\}_{i\geq 1}$. Each phase starts when the mobile agent leaves s via edge $f(s)$ indicated by the initial assignment of pointers and continues until the agent traverses all edges incident to s in both directions. The following properties hold:*

- *While the agent is visiting nodes saturated in some earlier phase, it retraces the route of phase P_{i-1}.*
- *If the agent encounters a node u that has been visited but not saturated in an earlier phase, it suspends the retracing of the tour of phase P_{i-1}. A new tour starts at u and ends there. Node u is now saturated. The tour of phase P_{i-1} is resumed (via port $f(s)$).*
- *Every edge is traversed at most once in each direction during each phase.*

Eventually all nodes in G get saturated. In other words, there exists an integer $j \geq 1$, such that, starting from the phase P_j the agent adopts the same (Euler) tour in G.

One can conclude from Lemma 2.19 that during each phase P_i the agent gets locked in a subgraph G_i of G where:

1. G_0 contains a single node s;
2. each G_i is a subgraph of G_{i+1};
3. all edges of G that are incident to nodes of G_i belong to the edge set of G_{i+1}.

Since the number of edges in each G_i is bounded by m, the $O(m\,D)$ bound on lock-in time follows.

We also remark that Lemma 2.19 implies that, if in some phase i of its exploration the agent has covered all of the nodes of the graph, then $G_i = G$. It follows that the agent will lock in to its Eulerian traversal within $2m$ steps from the moment when it has covered all nodes.

Proposition 2.20. *The difference between the cover time and the lock-in time of the rotor-router system is at most $2m$.*

Subsequently, we will consider an asymptotic analysis of the lock-in time of the rotor-router, noting that this is equivalent (up to constants hidden in notation) to its cover time.

The Euler tour lock-in problem against an adversary

In this section we examine the influence of the initial configuration of pointers and port numbers on the time needed to lock-in the agent in an Euler tour. The case study is performed in the form of 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 assume that both the player $\mathcal{P}$ and its adversary $\mathcal{A}$ have unlimited computational power, i.e., we do not take into account the cost of computation of the initial configuration of ports and pointers to be adopted by $\mathcal{P}$ and $\mathcal{A}$. The results of our studies are asymptotically tight in terms of the worst-case choice of the graph topology and the initial location of the agent.

We start our analysis with border cases. In the case $\mathcal{P}$-*all* where the player $\mathcal{P}$ is in charge of the initial arrangement of port numbers and pointers we observe that the lock-in in an Euler tour can be obtained in time $O(m)$. Also the case $\mathcal{A}(\circlearrowleft)\mathcal{P}(f)$, where $\mathcal{P}$ sets the pointers after the port numbers are assigned by $\mathcal{A}$, reduces to the border case where $\mathcal{P}$ is solely in charge of the initial configuration. On the other hand, in the case $\mathcal{A}$-*all* where the adversary $\mathcal{A}$ solely decides about the initial configuration, we show in [T3] that in any graph with m edges and diameter D the adversary $\mathcal{A}$ is able to enforce the lower bound $\Omega(m \cdot D)$ for the lock-in matching the upper bound from [175].

Theorem 2.21 ([T3]). *For any graph $G = (V, E)$ there exists a starting node s, and a port and pointer assignment in G, s.t., the lock-in requires time at least $\frac{1}{4} \cdot m\, D$.*

The lock-in of the rotor-router occurs shortly after all of the nodes have been discovered. In view of this, we obtain that for any graph G, the worst-case cover time of the rotor-router is $\Theta(m\, D)$.

In view of the above result, it is natural to ask which capability of the adversary contributes more to increasing the lock-in time of the rotor-router: that of setting ports in the graph, or the initial locations of the pointers? In [T3], we establish that for worst-case graphs, control over the initial pointers is more important. Indeed, in the case $\mathcal{A}(f)\mathcal{P}(\circlearrowleft)$, where $\mathcal{P}$ responds by appropriate port assignment to the initial setup of pointers by $\mathcal{A}$, we show that there exist graphs for which the lock-in still requires time $\Omega(m \cdot D)$. For any m and D, $D \leq m$, let $\mathcal{G}_{m,D}$ denote the class of graphs with diameter between D and $4D$ and a number of edges between m and $4m$.

Theorem 2.22 ([T3]). *For any m and $D \leq m$, there is a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ with starting node s, and a pointer assignment, s.t. for any port assignment the lock-in time (and cover time) is at least $\Omega(mD)$.*

At the same time, we present a non-trivial class of graphs with an arbitrarily large diameter in which an appropriate choice of port numbers leads to the lock-in in time $O(m)$.

In the case $\mathcal{P}(\circlearrowleft)\mathcal{A}(f)$ where $\mathcal{A}$ sets the pointers after the assignment of ports is revealed by $\mathcal{P}$, the lower bound of $\Omega(m \cdot D)$ for the lock-in time follows directly from the previous case. Also, here we propose a non-trivial class of graphs, this time with an arbitrary diameter $D \leq \sqrt{n}$, in which the lock-in is feasible in time $O(m)$.

On the other hand, we show that if the initial setup of pointers is performed by $\mathcal{P}$, and $\mathcal{A}$ provides its own port numbering for the graph afterwards (case $\mathcal{P}(f)\mathcal{A}(\circlearrowleft)$), then the complexity of the lock-in problem is bounded by $O(m \cdot \min\{\log m, D\})$.

Theorem 2.23 ([T3]). *For any graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ and any starting point s there exists a pointer assignment, s.t. for any port assignment the lock-in (and covering of the graph) can be obtained in time $O(m \cdot \min\{\log m, D\})$.*

We also propose a respective class of graphs in which the lock-in requires time $\Omega(m \cdot \min\{\log m, D\})$. At the same time we point out that, e.g., in Hamiltonian graphs the lock-in is obtained in time $O(m)$.

Our results are summarised in Table 2.3.

Robustness of the rotor-router mechanism

A useful property of graph exploration based on the rotor-router mechanism is its robustness. In case of link failures or other dynamic changes in the graph, after some

TABLE 2.3: Minimum and maximum values of the lock-in time (and cover time) of the rotor-router for different adversarial scenarios.

Scenario	*Worst case*	*Best case*
Case $\mathcal{P}$-*all*	$\Theta(m)$	$\Theta(m)$
Case $\mathcal{A}(\circlearrowleft)\mathcal{P}(f)$	$\Theta(m)$	$\Theta(m)$
Case $\mathcal{P}(f)\mathcal{A}(\circlearrowleft)$	$\Theta(m \cdot \min\{\log m, D\})$	$\Theta(m)$
Case $\mathcal{A}(f)\mathcal{P}(\circlearrowleft)$	$\Theta(m \cdot D)$	$\Theta(m)$
Case $\mathcal{P}(\circlearrowleft)\mathcal{A}(f)$	$\Theta(m \cdot D)$	$\Theta(m)$ for all $D \leq \sqrt{n}$
Case $\mathcal{A}$-*all*	$\Theta(m \cdot D)$	$\Theta(m \cdot D)$

additional stabilisation period the agent goes back into the regime of repeatedly traversing the graph along a (new) Eulerian cycle. We know that whatever the changes in the graph are, the length of that additional stabilisation period is $O(mD)$ (by Theorem 2.17, that much time is sufficient for establishing an Eulerian cycle from *any* initial configuration) but no better bounds have been shown before.

In this section we establish bounds on the length of that additional stabilisation period which depend on the extent of the failures or changes in the graph. The subsequent analysis of the rotor-router mechanism is based on the relation between the Eulerian cycles in the directed graph $\vec{G}$ and the spanning trees in the undirected graph G which underlies the following theorem. This theorem, sometimes referred to as the "BEST" Theorem, was discovered by de **B**ruijn and van Aardenne-**E**hrenfest [65] on the basis of earlier work by **S**mith and **T**utte [166].

Theorem 2.24 (Bruijn, van Aardenne-Ehrenfest, Smith, Tutte)**.**
The number of Eulerian cycles in the directed, symmetric version of an undirected connected graph $G = (V, E)$ is equal to $\prod_{v \in V}(d(v) - 1)!$ times the number of spanning trees of G, where $d(v)$ is the degree of node v in G.

We now describe the connection between Eulerian cycles and spanning trees in the context of the rotor-router mechanism.

If T is a tree in graph G (not necessarily spanning all nodes of G), then $\vec{T}$ obtained from T by directing all edges towards a selected node v in T is called an *in-bound tree* in $\vec{G}$, and node v is the root of $\vec{T}$. A subset of arcs $\vec{H}$ in $\vec{G}$ is an *in-bound tree with a root cycle*, if it is an in-bound tree with one additional arc outgoing from the root. That additional arc creates a (directed) cycle, which we call a root cycle. We can view $\vec{H}$ as consisting of one cycle (the root cycle) and a number of in-bound node-disjoint trees rooted at nodes of this cycle (no node of such a tree other than the root belongs to the root cycle).

Let $\vec{F} = \{\pi_v : v \in V\}$ be the set of the current port pointers. For the current node r, we are interested in the structure of $\vec{F}_r = \vec{F} \setminus \{\pi_r\}$, since, as we show later, the structure

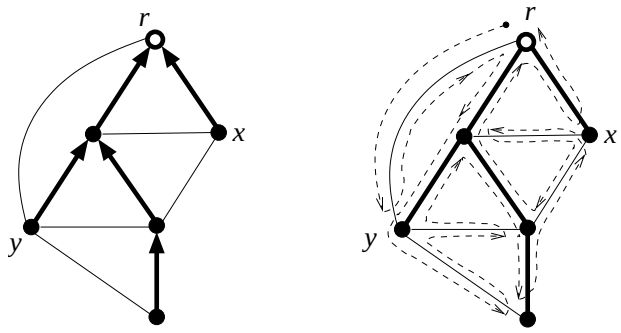

FIGURE 2.3: Left: the leading tree spanning all nodes of the graph (arcs in bold). Right: the corresponding Eulerian cycle, assuming the anti-clockwise order of arcs outgoing from a node (other cycles are obtained for other cyclic orders of arcs).

of $\vec{F}_r$ is a good indicator of how far the agent is from entering an Eulerian cycle. The component of $\vec{F}_r$ containing the current node r is an *in-bound tree* rooted at r, which we call *the leading tree*. Each component $\vec{H}$ of $\vec{F}_r$ other than the leading tree is an *in-bound tree with a root cycle*.

The following Lemma shows that the condition that the agent follows an Eulerian cycle and the condition that the leading tree spans all nodes of the graph are equivalent.

Lemma 2.25 ([T4])**.** *Assume that the current leading tree $\vec{T}$ spans all nodes of the graph. Then during the next $2m$ steps the agent traverses an Eulerian cycle in $\vec{G}$. Moreover, the leading tree after these $2m$ steps is again the same tree $\vec{T}$.*

Conversely, assume that at the current step i the leading tree $\vec{T}$ does not span all nodes of the graph. Then the route $\vec{\Gamma}$ traversed by the agent during the next $2m$ steps is not an Eulerian cycle.

Figure 2.3 illustrates Lemma 2.25. The diagram on the left shows a graph and the current leading tree (arcs in bold) which spans all nodes. The current node r is the root of this tree. The diagram on the right shows the Eulerian cycle followed by the agent. We assume that the cyclic order of the arcs outgoing from a node is the anti-clockwise order, and that arc (r, x) is the current value of the port pointer π_r. Thus, the first arc followed by the agent is arc $(r, y) = \mathit{next}(r, x)$.

The above Lemma implies, in particular, that the Euler tour lock-in time τ for a given initialization of the rotor-router and the time t_0 required to cover all the arcs of the graph satisfy $\tau \leq t_0 \leq \tau + 2m$.

As a side note, we remark that the operation of the rotor-router mechanism can be used to prove Theorem 2.24. We fix a node r as the current node and an arc (r, x) as the current value of the port pointer π_r (the agent will follow in the current step arc $\mathit{next}(r, x)$). Let $\vec{T}$ be an arbitrary in-bound spanning tree of $\vec{G}$ rooted at node r, and let

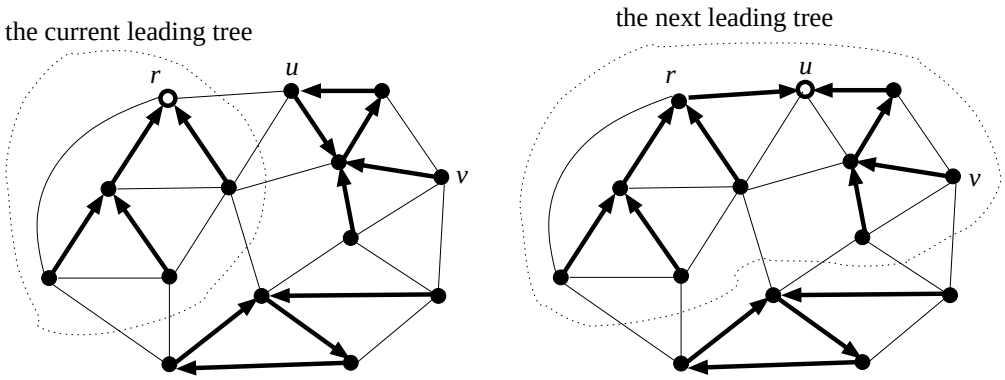

FIGURE 2.4: Left: the current step, when node v is outside the leading tree. Right: the next step, when the agent visits an ancestor u of v and v enters the leading tree.

ρ_v be an arbitrary cyclic order of the arcs in $\vec{G}$ outgoing from v. Lemma 2.25 assigns to the pair $(\vec{T}, (\rho_v)_{v\in V})$ the Eulerian cycle of $\vec{G}$ which is followed by the agent starting from the configuration $((\rho_v)_{v\in V}, \vec{T} \cup \{\pi_r\}, r)$, that is, starting with $\vec{T}$ as the leading tree. It is easy to verify that the cycles $\vec{\Gamma}'$ and $\vec{\Gamma}''$ assigned to two distinct pairs $(\vec{T}', (\rho'_v)_{v\in V})$ and $(\vec{T}'', (\rho''_v)_{v\in V})$ are distinct. Conversely, one can also show that each Eulerian cycle of graph $\vec{G}$ corresponds to some pair $(\vec{T}, (\rho_v)_{v\in V})$. This one-to-one correspondence between the Eulerian cycles in $\vec{G}$ and the pairs $(\vec{T}, (\rho_v)_{v\in V})$, where $\vec{T}$ is an in-bound spanning tree of $\vec{G}$ rooted at node r and ρ_v is a cyclic order of the arcs in $\vec{G}$ outgoing from v, gives a one-to-one correspondence between the Eulerian cycles in $\vec{G}$ and the pairs $(T, (\rho_v)_{v\in V})$, where T is a spanning tree in G: identify an in-bound spanning tree of $\vec{G}$ rooted at node r with the spanning tree of G obtained from $\vec{T}$ by disregarding the directions of arcs. For a node $v \in V$, there are $(d(v) - 1)!$ distinct cyclic orders ρ_v, so the number of Eulerian cycles in $\vec{G}$ is equal to $\prod_{v\in V}(d(v) - 1)!$ times the number of spanning trees of G, showing that Theorem 2.24 holds.

We now give bounds on the length of the additional stabilisation period after some failures or changes in the graph have occurred. To achieve this, we characterize the change to the leading tree after the change of the topology of the graph.

With respect to the set of port pointers $\vec{F}_r = \vec{F} \setminus \{\pi_r\}$, where r is the current node, a node v is an *ancestor* of a node u if, and only if, the path in $\vec{F}_r$ starting from v passes through u. If a node v belongs to the leading tree $\vec{T}$, then the ancestors of v are all the nodes on the path in $\vec{T}$ from v to the root r, including both v and r.

One can observe [T4] that if a node v belongs to the current leading tree, then it remains in the leading tree in all subsequent steps. On the other hand, if v is a node which is not in the current leading tree, then v enters the leading tree at the first step when the agent visits an ancestor of v, cf. Figure 2.4.

The above observations can be used to show the following theorem.

Theorem 2.26 ([T4])**.** *If the distance from a node v to the current node r is equal to k, then node v enters the leading tree within $2km$ steps.*

We now consider several model of faults or changes in the graph for the rotor-router system. First, a faulty change of the value of the port pointer π_v at a node v might occur when something unexpected makes the node believe that π_v should be re-set to some default value. Next, we assume that when a new edge $\{u, v\}$ is added, it is inserted in arbitrary places in the existing cyclic orders of edges adjacent to nodes u and v, but otherwise those cyclic orders remain as they were before. Similarly, when an edge $\{u, v\}$ is deleted, the cyclic orders of the remaining edges adjacent to nodes u and v remain as they were. On both addition and deletion of an edge $\{v, u\}$, we allow an arbitrary change of the value of the port pointers at nodes $\{v, u\}$. A concrete system would specify some default updates for the port pointers on insertion or deletion of an edge, but we do not want to make any assumptions about those defaults.

The behavior of the rotor-router in all of the above events, involving pointer changes, edge addition, or removal, can be easily characterized taking into account Theorem 2.26.

Theorem 2.27 ([T4])**.** *Suppose that a rotor-router operating in the graph has already locked-in into a traversal of an Eulerian cycle.*

(i) **Faults in port pointers.** *If at some step the values of k pointers π_v are changed to arbitrary edges (that is, the value of π_v is changed to an arbitrary edge adjacent to node v), then a new Eulerian cycle is established within $O(m \min\{k, D\})$ steps.*

(ii) **Addition of new edges.** *If at some step k edges are added to the graph, then a new Eulerian cycle is established within $O(m \min\{k, D\})$ steps.*

(iii) **Deletion of an edge.** *If at some step 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 smallest number of edges in a cycle in graph G containing the deleted edge.*

Regarding the $O(\gamma m)$ bound for the case of deleting an edge, we remark that there are non-trivial classes of graphs (e.g., random graphs) in which each edge belongs to a short cycle. For such graphs parameter γ is small and our bound implies that the additional stabilisation period is short.

The bounds which appear in Theorem 2.27 are all asymptotically tight in the worst case. Indeed, for some values of parameters s and d, the considered bounds may be obtained for the lollipop graph $G_{s,d}$, obtained by merging a vertex r of the clique K_s with an end-vertex of the path P_d (cf. Fig. 2.5a). Let the agent be located at vertex r after the stabilization of the rotor-router. When k port pointers are altered at internal nodes of path P_d ($k < d$), the rotor-router will only stabilize once more to an Eulerian cycle after visiting each of the edges of the clique at least k times. Hence, for any

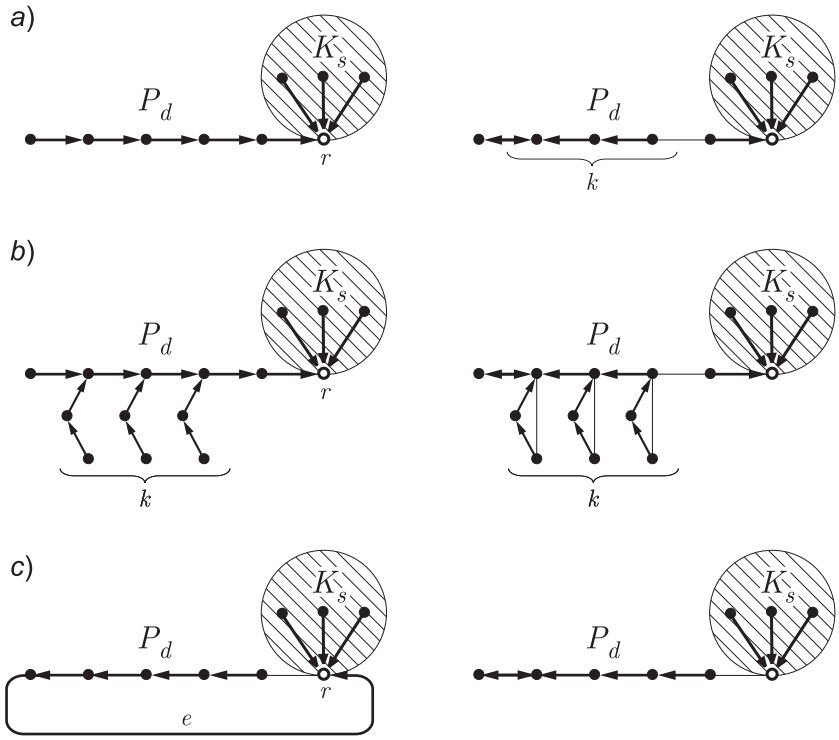

FIGURE 2.5: Worst-case examples for the stabilization period of the rotor-router after changes to the graph: (a) modification of k port pointers, (b) addition of k edges, (c) removal of a single edge e.

feasible set of parameters n, m, k, D there exists a graph of order $\Theta(n)$, having $\Theta(m)$ edges and diameter $\Theta(D)$, such that restoring the stable state of the rotor-router after a modification to k port pointers requires $\Omega(m \min\{k, D\})$ steps. Thus, the bound in Theorem 2.27 is asymptotically tight. Likewise, by the construction shown in Fig. 2.5b, we obtain a lower bound of $\Omega(m \min\{k, D\})$ steps for the stabilization period after adding k new edges to the graph, asymptotically matching the bound in Theorem 2.27. (Note that the addition of edges to the graph may by assumption result in modifications to pointer arrangements at the endpoints of added edges). Finally, Fig. 2.5c provides an example of a scenario in which removing a single edge leads to a stabilization period of $\Omega(\gamma m)$, asymptotically matching the bound in Theorem 2.27.

Applicability of the rotor-router in network exploration

The properties of the rotor-router are highly dependent on the amount of control over the mechanism, and especially the initial placement of the pointer. In all cases where the network designer is responsible for pointer assignment the complexity of the lock-in problem is either linear or close to linear. However, such an initialization of the mecha-

nism requires the application of an algorithm which is centralized, even if efficient.

A rotor-router with no centralized initialization also performs reasonably in comparison to the random walk. Assuming a worst-case port and pointer arrangement (i.e., in the $\mathcal{A}$-*all* scenario), for any graph, the lock-in time and the edge cover time of the rotor router are in the range $[\frac{1}{4}m\,D, 2m\,D]$, as characterized by Theorems 2.21 and 2.17, respectively. An edge cover time of $\Theta(m\,D)$ for the rotor-router compares interestingly to the expected edge cover time of a graph when using random walk, which can be bounded as $O(m\,D\log m)$ (Theorem 2.1). Whereas our bound for the rotor-router is tight for any graph, the bound for random walks is not. Indeed, a 2-dimensional grid on $\sqrt{n} \times \sqrt{n}$ has a worst-case edge cover time of $\Theta(n^{3/2})$ using the rotor router, and an expected edge cover time of $\Theta(n \log^2 n)$ using the random walk. At the other extreme, the rotor-router may have a lower edge cover time than that of the random walk. This is the case, for example, for the class of stars or the class of complete binary trees. The deterministic nature of the cover by the rotor-router is an important advantage with respect to the random walk.

We have also presented an asymptotically-tight evaluation of the robustness of the graph exploration based on the rotor-router mechanism. The analysis in this section can be applied to other possible models of faults and dynamic changes in the graph. For example, one may observe as a simple corollary of the analysis that the rotor-router mechanism tolerates spontaneous changes of the cyclic orders of the edges. More precisely, if at some step after the stabilisation period the cyclic orders of edges change in any way but the port pointers remain the same (that is, they point to the same edges as before), then the agent immediately enters a new Eulerian cycle (no need for any additional stabilisation period).

Further challenging questions arise with introduction of multiple agents to the rotor-router model. If we have many agents, then there are still interesting open questions left regarding the stabilisation and periodicity of exploration even in the static case (no faults, no dynamic changes of the graph). This question is given more attention in the last chapter of the monograph.

2.5.2 Equitable local exploration with edge counters

The exploration strategies studied in this section are designed so that the next edge in the agent's walk is chosen using only local information, and so that some local equity (fairness) criterion is satisfied for the adjacent undirected edges. Such strategies can be seen as an attempt to derandomize random walks, and are natural undirected counterparts of the rotor-router model, in which counters (or pointers) are associated with ports rather than edges.

The goal of achieving a traversal which visits all edges of the graph regularly gives rise to *locally equitable strategies*, i.e. strategies, in which at each step the robot chooses from among the adjacent edges the edge which is in some sense the "poorest", in an

effort to make the traversal fair. In this context, the following two natural notions of equity may be defined:

- An exploration is said to follow the *Oldest-First* (OF) strategy if it directs the robot to an unexplored neighboring edge, if one exists, and otherwise to the neighboring edge for which the most time has elapsed since its last traversal, i.e. the edge which has waited the longest.
- An exploration is said to follow the *Least-Used-First* (LUF) strategy if it directs the robot to a neighboring edge which has so far been visited by the robot the smallest number of times.

We remark that if the considered graph were to be *symmetric and directed*, and the above definitions were to be applied to directed edges (or equivalently, to the ports next to each node), then the Oldest-First notion of equity is precisely equivalent to the rotor-router model described in the previous chapter. Indeed, in the rotor-router, the next port to be traversed is always the one which has not been used for the longest time by the definition of the model. By a similar argument, one can conclude [175] that the directed Oldest-First strategy is strictly stronger than Least-Used-First, i.e. any exploration which follows the OF strategy is a special case of LUF strategy (note that LUF does not specify a tie-breaking mechanism for the case when several adjacent edges have been visited by the agent the same number of times). They also explore the whole graph quickly and visit all edges with the same frequency in expectation. The rotor-router stabilizes quickly to a Eulerian traversal, whereas the directed Least-Used-First exploration need not in general stabilize to such a traversal, but it retains the property that for any time moment, the number of visits to any two edges outgoing from the same vertex can differ by at most 1 [122, 123]. This property immediately implies that for symmetric directed graphs, any execution of Least-Used-First has a cover time of $O(m\,D)$, and also visits all directed edges with the same frequency.

In this section, we look at the Oldest-First and Least-Used-First strategies when applied to the *undirected* edges of a graph. For this case, the results, and the used techniques, turn out to be surprisingly different. We show that there exist graphs for which the undirected Oldest-First strategy *always* performs badly, whereas any execution of Least-Used-First achieves a cover time of $O(m\,D)$, and visits all edges with the same frequency.

The Oldest-First strategy

Whereas the rotor-router model leads to explorations which traverse directed edges with equal frequency, and have a cover time bounded by $O(m\,D)$, this is not the case for Oldest-First explorations in undirected graphs. Indeed, we show that in some classes of undirected graphs, any exploration which follows the Oldest-First strategy is unfair,

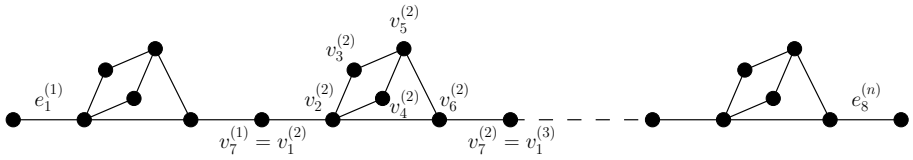

FIGURE 2.6: The graph G_n.

with an exponentially large ratio of visits between the most often and least often visited edges (Theorem 2.28).

Theorem 2.28 ([T5]). *There exists a family of graphs $(G_n)_{n\geq 1}$ of order $\Theta(n)$, such that for each graph G_n in this family, some two of its edges e and e' satisfy $\frac{f_e(G_n)}{f_{e'}(G_n)} = (\frac{3}{2})^n$ with $f_{e'}(G_n) \neq 0$, for any exploration following the* OF *strategy.*

A class of graphs for which OF performs badly is presented in Figure 2.6. It is composed of a sequence of 7-node graphs connected into a chain $v_j^{(k)}$, for any $j \in [1, 7]$ and $k \in [1, n]$. The proof relies on the observation that the edges of each subsequent subgraph in the chain are eventually visited a factor of $\frac{3}{2}$ times less often than those of its predecessor in the chain [T5].

From the above, it follows that there exist explorations following the Oldest-First strategy which have exponential cover time of $2^{\Omega(n)}$ in some graph classes.

Theorem 2.29 ([T5]). *There exists a family of graphs $(G_n)_{n\geq 1}$ of order $\Theta(n)$, such that for each graph G_n in this family, some exploration following the* OF *strategy has a cover time of $2^{\Omega(n)}$.*

The Least-Used-First strategy

In contrast to the case of directed graphs, in undirected graphs the Least-Used-First (LUF) strategy turns out to be fundamentally better than the Oldest-First strategy. In fact, we show that any exploration of an undirected graph which follows the Least-Used-First strategy is fair, achieving uniform distribution of visits to all edges.

Lemma 2.30 ([T5]). *For any moment of time t during the execution of* LUF *and any two edges e_1, e_2 which share a vertex, the number of times the agent traversed edges e_1 and e_2 until time t differs by at most 3.*

The above lemma can be extended to any pair of not necessarily adjacent edges of the graph, showing that the difference in the number of traversals between any pair of edges is at most $3(D+1)$. We conclude that in the limit, all edges of the graph are explored with the same frequency.

Theorem 2.31 ([T5]). *For any graph, any exploration following* LUF *achieves uniform frequency on all edges, $f_e(G) = 1/m$.*

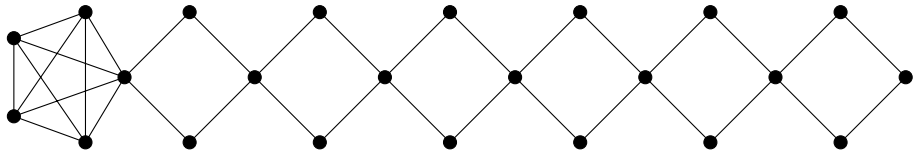

FIGURE 2.7: Worst-case example for LUF: a clique connected to a chain of 4-cycles.

Moreover, since there cannot exist an edge in the graph with a traversal count of more than $3(D+1)$ when some other edge has not been visited, it follows from Lemma 2.30 that any exploration of an undirected graph using LUF achieves a cover time of $O(m\,D)$. The bound given below follows from a slightly tighter analysis of the process.

Theorem 2.32 ([T5])**.** *For any graph, the cover time achieved by any* LUF *exploration is at most* $2m(D+1)$.

Moreover, this result can be extended to the case where some of the traversal counts initially take non-zero values, potentially due to faults of the system. Then, the cover time is bounded by $O((n+p)m)$, where p is the maximal value of a counter in the initial state [T5].

The cover time of the LUF strategy depends on how ties are broken when choosing among edges which have been traversed the same number of times. Indeed, for any undirected graph, there exists an exploration following the Least-Used-First strategy which is essentially the best possible, i.e., periodic with an exploration period of exactly $2m$ [T5]. On the other hand, one can also find graph classes for which certain initializations require a cover time which asymptotically matches the bound from Theorem 2.32.

Theorem 2.33 ([T5])**.** *For sufficiently large* n*, for any* $m \in [n-1, n(n-1)/2]$ *and* $D \leq n$*, the worst-case cover time of the* LUF *strategy in the family of graphs of at most* n *nodes, at most* m *edges, and diameter at most* D*, is* $\Omega(m\,D)$.

A worst-case example for LUF-exploration is shown in Figure 2.7. It consists of a chain of 4-node cycles, connected at one of the extremities of the chain a complete graph. The structure of this example resembles examples of worst-case instances for the cover time of the random walk in graphs.

Other measures of equity

We have shown that locally fair strategies in undirected graphs can mimic the properties of random walks, allowing us to obtain an exploration which is fair with respect to all edges, and efficient in terms of cover time. However, the fairness criterion for edges has to be chosen much more carefully than fairness for pointers: undirected Least-Used-First works well, but Oldest-First does not.

Strategies with local equity criteria have also been studied in the token circulation literature, in the context of strategies which are locally fair to vertices rather than edges. Two such strategies, named LF and LR, were proposed and analyzed in [141]. In the first of these, LF, the next vertex to be visited is always chosen as the least-often visited neighbor of the current vertex. In the second, LR, the next vertex to be visited is the neighbor which has not been visited for the longest time. The authors of [141] show that both of these strategies eventually visit all vertices, but in general do not satisfy any fairness criteria. Indeed, the time between successive visits to a vertex may be exponential in the order of the graph for LR, and unbounded for LF. In this sense, the results of [141] may be contrasted with our results for the LUF strategy.

In future work it would be interesting to study modified notions of equity, which are inspired by random walks which select the next edge to be traversed with non-uniform probability. For example, after biasing the probability distribution of the random walk to reflect the degrees of the nearest neighbors of the current node, Metropolis-type walks achieve a cover time of $O(n^2 \log n)$ steps in expectation. It is an open problem to define a reasonable local exploration strategy which could achieve such cover time in a deterministic sense.

3 Map construction in anonymous networks

The task of identifying the topology of the network graph is one which is not easier than graph exploration. Indeed, in order to be able to identify the structure of the network, the agent needs (in general) to explore all or almost all of its nodes. On the other hand, simply performing a sequence of moves which guarantees that all the the nodes (or edges) of the network are visited, need not provide the agent with sufficient information to identify the graph forming the network. When a bound on the number of nodes n of the network is known, the exploration task may always be performed in polynomial time by following a universal exploration table (Section 2.3). By contrast, it turns out that there exists scenarios in which an agent moving around an anonymous network can *never* uniquely identify the network it is operating in, even with knowledge of the number of nodes n and given unbounded time and unlimited computational resources. There exist examples of distinct anonymous graphs with distinguished starting nodes such that identical agents operating in these two graphs will always be located at indistinguishable nodes, and hence will always find themselves in the same memory state, i.e., unable to tell between the two graphs. This general-case impossibility result, first observed in [35, 172], is an indication of the difficulties observed in anonymous networks in comparison to scenarios in which all nodes have unique identifiers known to the agents. More broadly, efficient symmetry breaking is one of the fundamental areas of study in distributed computing with anonymous nodes. Starting with the seminal paper [19], various computational tasks for processors in anonymous networks have been studied in the literature (cf. e.g. [21, 34, 133]), including leader election or computing Boolean functions.

In this chapter we address the following question: How should an agent move around an unknown anonymous network, so as to obtain as much information as possible about its topology? Whereas we know that discovering the topology of the network completely may be impossible, the set of all topological information which the agent can discover in some sense defines the computational capabilities of the agent: a parameter or property of the network can only be computed by the agent if and only if it is a function of the "map" of the host network which the agent can construct.* The ability of the

*Cf. Section 1.3 for a discussion of various aspects of computability with mobile agents in an anonymous network, or [[46] Chapter 8] for a more formal characterization of the case of a single agent.

agent to construct such a partial map also forms the basis of algorithms for coordination problems, such as leader election or rendezvous, which we discuss in more details in Chapter 4.

As in the previous Chapter, we will focus on the complexity of map construction in terms of the required computational capabilities of the agent: its available state memory, number of required rounds, etc. We start by introducing in Section 3.1 the necessary concepts related to map construction in undirected anonymous graphs, namely those of the view and the quotient graph. (For a more complete mathematical background on the topic, including concepts of universal covers, port-preserving isomorphisms, and fibrations of directed graphs, we refer the interested reader to [35, 36].)

TABLE 3.1: Comparison of map construction algorithms in n-node anonymous networks of maximum degree d. The log-space algorithm marked with (*) is only capable of storing a single cell of the adjacency matrix of the quotient graph in the agent's memory, but may write the entire adjacency matrix to an auxiliary output tape.

Algorithm	*Memory*	*Time (steps)*	*Reference*
Identification with UXS	$O(nd \log n)$	$O(n^6 d^2 \log n)$	Sec. 3.2, [T6, T7]
Equivalence class refinement	$O(n^3 \log n)$	$O(n^5 d^2 \log^2 n)$	Sec. 3.3, [T6]
Log-space UXS comparison *	$O(\log n)$	polynomial	Sec. 3.4, [T6, T7]

The complexity of map construction in anonymous graphs is studied for a mobile agent that is not allowed to write on the nodes of the graph. Note that if the agent is allowed to somehow mark the nodes that it visits (such that it can recognize them on future visits), then a simple depth-first search suffices to solve the problem. When the agents do not have the capability to mark nodes it is sometimes difficult to solve the map construction problem.

3.1 Network maps: The view and the quotient graph

Consider a port-labeled anonymous network G, in which the agent starts from a home-base node v. We introduce the notion of a map as follows: We will call a mathematical object M a *map* for (G, v) under decoding function f, if there exists a function f such that for any deterministic agent A, $f(M, A, t)$ describes the memory state of A after t steps on G, when starting from node v. In other words, a map of (G, v) has to capture all the information about the structure of G which can be obtained during a traversal of the graph by an agent starting at a given node v. Whenever the decoding of the map

is clear from the context, we will simply say that the considered object M is a map, without specifying the decoding. By virtue of the definition, the pair (G, v) is a map of itself, but this trivial map is of little interest since it cannot be constructed by an agent deployed in the network G.

A fundamental construction of a map which encodes only information which is accessible to the agent was introduced in [172] under the name of the *view* from the node at which the agent is currently located. Let G be a graph and v a node of G, of degree k. The *view* from v is then the infinite directed rooted tree $\mathscr{V}(v)$ with labeled ports, defined recursively as follows. $\mathscr{V}(G, v)$ has the root x_0 corresponding to v. For every node v_i, $i = 1, \ldots, k$, adjacent to v, there is an out-neighbor x_i in $\mathscr{V}(G, v)$ such that the port number at v corresponding to edge $\{v, v_i\}$ is the same as the port number at x_0 corresponding to the arc (x_0, x_i), and the port number at v_i corresponding to edge $\{v, v_i\}$ is the same as the port number at x_i corresponding to arc (x_0, x_i). Each node x_i, for $i = 1, \ldots, k$, is now the root of the view from v_i. For any sequence of moves of the agent in graph G starting at v, there exists a directed path in the tree $\mathscr{V}(G, v)$, starting from its root and having a corresponding port labeling. The intuition that the view precisely encodes all the useful information that an agent may discover in the graph is captured by the following proposition.

Proposition 3.1. *The view $\mathscr{V}(G, v)$ is a map of (G, v). Conversely, given any map M under a known decoding function, the corresponding view $\mathscr{V}(G, v)$ is uniquely determined.*

From the perspective of computations, the view is a purely theoretical object, since it has infinite size. A more computationally-friendly map is obtained as follows. By $\mathscr{V}^t(G, v)$ we denote the view $\mathscr{V}(G, v)$ truncated to depth t from its root. One can ask, what is the minimum value of t such that for any pair of nodes v_1, v_2 in n-node graphs G_1, G_2, $\mathscr{V}(G_1, v_1) = \mathscr{V}(G_2, v_2)$, if and only if $\mathscr{V}^t(G_1, v_1) = \mathscr{V}^t(G_2, v_2)$? This value of t was first bounded as $t \leq n^2$ in [172], and then the precise worst-case bound of $t = n - 1$ was established by Norris [148].* More recently, another bound of the form $t \leq (D+1)(\log_2 n + 1)$ has been established by Hendrickx [105] for the case of graphs of diameter at most D.† Combining the state-of-the-art, we obtain the following theorem.

Theorem 3.2. *The truncated view $\mathscr{V}^t(G, v)$ is a map of (G, v), for any integer $t \geq \min\{n - 1, (D+1)(\log_2 n + 1)\}$.*

The truncated view $\mathscr{V}^t(G, v)$ is a computationally tangible resource and it can be constructed by an agent operating in the graph G, starting from node v. The agent simply

*The cited results on the equality of views are shown for the case of distinct starting nodes $v_1 \neq v_2$ in a single network $G = G_1 = G_2$, but their proofs hold without change for distinct graphs G_1, G_2.

†In current work in progress [68], we show this result to be asymptotically tight.

traverses all possible routes in the graph of length at most t, described by sequences of port numbers of length at most t, each time recording the corresponding arcs of the view and returning to the starting node by retracing its steps. To obtain a map of the graph in this way, the agent may need to traverse all routes of length $t = n - 1$, building a tree with $O(d^n)$ nodes in $O(d^n)$ steps, for a graph of maximum degree Δ and given a bound n on the order of the graph. It is natural to look for more compact and more efficiently constructible representations of the view.

As mentioned before, the view is not unique to a specific node of a specific anonymous graph: there can exist different anonymous graphs with vertices having the same view, and even multiple vertices in the same anonymous graph, having the same view. However, it turns out that for any pair (G, v), there exists a unique smallest graph H with a distinguished vertex v' which is indistinguishable from G in terms of the agent's views, i.e., $\mathscr{V}(G, v) = \mathscr{V}(H, v')$. Such a graph H is known as the quotient graph of G [172] and is defined by the following construction.

For a given connected graph G with a fixed port-labeling, we introduce the following equivalence relation: if two nodes have identical views, then these nodes are said to be equivalent to each other. In this way, we partition $V(G)$ into a set of equivalence classes $V_H = \{V_1, \ldots, V_l\}$. We define the *quotient graph* H of G as the port-labelled multigraph having node set V_H, with port i at node V_i leading to node V_j in H if and only if for some nodes $v_i \in V_i$, $v_j \in V_j$ port i at node v_i leads to node v_j in G. The mapping between graphs G and H is a port-preserving graph homomorphism. An illustration of the definition of views and quotient graphs is provided in Figure 3.1. It is interesting to observe that all the equivalence classes of nodes of G with respect to view have the same cardinality, i.e., $|V_i| = n/l$ [172].

Since for a vertex v_i belong to equivalence class V_i we have $\mathscr{V}(G, v_i) = \mathscr{V}(H, V_i)$, it follows that the quotient graph encodes all the information about the view.

Theorem 3.3. *The quotient graph of G, with a distinguished node corresponding to the equivalence class of v, is a map for (G, v).*

The quotient graph can be seen as the most compact form of a map of an anonymous network that the agent may construct. As we will see in the next section, the quotient graph of any anonymous network of known size can be computed in polynomial time, and the existence of particular edges of the quotient graph can even be tested for in logarithmic space.

3.2 Constructing a map using universal sequences

We recall that Universal Exploration Sequences (UXS-s), discussed in Section 2.3, allowed us to visit all the nodes of any anonymous network, up to a given bound n on

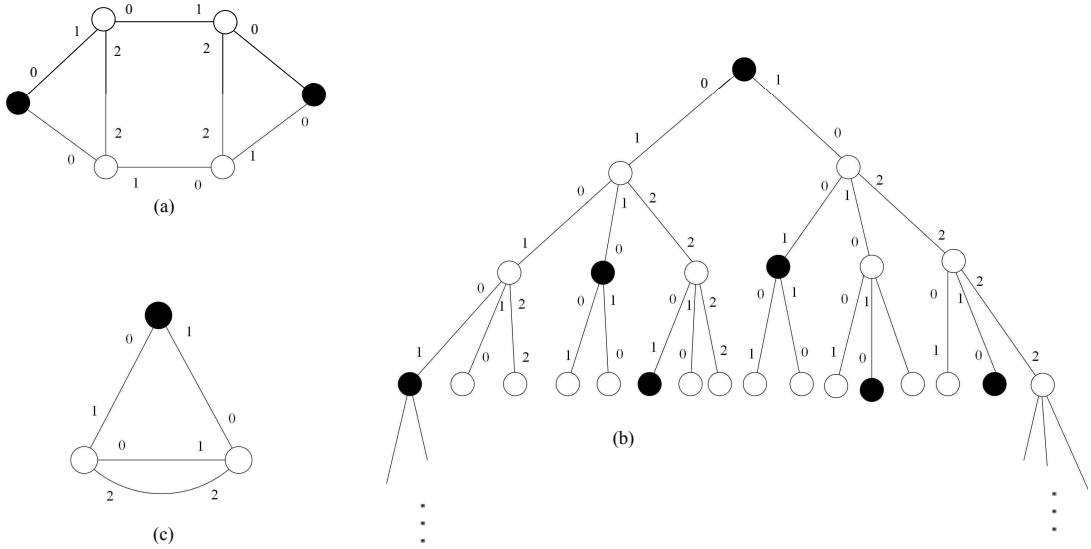

FIGURE 3.1: (a) An exemplary graph containing two agents initially located at the marked nodes. The view of each agent is shown in (b) while the quotient graph is shown in (c).

the number of nodes. Such universal sequences also allowed back-tracking, meaning that it was possible to come back to the starting location. In the course of traversing a (n, d)-UXS, the agent visited a sequence of nodes, observing the degrees of the visited nodes and the port numbers at the endpoints of the visited edges. It is tempting to ask whether the information collected by the agent, when traversing a UXS, is sufficient for it to compute the view $\mathscr{V}(G, v)$ from its homebase v?

Formally, let $Y = (a_1, \ldots, a_M)$ be a UXS, and let $(u_0, \ldots, u_M)$ be its application starting from a node $v = u_0$ in G. The *signature* S of node v is defined as the sequence of edge labels which are traversed by an application of the UXS at v: $S(G, v) = (lab(u_0, u_1), \ldots, lab(u_M, u_{M+1}))$.

The results in [T6, T7] provide a constructive criterion for distinguishing the views of two vertices of a graph G based on the signatures of vertices. It turns out that we can show that the view $\mathscr{V}(G, v)$ is completely encoded in the signature $S(G, v)$ if the traversed sequence is a UXS, but for graphs of larger order than G, having up to $n^2 + n$ nodes.

Theorem 3.4 ([T6,T7])**.** *Let n be a known upper bound on the number of nodes of graph G, let d be an upper bound on its degree, and let Y be a fixed $(n^2 + n, d)$-UXS. Then, the signature $S(G, v)$ obtained by applying sequence Y using a mobile agent starting from node v in G, is a map of (G, v).*

For the sake of intuition, we provide a sketch of the argument used in the proof of the above theorem. The proof proceeds by contradiction. Suppose that there exist two

pairs of graphs with distinguished starting nodes, (G_1, v_1) and (G_2, v_2), such that mobile agents applying the UXS Y to the respective graphs G_i starting from node v_i, $i = 1, 2$, obtain the same signatures: $S(G_1, v_1) = S(G_2, v_2)$, whereas $\mathscr{V}(G_1, v_1) \neq \mathscr{V}(G_2, v_2)$.

Let graph K be defined as follows: The set of nodes of K is $(V_1 \times V_2) \cup S$, where V_1 is the node set of G_1, V_2 is the node set of G_2, and the set S consists of n nodes $S = \{s_0, s_1, \ldots, s_{n-1}\}$ outside of $V_1 \times V_2$. The set of edges of K is defined as follows.

1. For every pair of couples $(u_1, u_2), (v_1, v_2)$ from $V_1 \times V_2$, such that there exist edges $\{u_1, v_1\}$ in G_1 and $\{u_2, v_2\}$ in G_2 having the same edge labels, $lab_{G_1}(u_1, v_1) = lab_{G_2}(u_2, v_2) = (i, j)$, there is an edge joining node (u_1, u_2) with (v_1, v_2) in K, with the same edge labels, i.e. $lab_K((u_1, u_2), (v_1, v_2)) = (i, j)$.

2. For every couple $(u_1, u_2) \in V_1 \times V_2$ and for every port $p < \max\{\deg_{G_1}(u_1), \deg_{G_2}(u_2)\}$, if either port p exists at exactly one of the nodes u_1, u_2, or if edge labels $(p, end_{G_1}(u_1, p))$ and $(p, end_{G_2}(u_2, p))$ are different, then there exists an edge $\{(u_1, u_2), s_p\}$ in K. For this edge we set port labels p at node (u_1, u_2) and k at node s_p, where k is the smallest integer not yet used for a port at s_p. Edges joining a node from $V_1 \times V_2$ with a node from S are called *special*. Observe that a special edge incident to node s_p is added in one of two cases:
   (a) when port p exists at exactly one of the nodes u_1 in graph G_1 or u_2 in graph G_2;
   (b) when port p exists at both nodes u_1 in graph G_1 and u_2 in graph G_2, but $end_{G_1}(u_1, p) \neq end_{G_2}(u_2, p)$.

An example of the construction is shown in Fig. 3.2. The proof is completed by observing that if $\mathscr{V}(G_1, v_1) \neq \mathscr{V}(G_2, v_2)$, then in graph K, node (v_1, v_2) must belong to the same connected component as some $s_i \in S$. Indeed, since the views $\mathscr{V}(G_1, v_1)$ and $\mathscr{V}(G_2, v_2)$ are different, then there must exist some sequence of ports whose application to (G_1, v_1) and (G_2, v_2) leads to different signatures; by applying this sequence to the graph K starting from node (v_1, v_2) we necessarily pass through some node $s_i \in S$. It follows that by applying the (n^2+n, d)-UXS Y to K starting from node (v_1, v_2) one also passes through node s_i, since graph K has at most $n^2 + n$ vertices and maximum degree at most d. However, this directly implies that an edge leading to s_i was traversed in the process, and so by the definition of graph K, the signatures $S(G_1, v_1)$ and $S(G_2, v_2)$ have to differ at the corresponding position, leading to a contradiction.

We now analyze the complexity of the proposed approach. Since the length of a $(n^2 + n, d)$-UXS is $O(n^6 d^2 \log n)$ (Section 2.3), and traversing each step of the UXS requires a single round, the time needed to compute the quotient graph is $O(n^6 d^2 \log n)$. Bounding the space requirement of the agent is a bit more subtle. A straightforward approach consists in traversing the UXS of length $O(n^6 d^2 \log n)$ and remembering each of the observed port numbers along the way; since each port number requires $O(\log n)$

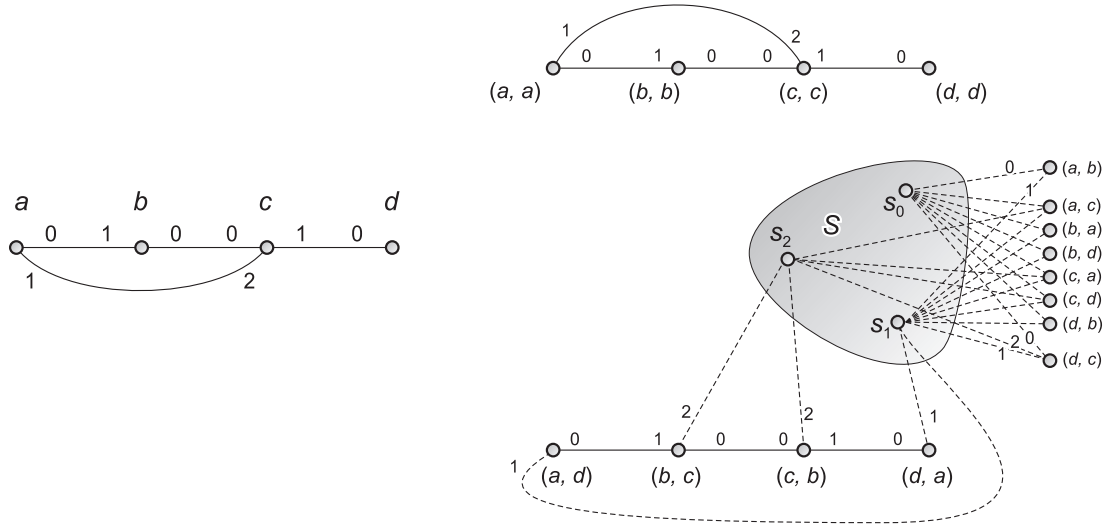

FIGURE 3.2: An example of a labeled graph G (left) and the corresponding graph K (right) for $G_1 = G_2 = G$. Port labels within set S have been left out.

space, we then obtain a total space complexity of $O(n^6 d^2 \log^2 n)$. However, this is somewhat wasteful: the number of graphs of maximum degree at most d and at most n nodes with a distinguished starting vertex can be bounded by n^{dn}; hence, only $O(dn \log n)$ bits of memory are required to identify the quotient graph and its starting node uniquely. Knowing a bound on n, the agent can at each step of its walk, construct a data structure in the form of a rooted search tree, having at most n^{dn} leaves representing unique quotient graphs. The distance from the root to each leaf is precisely equal to the length of the UXS traversed by the agent, and the i-th edge on the path leading from the root to a leaf H is labeled by the port number encountered by the agent in the i-th step of the UXS, given that the network it is operating in has a quotient graph isomorphic to H. The tree has $O(n^{dn} \cdot n^6 d^2 \log n)$ nodes, which can be lexicographically ordered and assigned identifiers in the form of consecutive integers with a binary representation of length $O(dn \log n)$. In this way, in stead of storing the whole sequence of port numbers encountered in its traversal, the agent only needs to memorize the identifier of the node of the tree at which it is currently located, using $O(dn \log n)$ space. In this way, we obtain the following theorem.

Theorem 3.5 ([T6])**.** *A mobile agent, starting at a node v of graph G on at most n nodes, equipped with $O(nd \log n)$ bits of state memory and knowledge of n, can determine a map of (G, v) in $O(n^6 d^2 \log n)$ time steps.*

We remark that the obtained map is encoded in the form of the signature $S(G, v)$, but this can then be used to compute e.g. the quotient graph of G. Such a computation can be performed locally in the final time round of the algorithm, through exhaustive search

over all multigraphs of order at most n for the quotient graph corresponding to G.

For some graphs of order n, the algorithm implied by Theorem 3.5 requires $\Omega(n^8)$ bits of memory. In this chapter we will provide two alternative approaches, which reduce the space of the requirement of the algorithm, while retaining polynomial running time.

3.3 Map construction by class refinement

In this section we now present an algorithm which allows the agent to solve the map construction more efficiently in terms of space, using $O(n^3 \log n)$ bits of memory. Our algorithm combines an application of a universal exploration table (Section 2.3) with ideas that are usually used to minimize a deterministic automaton. As before, we assume that the agent has prior knowledge of an upper bound on n, the number of nodes of the graph.

Theorem 3.2 implies that for any pair of nodes of the graph having distinct views, their views must differ at some depth $s < n$. If two trees differ at a certain depth s, there must exist a path of length s in the trees which starts at their roots and "distinguishes" the two trees. In order to characterise the views of all the nodes in the graph, we only need to find distinguishing paths for each pair of distinct views. Formally, given a graph G and node u of G and a sequence of edge-labels $Y = ((p_1, q_1), (p_2, q_2), \ldots, (p_j, q_j))$, we will say that Y is *accepted* from u if there exists a path $P = (u = u_0, u_1, \ldots, u_j)$ in G such that $\Lambda(P) = Y$, i.e. for each i, $1 \leq i \leq j$, $(p_i, q_i) = \lambda(u_{i-1}, u_i)$. For any $k > 0$, two vertices u, v that have the same view up to depth k are said to be k-equivalent; we denote it by $u \sim_k v$. The k-class of u is the set of all vertices that are k-equivalent to u and this set is denoted by $[u]_k$, with $[u] = [u]_\infty$. Given any two distinct k-classes C, C', a (C, C')-*distinguishing path* is a sequence of edge-labels $Y_{C,C'} = ((p_1, q_1), (p_2, q_2), \ldots, (p_j, q_j))$ of length at most k such that $Y_{C,C'}$ is accepted from each node $u \in C$ and it is not accepted from any node $v \in C'$. For any two distinct k-classes, there always exists either a (C, C')-distinguishing path or a (C', C)-distinguishing path.

To compute the quotient graph of G, it suffices to visit every node v of G and identify the s-class of v and each of its neighbors (recall that $s < n$). Once these equivalence classes are known, one can construct the quotient graph H as follows. The vertices of H are the equivalence classes, and there is an edge labelled by (p, q) from $[u]$ to $[v]$ in H if and only if, u has a neighbor $v' \in [v]$ such that $\lambda_u(u, v') = p$ and $\lambda_v(v', u) = q$.

An outline of the pseudocode of the map construction procedure is given in Algorithm 3. Algorithm 3 iterates over k, and for each k, explores the graph and identifies the k-classes of the visited nodes and their neighborhoods. For traversing all nodes of the graph, we use a universal exploration table $T(n, d)$ for graphs of up to n nodes and maximum degree d; recall that $|T(n, d)| = O(n^3 d \log^2 n)$ by Theorem 2.9. We choose the table T to be defined as an analogue of a UXS, i.e., to allow the agent to backtrack to its initial location after following T.

For $k = 1$, it is easy to determine the k-class of any node v by traversing each edge incident to v and noting the labels. From this information, one can find the distinguishing paths for any pair of 1-classes. For $k \geq 2$, it is possible to identify the k-classes and the corresponding distinguishing paths (from knowledge of the $k-1$ classes) using the following property.

Lemma 3.6 ([T6]). *For $k \geq 2$, two nodes u and v belong to the same k-class, if and only if (i) u and v belong to the same 1-class and (ii) for each i, $0 \leq i \leq deg_G(u) = deg_G(v)$, the ith neighbor u_i of u and the ith neighbor v_i of v belong to the same $(k-1)$-class and $\lambda(u, u_i) = \lambda(v, v_i) = (i, j)$, for some $j \geq 0$.*

Algorithm 3: Class-Refinement(n)

Let $v_1, v_2, \ldots v_t$ be the sequence of nodes visited by $T(n, d)$, possibly containing duplicate nodes ;
Apply $T(n, d)$ and **for** *each node v_i visited during the traversal* **do**
    Store the labels of each edge incident to v_i;
Compute the number of 1-classes and store a distinguishing path for each pair of distinct classes ;
Backtrack along $T(n, d)$ to the starting vertex ;
k := 2;
repeat
    Apply $T(n, d)$ and **for** *each node v_i visited during the traversal* **do**
        for *each edge (v_i, w) incident to v_i* **do**
            Compute the $(k-1)$-class of w (by testing the distinguishing paths);
            Store the label of (v_i, w) and the index of the $(k-1)$-class of w ;
        Discard the computed k-class information about v_i if duplicate of a previously visited vertex ;
    Backtrack along $T(n, d)$ to the starting vertex ;
    Compute the number of k-classes and store a distinguishing path for each pair of distinct k-classes ;
    Increment k;
until *the number of k-classes is equal to the number of $(k-1)$-classes*;
Compute the quotient graph ;

Let n_k be the number of k-classes. Observe that during the kth iteration, on each node v reached in $T(n, d)$, for each neighbor w of v, the agent computes the $k-1$ class of w. To do so, it needs to check at most n_k different paths of length $k-1$. Consequently, for each node v, it needs $O(\deg(v) \cdot n_k \cdot k)$ moves to compute the k-class of v. Thus, during the kth iteration of the algorithm, the agent performs $O(d \cdot n_k \cdot k \cdot |T(n, d)|)$

moves, where d is the maximum degree of the graph. Due to Theorem 3.2 there are at most $s < n$ iterations, and $n_k \leq n$; so the total number of moves made by the agent is $O(dn|T(n,d)| \cdot \min\{n, D \log n\})$, where $|T(n,d)| = O(n^3 d \log^2 n)$.

At the end of the kth iteration, the agent needs to remember the number n_k of k-classes and $n_k(n_k-1)/2$ distinguishing paths, each of length at most $k \leq s$. This can be stored using $O(n^2 \min\{n, D \log n\} \log n)$ bits. During the kth iteration, the agent needs to remember for each v and for each neighbor w of v, the label of the edge (v,w) and the index of the $(k-1)$-class of w. For each v, it needs $O(\deg(v) \cdot \log n)$ bits. However, the agent does not need to remember the k-class of each v_i, but it is sufficient to identify the distinct k-classes that exist in the graph. Thus, since there are at most n different k-classes, the agent needs $O(n \cdot d \cdot \log n)$ bits of memory to compute the number of k-classes, and to compute the corresponding distinguishing paths using the distinguishing paths for the $(k-1)$-classes. We obtain the following Theorem.

Theorem 3.7 ([T6]). *Algorithm 3 solves map construction for any graph of at most n nodes, where n is known to the agent in $O(n^5 d^2 \log^2 n)$ moves using $O(n^3 \log n)$ bits of memory.*

We remark that by Theorem 3.2 it is also possible to bound the depth s required to distinguish views as $s = O(D \log n)$. Consequently, we obtain that Algorithm 3 solves map construction in $O(n^4 d^2\, D \log^3 n)$ moves using $O(n^2\, D \log^2 n)$ bits of memory.

3.4 Computations on the Quotient Graph with a Log-Space Agent

A natural question concerns the feasibility of performing computations when the memory size of the agent is constrained. We showed in the previous sections that a map of the graph can be constructed using an agent equipped with a polynomial number of bits of state memory. In other words, it turns out that any computation which can be performed deterministically by a mobile agent can also be performed by a mobile agent capable of transmitting a polynomial number of state bits along edges. Surprisingly, it turns out that even an agent equipped with a logarithmic number of bits of state memory is sufficient to recognize and navigate a quotient graph. When designing such an agent, we need to overcome the obvious obstacle, namely, that logarithmic space is in general insufficient for the agent to "memorize" the topology of the quotient graph. So, our goal will instead be to allow for the agent to be able to answer in logarithmic space basic queries concerning, in particular, the existence and port numbers of edges connecting particular nodes of the quotient graph. To allow for easier formulation of such queries, we will assume that the agent makes use of some canonical labeling of the nodes of the quotient graph, i.e., for a graph G with equivalence classes $\{V_1, \ldots, V_l\}$, we will

consider an isomorphic copy of its quotient graph H in which the vertices are labeled with consecutive integers as $V_H = \{1, \ldots, l\}$. The assignment of integer labels to the vertices of graph H can be chosen by the agent.

We will say that a mobile agent *resolves a query* if it runs a subroutine at the end of which it returns to the node at which the query was initiated, with a correct reply to the query encoded in a predefined area of its state memory. The results of [T7] imply the following theorem.

Theorem 3.8. [T7] *For any given n, there exist agents with $O(\log n)$ state memory which, when deployed in an anonymous graph G of at most n nodes, resolve the following queries:*

- *What is the number l of nodes of the quotient graph H of G?*
- *What is the identifier (in the range $\{1, \ldots, l\}$) of the node of the quotient graph H which corresponds to the equivalence class of the node of G at which the agent is currently located?*
- *Given two nodes of the quotient graph H with identifiers $i, j \in \{1, \ldots, l\}$, are they connected by an edge, and if so, what are the port numbers at its endpoints?*

The algorithmic approach which lies at the heart of Theorem 3.8 follows from an adaptation of the UXS-based method proposed in Section 3.2. However, before we provide an intuition of the arguments used in the proof, we remark that the capabilities of a log-space agent described in Theorem 3.8 effectively allow the agent to perform any computation on the quotient graph which could be performed in log-space in a centralized model with Random Access Memory, i.e., for any problem which belongs to the complexity class known as L (or LSPACE) [20].

Corollary 3.9. *For any problem P $\in$ L which takes a graph as input, there exists an agent with $O(\log n)$ state memory, such that for any (connected) anonymous graph G on at most n nodes, where n is known to the agent, the agent resolves the query: "What is the answer to problem P for the quotient graph of the network?".*

Many log-space testable properties of the quotient graph provide useful information about the anonymous network G. For example, one may attempt to design a mobile agent in network G to solve the problem: "Determine if graph G is a tree and, if not, find an edge which can be removed without disconnecting the graph". This question is equivalent to finding an edge belongs to some cycle in G, or determining if one does not exist. Graph G is a tree if and only if its quotient graph is either a tree without self-loops or a tree with a single self-loop, depending on whether the tree is symmetric with respect to its central edge. Moreover, by the properties of graph homomorphisms, if for some edge $\{u, v\}$ of G, its counterpart in the quotient graph H belongs to some cycle

in H (or is a self-loop in H), then $\{u, v\}$ belongs to a cycle in G. Both the problem of determining if a graph is a tree, and the problem of determining if a given edge belongs to a cycle, are known to belong to the complexity class L [13, 156]. It follows from the Corollary that the posed question about the existence of a non-disconnecting edge can be resolved by a log-space mobile agent, given that the agent is given an upper bound on the number of nodes n of the network.

In all of the above considerations, knowledge of an upper bound n by the agent is in general indispensable, since otherwise the agent is unable to construct the quotient graph. Without knowledge of n, the agent would require some additional advice about the environment it is operating in. For example, when the agent knows that the network G in which it is operating is a tree, it is possible for the agent to compute the precise number of nodes of the tree. This can even be achieved in $O(\log n)$ space using an approach proposed by [15]. The same goal can be achieved by testing increasing bounds on n in Theorem 3.8 and traversing the resulting candidate for the quotient graph, repeating the process until a candidate is found which is consistent with the explored tree network.

We return to the sketch of proof of Theorem 3.8 from [T7]. Consider an application of the same identifying UXS at two different nodes u, v of the anonymous network G, giving sequences of port labels $S(u) = (lab(u_0, u_1), \ldots, lab(u_M, u_{M+1}))$ and $S(v) = (lab(v_0, v_1), \ldots, lab(v_M, v_{M+1}))$. (By contrast to Section 3.2, here we only consider a single network G, so for compactness we use the notation $S(v) \equiv S(G, v)$.) By Theorem 3.4, two nodes have distinct views $\mathscr{V}(v) \neq \mathscr{V}(w)$, if and only if $S(v) \neq S(w)$. Thus, the sequence $S(u)$ can be treated as a compact representation of the view $\mathscr{V}(u)$. We can now consider a linear order on sequences S: we say that $S(u) \prec S(v)$, if there exists $t \leq M$, such that for each $j < t$ we have $lab(u_j, u_{j+1}) = lab(v_j, v_{j+1})$ and $lab(u_t, u_{t+1}) \prec lab(v_t, v_{t+1})$. In the partition of the set of nodes $V = V_1 \cup \ldots \cup V_l$ into equivalence classes such that all nodes in V_i have the same view, we will consider an indexing of the classes for which V_i is the set of nodes v_i, such that $S(v_i)$ is the i-th smallest (distinct) element in the set $\{S(v) : v \in V\}$ in the defined linear order. It remains to be shown how, given a node v_i at which the agent is currently located, the corresponding label i can be computed by a logarithmic-space agent.

Let $Y = (a_1, \ldots, a_N)$ be the UXS $Y(n^2+n)$ of length N for the class $\mathcal{G}_{n^2+n}$ of graphs with at most $n^2 + n$ nodes. Fix integer parameters i and j. The aim of the auxiliary function `CompareSignature`(i, j) is to lexicographically compare the signatures S of the i-th and the j-th nodes of the application of Y at the starting node of the agent. This comparison is done term by term, for $k = 1, \ldots, N$ until a difference is found. The function `CompareSignature` uses two procedures: `TravFwd` and `TravBack`. Procedure `TravFwd`(k) traverses k consecutive nodes of the application of Y at the node u_0 at which it is called in the graph G. Node u_1 is found as $succ(u_0, 0)$. In order to find node u_{i+1} for $i \geq 1$, of this application, the term a_i is first computed

and then the formula $u_{i+1} = succ(u_i, (p + a_i) \mod d(u_i))$ is used, where p is the port at node u_i used to enter this node. It is supposed that the edge label of the last traversed edge is always kept in memory. Hence the second component of the edge label — the port identified while the current node u was being entered — is available until it is used to enter the successor of u. Port 0 is used to start the walk at the node at which Y is applied. Procedure `TravBack`(k) performs the backward walk along an application of Y, starting at some node v of it and using the formula $succ(u, (p - a_i) \mod d(u))$ to determine where to backtrack from u. To start the walk back from v, the port by which node v was entered is used first. In the case of `TravBack`, elements $a_k, a_{k-1}, \ldots, a_1$ of Y are used in the order of decreasing indices. In each iteration k of the loop within function `CompareSignature`, the agent is guided through the graph G until the corresponding edge labels of both signatures are retrieved with the aid of function `EdgeLabel`, and the initial position of the agent is regained. By analysing the pseudocode, we obtain the following lemma.

```
int function CompareSignature(int i, int j)    {0 ≤ i, j ≤ N}
1. for k ← 1 until N do
2.     el1 ← EdgeLabel(i, k) ; el2 ← EdgeLabel(j, k) ;
3.     if el1 < el2 then return smaller ;
4.     if el1 > el2 and return greater ;
5. return equal ;

edge label function EdgeLabel(int i, int k)
1.     TravFwd(i) ;
2.     TravFwd(k) ;
3.     EdgeLab ← the edge label of the last traversed edge ;
4.     TravBack(k) ;
5.     TravBack(i) ;
6.     return EdgeLab ;
```

Lemma 3.10. *Let u_i and u_j be the i-th and the j-th node of the application of Y at the starting node of the agent. Function* `CompareSignature`(i, j) *compares signatures $S(u_i)$ and $S(u_j)$ according to the $\preceq$ ordering and returns the value "smaller" if $S(u_i) \prec S(u_j)$, "equal" if $S(u_i) = S(u_j)$, and "greater" otherwise. An agent with $O(\log n)$ bits of memory is sufficient to execute the function.*

Using the above subroutine, we design the function `QuotientNodeLabel`which computes a canonical label of the node of the quotient graph corresponding to the agent's current location. The idea of the labeling is to use the function comparing two signatures in order to compute the number of different signatures that are not greater than the signature of the initial position of the agent. This number, belonging to the range $[1, n]$, is output by the function.

int function `QuotientNodeLabel`
{ returns the label of the node of the quotient graph corresponding to the agent's position }
1. $CurId \leftarrow 0$;
2. **for** $i \leftarrow 0$ **until** N **do**
3. **if** `CompareSignature`$(i, CurId)$ = smaller **then** $CurId \leftarrow i$;
4. **for** $j \leftarrow 1$ **until** n **do**
5. **if** `CompareSignature`$(CurId, 0) =$ equal **then return** j ;
6. $NextId \leftarrow 0$;
7. **for** $i \leftarrow 0$ **until** N **do**
8. **if** `CompareSignature`$(i, NextId) =$ smaller **and**
`CompareSignature`$(i, CurId) =$ greater
then $NextId \leftarrow i$;
9. $CurId \leftarrow NextId$;

Lemma 3.11. *An agent located at any node u of a graph with at most n nodes, computes the integer value of the function* $\texttt{QuotientNodeLabel}_u$*, belonging to the interval $[1, n]$, such that for any two nodes $u, v \in V$, the following equivalence holds:*

$$\texttt{QuotientNodeLabel}_u = \texttt{QuotientNodeLabel}_v \iff \mathscr{V}(u) = \mathscr{V}(v).$$

Moreover, an agent with $O(\log n)$ bits of memory is capable of executing the function `QuotientNodeLabel`.

Note that when calling function `QuotientNodeLabel`, the agent is returned to the node at which the function was called. The function can therefore be used as a transparent subroutine to learn the label of the current node in the quotient graph. The agent can move to a node corresponding to label i in the quotient graph by applying a UXS from an arbitrary starting location, and testing the value of the label for successive nodes until the condition $\texttt{QuotientNodeLabel} = i$ is satisfied. It can also test the existence of an edge $\{i, j\}$ in the quotient graph by finding a node satisfying $\texttt{QuotientNodeLabel} = i$, and checking whether any of the neighbors of this node in G satisfies $\texttt{QuotientNodeLabel} = j$. If such an edge exists, the port labels at its endpoints are also immediately recovered. In this way, one can implement all of the basic operations on the quotient graph with a logarithmic-space agent, as required in the claim of Theorem 3.8.

The time complexity of the designed log-space procedures operating on the quotient graph is necessarily polynomial, but with a high exponent, which can be roughly estimated as $O(n^{20})$. It is also possible to impose the requirement that not only the persistent state memory of the agent is of logarithmic size, but the same also holds for the memory used by the agent in local computations. Then, the UXS-s applied by the agent needs to be log-space constructible (following a variant of the approach from [156]), resulting in an even higher time complexity, on the order of n to he power of several hundred.

3.5 Remarks on other models

The algorithms considered in this chapter can be applied by mobile agents operating in an anonymous network, without the need for leaving any marker information at nodes of the network. The only assumption we make is that the agent knows some upper bound on the number of nodes n of the network. Without *a priori* knowledge of such an upper bound, the agent can attempt to apply a doubling technique, assuming increasing upper bounds on the number of nodes of the graph. Such an algorithm, however, can never terminate, since the agent may never decide if the map it has computed corresponds precisely to the real quotient graph of the host network.

Interestingly, it is possible to solve the mapping problem deterministically and with termination without knowledge of any global parameters, such as an upper bound on n, provided that the homebase of the agent is specially marked, and uniquely distinguishable from all other nodes of the anonymous network. Such an assumption changes the nature of the problem, since the abstraction class of the homebase in the quotient graph consists of precisely one element, and so the quotient graph is isomorphic to the host network G. In this scenario, even without knowledge of n, it is possible to solve the mapping problem using a log-space agent, or more quickly, in $O(n^3 d)$ time steps using $O(nd \log n)$ bits of memory [T6].

The task of map construction becomes simpler and admits faster solutions as we increase the capabilities of the agent, allowing it to leave pebbles at nodes or write to whiteboards. An interesting twist of the mapping problem is considered in [64], where multiple identical agents operating in the same network attempt to solve the problem simultaneously, and cannot distinguish between marks left in the graph by themselves and by other agents. This type of study fits more closely into the framework of simultaneous symmetry-breaking problems for multiple agents, which we consider in the following chapter.

4 The rendezvous problem

In the previous Chapters, we have considered questions of computability in an anonymous network, given a single mobile agent with limited resources. The introduction of multiple agents into the same network opens completely new research questions. The agents may be required to perform specific tasks with respect to the graph, such as exploration or computation of topological properties, but also to solve coordination problems for which the initial configuration of the agents can be seen as part of the input.

Whereas in Chapter 3 we consider the limits of computability with a single agent, the case of multiple agents turns out to be much more complicated. The answer depends, in particular, on whether agents move in synchronous rounds, or whether each traversal of an edge may take an arbitrary period of time. Another question concerns the ability of agents to interact with each other in a graph. Given that the agents are unaware of each others' presence, the computational power of each agent is no different from that in the single-agent case. Still, even then certain tasks may be performed more efficiently: if the goal of the agents is to minimize the expected time before each node is explored by some agent, then it makes sense to deploy multiple agents following a random walk, and not just one. In this chapter, we will assume that the agents have the ability to exchange information only when they are located at the same node at the same time. Arguably, this can be considered the most powerful model of communication which does not require non-local interaction or information storage in the environment. More powerful models, in which the agents may either communicate at a distance, or leave information at for another agent arriving later by writing on a whiteboard [87, 132] or placing a marker at a node [176], are beyond the scope of this thesis.

Since meeting is the only way for agents to coordinate their actions, in the context of this work, the most fundamental problem for multiple agents is that of rendezvous: two identical mobile agents, initially located in two nodes of the network, move along links from node to node, and their goal is to get to the same node at the same time. The rendezvous problem has been thoroughly studied in the literature in different contexts (see the monograph [12] for an extensive bibliography of the subject), both under the randomized and the deterministic scenarios. In a general setting, the rendezvous problem was first mentioned in [161]. Authors investigating rendezvous (cf.[12] for an extensive survey) considered either the geometric scenario (rendezvous in an interval of the real line, see, e.g., [27, 28, 96], or in the plane, see, e.g., [16, 17]) or the graph scenario (see, e.g., [69, 94, 129]). Many papers, e.g., [10, 11, 18, 27, 110] study the probabilistic setting: inputs and/or rendezvous strategies are random. A natural extension of the rendezvous

problem is that of gathering [89, 110, 137, 165], when more than 2 agents have to meet in one location.

In the anonymous graph model, a fundamental problem when studying feasibility and efficiency of deterministic rendezvous is that of breaking symmetry. Without resorting to marking nodes, this can be achieved by taking advantage of the different labels of agents [69, 129, 142]. Labeled agents allowed to mark nodes using whiteboards were considered in [176]. Recently, rendezvous of labeled agents using variants of Universal Exploration Sequences, was also investigated in [164].

We start this chapter by providing criteria for describing the feasibility of deterministic rendezvous for two agents (labeled or unlabeled), operating in a synchronous setting. In Section 4.1 we show that if rendezvous is feasible, then it can be achieved even when using logarithmic-space agents. We also consider the tradeoff between the amount of memory available to the agent, and the time required for rendezvous. Whereas establishing a tight time-space tradeoff for the task of exploration is a challenging open question, such an interdependence turns out to be much more apparent for the rendezvous problem. In Section 4.2, we establish the precise form of the time-space tradeoff for rendezvous of anonymous agents in trees. We close our considerations by discussing extensions of feasibility criteria for rendezvous to the case of asynchronous agents, and to teams of more than two agents.

4.1 Synchronous rendezvous in log-space

When two agents placed in a graph are initially assigned distinct labels, encoded in their starting state, this information is sufficient for the agents to achieve rendezvous. In fact, Dessmark *et al.* [69] provided a deterministic algorithm which relies on repeated traversal of Universal Exploration Sequences, in a way modulated by the label of the agent, to achieve rendezvous in polynomial time. It is assumed that the system operates in synchronous rounds, but that one of the agents may be delayed and may only appear in the graph after some period of time τ, and that the time required for rendezvous is only counted from the moment of appearance of the later agent. The runtime of their algorithm is given as $\tilde{O}(n^5\sqrt{\tau l} + n^{10}l)$, where l represents the bit length of the shorter of the agents' labels. Subsequently, Kowalski and Malinowski [129] and Ta-Shma and Zwick [164] designed algorithms with runtime which is independent of the delay τ, namely, $\tilde{O}(n^{15} + l^3)$ and $\tilde{O}(n^3d^2l)$, respectively, where d is the maximum degree of the graph. Interestingly, all of these algorithms operate without knowledge of an upper bound on n. By contrast to the exploration problem, in which an agent can never detect if it has successfully explored a graph of unknown size, in the rendezvous problem, the termination condition is achieved very simply by observing the meeting with the other agent operating in the graph. The approach of Ta-Shma and Zwick also works for log-space agents, or more formally, with agents capable of storing their label and $O(\log n)$

bits of auxiliary state.

The situation becomes more complex when the agents are not guaranteed to have unique identifiers. Then, rendezvous is not always feasible. Indeed, supposing that two agents occupy nodes having the same view (i.e., nodes corresponding to the same node of the quotient graph), any sequence of moves by the two agents will always leave the agents at a pair of nodes having the same view.

Proposition 4.1 ([T7])**.** *Suppose that a pair of anonymous agents are placed originally at distinct nodes $u_1, u_2 \in V$ and start walking simultaneously in G. If rendezvous is feasible then $\mathscr{V}(u_1) \neq \mathscr{V}(u_2)$.*

It turns out that the converse of the above lemma is true, i.e., given that $\mathscr{V}(u_1) \neq \mathscr{V}(u_2)$, it is always possible to achieve rendezvous. Moreover, such rendezvous can be achieved in polynomial time, and even using agents equipped with logarithmic space. The idea of the approach is the following. First, the two agents follow the procedure from Section 3.4 to compute the identifier in the quotient graph of the node representing their starting position. Since $\mathscr{V}(u_1) \neq \mathscr{V}(u_2)$, this procedure will compute distinct identifiers from the range $\{1, \ldots, n\}$ for nodes u_1 and u_2, which we will denote by L_1 and L_2, respectively. Now, the identifiers L_1 and L_2 can be used as distinct labels by the agents, with the representation of the shorter label having length $l = \log \min\{L_1, L_2\} = O(\log n)$. After that, to allow the agents to meet it suffices to execute a rendezvous procedure for labeled agents with arbitrary delay τ, using the rendezvous algorithm of Ta-Shma and Zwick [164]. The details of this process are also laid out in [T7]. The procedure can be adapted so as to operate without explicit knowledge of the value of n.

Theorem 4.2 ([T7])**.** *There exists a pair of anonymous agents with $O(\log n)$ bits of memory which solves the rendezvous problem, with arbitrary delay, for any pair of starting nodes $u_1, u_2 \in V$ with distinct views ($\mathscr{V}(u_1) \neq \mathscr{V}(u_2)$).*

Proposition 4.1 combined with Theorem 4.2 can be seen as a characterization of the feasibility of solving the rendezvous problem, and of the computational capabilities of an agent required to achieve such a solution.

It turns out that logarithmic space is necessary to achieve rendezvous even in relatively restricted graph classes. This is the case, for instance, when the considered graph is a ring, i.e., there is a lower bound of $\Omega(\log n)$ on the number of memory bits required for rendezvous in the class of rings with at most n nodes.

The idea of the proof of the lower bound presented in [T7] is the following. We suppose that agents have memory of size $m \leq \frac{1}{5} \log n$ bits, hence they are modeled as an automaton with $2^m \leq n^{\frac{1}{5}}$ states. We show that there exists a ring of at most n nodes and non symmetric initial positions of the agents, such that the agents cannot meet. Hence agents with such a small memory cannot solve the rendezvous problem for the class of rings with at most n nodes. We will construct a ring formed by the concatenation

of port-labeled segments of $n^{\frac{2}{5}}$ nodes each. We take into account the states of the agent at the moment when it is entering and exiting such segments through their endpoints. We observe by a counting argument that it is possible to choose two different segments T_1, T_2, each one having $n^{\frac{2}{5}}$ nodes, such that the agent entering a segment in any state s_i will exit it in some state s_j independently of whether the visited segment is T_1 or T_2. The sequence of the agent's states considered during visits to the endpoints of the segments must form a cycle of some size $d_c \leq n^{\frac{1}{5}}$, called its *state cycle*.

We first construct a ring, which prevents rendezvous, unless the agent advances by a constant distance around the ring at each iteration of its state cycle. Such a ring can be constructed because the agent's movement must then be localized to a bounded portion of the ring and both agents may be directed to operate in disjoint portions. Then another ring is constructed to prevent rendezvous also for such positively-advancing agents. This ring is formed of $d_c n^{\frac{2}{5}}$ segments of size $n^{\frac{2}{5}}$ each (hence of total size at most n) containing exactly one copy of T_2 and $d_c n^{\frac{2}{5}} - 1$ copies of T_1. We place the agents at antipodal positions of the ring and prove that each agent must indefinitely, cyclically return to some position of the ring in the same state and time which are the same for both agents. Finally, it is proven that the variations of the distance between the agents during such tours around the ring are too small to allow their meeting.

Theorem 4.3 ([T7]). *For any $n \geq 9^5$, any pair of anonymous agents solving rendezvous in rings with at most n nodes requires at least $(\frac{1}{5}\log n - 1)$ bits of memory (even assuming simultaneous start).*

The above result shows the intrinsic difference in space complexity between the rendezvous and exploration problems, since perpetual exploration of rings can be achieved even by a memoryless agent which always exits a node by a port different from its port of entry.

4.2 Time-space trade-off for rendezvous in trees

In this section we focus attention on deterministic rendezvous in trees and our goal is to establish trade-offs between the optimal time of completing rendezvous and the size of memory of the agents. We recall from the previous section that rendezvous with simultaneous start is impossible if the initial positions of the two agents have identical views. Hence, all algorithmic considerations are performed under the assumption that the initial positions of agents are not symmetric.

Rendezvous time (both deterministic and randomized) of anonymous agents in trees without marking nodes has been studied in [82]. It was shown that deterministic rendezvous in n-node trees can be always achieved in time $O(n)$. Memory required by the agents to achieve deterministic rendezvous in trees has also been studied in [94, 95], though the model adopted in the latter paper assumes restrictions on the topology of the

considered trees. In [23] the authors study the memory size needed to solve a variant of the rendezvous problem in trees in optimal time in the asynchronous model. They do not allow rendezvous inside an edge, but for symmetric trees they allow that agents terminate not in one node but in two adjacent nodes. They show that the minimum number of memory bits to achieve rendezvous in linear time is $\Theta(n)$.

The main result of this section is a tight trade-off between optimal time of completing rendezvous and the size of memory of the agents [T8]. For agents with k memory bits, we show that optimal rendezvous time is $\Theta(n+n^2/k)$ in n-node trees. More precisely, if $k \geq c \log n$, for some constant c, we show agents accomplishing rendezvous in arbitrary trees of unknown size n in time $O(n + n^2/k)$, starting with arbitrary delay.

We will describe only a sketch of the approach from [T8], for the case of trees of maximum degree bounded by 3 and under the assumption that agents know a bound N on the number of nodes of the tree, such that $N \geq n \geq N/16$. The overview of the algorithm is the following. In the first phase, whose time is $O(n + n^2/k)$, each agent computes an integer value $l \in \{0, 1, \ldots, n-1\}$ called its *signature*, such that agents with non-symmetric initial positions have different signatures. These signatures are used in the second phase to break symmetry and achieve rendezvous. The way in which this is done depends on the amount of memory available to the agents. If the agents have large memory (at least $\Omega(n/\log n)$ memory bits), then they can quickly locate either the central node or the central edge of a specifically chosen subtree of the tree in which they operate. In the first case they meet at its central node, in the second case they use the signatures to meet at one of the endpoints of its central edge. In the case of small memory ($o(n/\log n)$ memory bits), each agent uses a sequence of active and passive periods, each of length $4(N-1)$, determined by the successive bits of its signature: in an active period (bit 1 of the signature) an agent visits all nodes of the tree, in a passive period (bit 0 of the signature) it waits. This guarantees rendezvous in additional time at most $O(n \log n)$ which is dominated by $O(n^2/k)$, for small memory.

To compute its signature, the agent performs a traversal of the tree using the basic walk procedure considered in Section 2.4. We recall that a *basic walk* in a n-node tree T, starting from node v, is a traversal of all edges of the tree ending at the starting node v and defined as follows. Node v is left by port 0; whenever the walk enters a node by port i, it leaves it by port $(i+1)$ mod d, where d is the degree of the node. We sometimes consider more than $2(n-1)$ steps of a basic walk, noting that this traversal is periodic with a period of length $2(n-1)$. The basic walk starting at a node v may be uniquely coded by the sequence (string of symbols) $BW(v) = (p_1(v), q_1(v), p_2(v), q_2(v), \ldots, p_{2(n-1)}(v), q_{2(n-1)}(v))$, where $p_1(v) = 0$, $p_i(v)$ is the port number by which the node is left in the i-th step of the walk, and $q_i(v)$ is the port number by which the node is entered in the i-th step of the walk. A pair of nodes v_1 and v_2 of a tree T is not symmetric if and only if $BW(v_1) \neq BW(v_2)$. Thus an agent starting at node v can be uniquely identified in the tree using the string $BW(v)$, or using

any string describing a longer traversal which has $BW(v)$ as its prefix. The definition of the string $BW(v)$ is independent on the upper bound N on n which is known to the agent.

We also need to introduce some auxiliary notation. A *reverse basic walk* starting from node w with port p is a traversal of all edges of the tree ending at the starting node w and defined as follows. Node v is left by port p; when the walk enters a node by port i, it leaves it by port $(i-1)$ mod d, where d is the degree of the node.

For a string σ of length m, the rotation $rot_l(\sigma)$ is the string σ', such that $\sigma'[i] = \sigma[(i+l) \bmod m]$, for all indices $0 \leq i \leq m-1$. Any string σ can be uniquely encoded by its lexicographically minimal rotation $LMR(\sigma)$ and the smallest non-negative integer l such that $LMR(\sigma) = rot_l(\sigma)$.

Due to the periodic nature of tree traversal using the basic walk, all the strings $BW(v)$, for $v \in V$, are identical up to rotation, and hence have the same string describing their lexicographically minimal rotation. We define the signature $sig(v)$ of an agent with initial starting position v as the minimum l such that $LMR(BW(v)) = rot_l(BW(v))$. Hence, agents with non-symmetric initial positions have different signatures. Observe that $0 \leq sig(v) < 2(n-1)$.

To compute the value of $sig(v)$, we apply the following procedure, called FINDSIGNATURE, which allows an agent starting at node v to detect the starting position of $LMR(BW(v))$ as a rotation of $BW(v)$. To do this, we apply a variant of Duval's efficient maximum suffix algorithm [79] (cf. also [158] for an external I/O memory implementation), adapting it for the mobile agent computational model with limited memory. Intuitively, the agent makes use of two pointers to symbols of $BW(v)$, represented by positions $left$ and $right$, which it sweeps from left to right. Index $left$ represents the starting position of the lexicographically minimal rotation which has been detected so far, while index $right$ represents the currently considered candidate for such a starting position.

Our implementation of FINDSIGNATURE has two important features. Firstly, the comparison of characters within the string $BW(v)$ is encapsulated in subroutine COMPARESTRING ($left, right, maxLength$), which lexicographically compares the two substrings of $BW(v)$ having length $maxLength$ and starting at offsets $left$ and $right$, respectively. Secondly, the agent is not assumed to know the exact length $4(n-1)$ of sequence $BW(v)$; instead, the known upper bound of $4(N-1)$ is used, without affecting the correctness of the algorithm.

```
procedure FINDSIGNATURE ()
  left ← 1; right ← 2;
  repeat
    maxLength ← right − left;
    COMPARESTRING (left, right, maxLength);
    if 'left' string is greater { at some index } then
      left ← right; right ← right + 1;
    else
      if 'left' string is smaller at index i then
        right ← right + i;
      else {strings are equal}
        right ← right + maxLength;
  until right > 4(N − 1);
  return left;
```

When describing procedure COMPARESTRING, we will assume that the agent is equipped with four memory blocks, called *views*, each of which can store a substring of μ successive symbols from the string $BW(v)$, where μ is some integer smaller than $k/4$. In this way, by carefully implementing procedure COMPARESTRING as described in [T8], we obtain the following lemma.

Lemma 4.4 ([T8]). *For any upper bound N, such that $N \geq n \geq N/16$, and $k \geq c \log N$, where c is a constant, an agent starting at node v and equipped with k bits of memory can compute its signature $sig(v)$ in $O(n^2/k)$ rounds by following procedure* FINDSIGNATURE *for the value $\mu = k/8$.*

The rendezvous procedure for an agent with an already computed signature $sig(v)$ depends on the relation between the number k of memory bits and the known upper bound N on the order of the tree. The first procedure, called SMALLMEMORYRV, guarantees rendezvous of agents with known signatures in $O(N \log N)$ rounds and using $\Theta(\log N)$ bits of memory. Consequently, the procedure will be applied in the case when $k < N/\log N$, since then $N \log N \in O(N^2/k)$ and the bound of $O(N^2/k)$ on execution time is achieved. A faster procedure for agents with larger memory will be presented further on.

Procedure SMALLMEMORYRV assigns to each agent a unique label defined as the string of $\lceil 2 \log N + 3 \rceil$ bits encoding the binary representation of $2sig(v) + 1$. The procedure is composed of phases such that in the i-th phase, depending on the value of the i-th bit of this label, the agent either visits all the nodes of the tree at least twice, or waits at its initial location for a number of rounds corresponding to such an exploration. This is iterated for all i, $1 \leq i \leq \lceil 2 \log N + 3 \rceil$, and then the whole process is repeated until rendezvous is achieved. The traversal of the tree, which needs to be performed as

a subroutine, is implemented by $2(N-1)$ steps of the basic walk, and then returning to the starting node in $2(N-1)$ steps of the reverse basic walk.

```
procedure SMALLMEMORYRV ()
  sig ← FINDSIGNATURE ();
  repeat
    OSCILLATE (sig, 2(N − 1), 0);
  until rendezvous;

procedure OSCILLATE (sig, distance, firstPort)
  for i ← 1 .. ⌈2 log N + 3⌉ do
    for j ← 1, 2 do
      if i-th bit of (2sig + 1) is '1' then
        perform distance steps of the basic walk;
        perform distance steps of the reverse basic walk, starting from the
        last port of entry;
      else
        remain idle for 2 · distance steps;
```

Lemma 4.5 ([T8]). *For any upper bound N, such that $N \geq n \geq N/16$, and $k \geq c \log N$, where c is a constant, a pair of agents equipped with k bits of memory, starting at non-symmetric initial positions with arbitrary delay, can achieve rendezvous in time $O(n^2/k + n \log n)$ using procedure* SMALLMEMORYRV.

For the case when $k > N/\log N$, we make use of Procedure LARGEMEMORYRV, which applies a more time-efficient approach to rendezvous by restricting the meeting location of the agents either to a specific node of the tree, or to one of the endpoints of a specific edge. Since the memory of the agent may be sublinear compared to the order of the tree T, we do not perform a structural (e.g., DFS-based) analysis of the entire tree to determine such a location. Instead, the agent attempts to determine a meeting point in the so called *trimmed tree* T', which is the port-labeled tree given by the following construction (provided for purposes of definition, only):

1. Initially, let $T' = T$.

2. *Trimming*. Let $z = \lceil 32N/k \rceil < n/2$. Remove from T' all edges e such that one of the connected components of tree $T \setminus \{e\}$ has less than z nodes. Remove from T' all isolated nodes.

3. *Path contraction*. Remove from T' all nodes of degree 2 by contracting each path passing through such nodes into a single edge of the tree, preserving the port labeling at all the remaining nodes (of degree 1 or 3).

We remark that T' is a non-empty tree with at most $k/16$ nodes [T8], which are by definition also nodes of tree T. The meeting node of the agents in procedure LARGEMEMORYRV is selected as follows. If the trimmed tree has a central node v, then the agents will meet at v. Otherwise, the trimmed tree must have a central edge e which corresponds to some path $(v_0, v_1, \ldots, v_l)$ of length l in T. If l is even, then the agents meet at the node $v_{l/2}$. Otherwise, the agents meet at one of the endpoints of the edge $\{v_{\lfloor l/2 \rfloor}, v_{\lceil l/2 \rceil}\}$ of T. Observe that since T' is uniquely defined, the node or pair of nodes which will be selected for rendezvous is independent of the starting positions of the agents.

Procedure LARGEMEMORYRV relies on two key subroutines which allow the agent to navigate in the tree T'.

- Procedure TRIMMEDTREENEIGHBORHOOD, when called at a node u, computes the set of port numbers at node u which correspond to edges remaining after the trimming phase in the definition of tree T', i.e., edges of T leading from u to a subtree of at least z nodes. Testing if a port p at u leads to a sufficiently large subtree is implemented by performing $2z$ steps of the basic walk on T starting with port p at node u, memorizing the current tree-distance of the agent from u throughout this traversal. If the agent returns to u before completion of the last step of the walk, then the subtree has less than z nodes, and port p is not included in the output of the procedure.

- Procedure TRAVERSECOMPRESSEDPATH, when called at a node $u \in T'$ with a single argument $nextPort$ (describing a port number at u in T') moves the agent using port number $nextPort$, to its neighbor w in T', following a contracted path in T. The values returned by the procedure are the port number by which w was entered when coming from u, and the length of the path in T connecting u and w. An optional second argument passed to TRAVERSECOMPRESSEDPATH allows the agent to move a specified number of steps along the path between u and w in T, e.g., in order to reach its center.

Procedure LARGEMEMORYRV consists of the following phases. First, the agent follows the basic walk on T, starting from its initial position, until it encounters the first node which is identified as a leaf of tree T', by using procedure TRIMMEDTREENEIGHBORHOOD. Next, the agent performs a basic walk in tree T', using procedures TRIMMEDTREENEIGHBORHOOD and TRAVERSECOMPRESSEDPATH to discover node neighborhoods and to navigate along edges of T', respectively. A basic walk in T' is defined as in tree T, with the additional condition that an agent leaving a leaf follows the only available port, regardless of its port number. The agent memorizes the entire port number sequence BW' used during this basic walk in T' and, by keeping track of the T' tree-distance from the starting node, detects the completion of the tour of the

entire tree T'. Using local computations on the sequence BW', the agent now identifies the location of the central node or the central edge of tree T', expressed by the number of steps of the basic walk on T' required to reach this location from its initial position. If T' has a central node, then the agent reaches it, and stops, waiting for the other agent to arrive there. Otherwise, if T' has a central edge e, the agent proceeds to it and identifies the length l of the corresponding path $(v_0, v_1, \ldots, v_l)$ in T using procedure TRAVERSECOMPRESSEDPATH. If l is even, the agent moves to node $v_{l/2}$ by applying once more procedure TRAVERSECOMPRESSEDPATH, and stops. Otherwise, the agent reaches node $v_{\lfloor l/2 \rfloor}$ and applies procedure OSCILLATE. This is equivalent to performing SMALLMEMORYRV, but restricted to the two-node subtree (edge) $\{v_{\lfloor l/2 \rfloor}, v_{\lceil l/2 \rceil}\}$ of T.

procedure LARGEMEMORYRV () { starting at v }
  $sig \leftarrow$ FINDSIGNATURE ();
  while |TRIMMEDTREENEIGHBORHOOD()| $\neq 1$ **do**
    traverse one step of the basic walk on T;
  { perform the complete basic walk on the reduced tree T', using procedure TRIMMEDTREENEIGHBORHOOD to discover ports leading to neighbors in T' and TRAVERSECOMPRESSEDPATH to traverse edges of T' }
  $BW' \leftarrow$ basic walk string for reduced tree T' starting from the current location of the agent;
  { using string BW', locally compute whether T' has a central node or a central edge, and determine its location }
  $i \leftarrow$ distance along basic walk on T' to central node/edge of T';
  move for i steps of the basic walk on T';
  if T' has a central node **then**
    stop {at central node of T' }
  else {T' has a central edge, which has just been reached}
    ($returnPort$, l) $\leftarrow$ traverse central edge of T' using TRAVERSECOMPRESSEDPATH;
    { move to the center of the path in T corresponding to central edge of T' }
    ($port$, $\cdot$) $\leftarrow$ TRAVERSECOMPRESSEDPATH ($returnPort$, $\lceil l/2 \rceil$);
    if l is even **then**
      stop {in the middle of the central path of T' }
    else
      repeat
        OSCILLATE (sig, 1, $port$)
      until rendezvous;

A bound of the performance of Procedure LARGEMEMORYRV is given by the following lemma.

Lemma 4.6 ([T8]). *For any upper bound N, such that $N \geq n \geq N/16$, and $k \geq cN/\log N$, where c is a constant, a pair of agents equipped with k bits of memory, starting at non-symmetric initial positions with arbitrary delay, achieves rendezvous in time $O(n^2/k)$, using procedure* LARGEMEMORYRV.

Combining Lemmas 4.5 and 4.6, we obtain an algorithm which solves the rendezvous problem in time $O(n^2/k)$ for a known linear upper bound N on n and for trees of maximum degree 3: if $k > N/\log N$, then procedure LARGEMEMORYRV is called; otherwise, procedure SMALLMEMORYRV is called. A generalization of the approach to trees of arbitrary degree is possible by considering exploration of a virtual tree of degree 3, which in fact also explores the real underlying tree T which may contain nodes of higher degree. Independence of the value of n is achieved by a doubling argument, iterating geometrically increasing bounds on the value of a bound on n. Interestingly, by introducing artificial phases during which the agents wait, it is possible to design the procedure so that the agents meet quickly even if they assume different bounds on n, due to the fact that one of the agents is delayed. The complete procedure is presented below.

procedure RENDEZVOUSINTREES ()
  $N \leftarrow k$;
  repeat { Phase 1: attempt rendezvous with large memory }
    try LARGEMEMORYRV (), aborting if tree T' has more than $k/16$ nodes;
    { the further steps are executed only if the assumed bound is too small $(N < n)$ }
    return to the starting position;
    $N \leftarrow 4N$;
  until $k < N/\log N$;
  repeat { Phase 2: achieve rendezvous with small memory }
    $t \leftarrow$ current round number;
    wait for $2(N-1)$ rounds; { (a) }
    perform $2(N-1)$ steps of the basic walk; { (b) }
    wait for $2(N-1)$ rounds; { (c) }
    perform $2(N-1)$ steps of the reverse basic walk; { (d) }
    $\tau \leftarrow 8(N-1)$ + duration of the slowest possible execution of FINDSIGNATURE for current N;
    $sig \leftarrow$ FINDSIGNATURE ();
    wait until round number $t + \tau$;
    repeat
      OSCILLATE (sig, $2(N-1)$, 0);
    until round number is larger than $t + 2\tau$;
    $N \leftarrow 4N$;
  until rendezvous;

By applying the above procedure, one can show that rendezvous is guaranteed to occur while the second (delayed) agent is still assuming a bound on n which is linear with respect to n. By bounding the number of steps this agent performs before rendezvous, we obtain the following theorem.

Theorem 4.7 ([T8]). *For any $k \geq c \log n$, where c is some constant, there exists a pair of agents equipped with k bits of memory which achieves rendezvous in time $O(n^2/k)$ when starting at non-symmetric initial positions with arbitrary delay.*

It turns out, however, that no pair of agents can accomplish rendezvous in time $o(n + n^2/k)$, even in the class of lines and even with simultaneous start.

Theorem 4.8 ([T8]). *Consider a pair of agents equipped with k bits of memory and achieving rendezvous in any n-node line starting from arbitrary non-symmetric initial positions. Then:*

1. *For some constant c_1 and arbitrarily large n, we have $k \geq c_1 \log n$.*

2. *For some constant c_2 and arbitrarily large n, there exists a n-node line for which these agents use time at least $c_2(n + n^2/k)$ to accomplish rendezvous from some non-symmetric initial positions, even for simultaneous start.*

For an agent knowing an upper bound on the number of nodes n of a tree, we can compare the hardness of the rendezvous and exploration problems. Exploration with termination can be performed by following $2(n-1)$ steps of the basic walk, given that the agent is equipped with at least $O(\log n)$ bits of memory to be able to count the number of completed steps. On the other hand, the rendezvous problem can only be solved in linear time when given linear memory, and requires $\Omega(n^2/\log n)$ time for a log-space agent.

4.3 Other models of rendezvous

The results from [164, T7] discussed in Section 4.1 provide an almost complete answer to the question of when the rendezvous of a pair of synchronous agents in a graph is feasible. Agents can meet if they have distinct labels, or if they have the same labels but distinct initial views. In any other case, rendezvous will not be feasible, as long as the agents start simultaneously. Considerations of larger groups of agents have led Dieudonné and Pelc [71] to a similar criterion for gathering a larger group of agents at a single node. They introduce the notion of an *enhanced view* of the agent, defined as its view of the anonymous network with additional labels assigned to nodes, equal to 0 for all nodes which do not contain an agent, and to L for any node initially containing an

agent with label L.* Then, [71] show that a starting configuration with multiple agents is gatherable, if and only if there exist agents with different views, and each agent has a unique enhanced view. Moreover, they provide a universal algorithm for gathering all gatherable configurations.

The situation becomes more complex for the case of asynchronous agents. An asynchronous system can be thought of as controlled by an adversary, which decides for each agent the duration of its traversal of an edge of the network. If agents are only allowed to meet at nodes, then the adversary controlling the speeds of the agents can always guarantee that the agents never meet while they are both in motion. In fact, rendezvous can only be achieved when one of the agents comes to a definite stop and decides to wait for the other agent at some node, while the other either engages in perpetual exploration of the graph along a trajectory which includes the node at which the other agent is waiting, or simply stops at the same node. This restricts the feasibility of asynchronous rendezvous of anonymous agents to a narrow class of networks, including those in which an agent can elect a unique leader node (i.e., networks equal to their own quotient graph, in the case when the agents know a bound on the number of nodes of the network). For this reason, most of the literature assumes a different definition of asynchronous rendezvous, in which agents can meet not only at nodes, but also while "traversing an edge". Alternatively, one can view the moves of the agent along an edge as instantaneous, and allow the adversary to introduce arbitrary delays in the time spent by the agent at nodes. In such an asynchronous scenario, it has recently been shown that agents can always meet within a polynomial number of moves if they have unique labels [72]. For the case of anonymous agents, the class of instances for which asynchronous rendezvous is feasible is quite similar to that in the synchronous case, though due to the ability of the agents to meet on edges, certain configurations with a mirror-type symmetry also turn out to be gatherable [102].

Finally, we remark on the complexity and feasibility of randomized rendezvous. For rendezvous of a pair of agents, a reasonable approach consists in allowing both agents to perform a random walk until they meet. As time tends to infinity, the probability of successful completion of rendezvous tends to 1; thus, such an approach may be considered to be randomized in the Las Vegas sense. When gathering a larger group of agents, agents which have met stick together following the same walk, and the process continues until all groups of agents have merged into one. Such processes, known under the name of coalescing walks, have been extensively studied in the literature [5, 43, 51]. Aldous [5] established that the expected time until two identical, synchronous Markovian processes meet is bounded by a constant times the maximum hitting time of the walk, over the nodes of the graph. Thus, if both of the agents were for example to fol-

*The scenario defined in [71] is in fact restricted to anonymous agents, i.e., the assumption that the agent's label satisfies $L = 1$ is made for all agents. A similar generalization of the concept of view was previously introduced by Yamashita and Kameda in [173].

low a Metropolis walk, their expected meeting time would be $O(n^2)$. By considering a network consisting of two identical copies of some graph, containing one agent each at some two corresponding nodes, and joined by connecting with an edge the nodes with the highest hitting time, it follows that such a bound is asymptotically tight. Achieving improved bounds on the time of randomized rendezvous would only be possible given the existence of substantially faster randomized exploration algorithms than those currently known, which appears very unlikely.

It is worth noting that the technique of achieving randomized rendezvous of agents (or tokens) in a network by means of coalescing walks proves useful in different contexts. In the study of self-stabilizing algorithms, the algorithm of Israeli and Jalfon [110] solves the mutual exclusion problem by providing a method of eliminating superfluous tokens in a distributed network through coalescence, until the system eventually only contains a single token, circulating among the nodes. The role of the token is to select the unique active processor in the system. Concepts related to randomized rendezvous also appear in techniques for bounding the mixing time of random processes, which rely on the analysis of the time needed by a specifically contrived pair of Markovian processes to coalesce to an identical state for both processes of the pair. This approach, known as coupling [4, 44], has proved to be a powerful tool, used among others to design parallel algorithms for rapid shuffling of a permutation [54].

5 Challenges in agent-based distributed computing

The results presented in this manuscript cover a range of topics related to agent-based computing in anonymous graphs. The central theme question which we have ask throughout the text can be posed as follows: "What are the computational capabilities of a single agent in an anonymous network?", and one can safely say that the state-of-the-art of the literature, including the results discussed in this thesis, provide us with a reasonable understanding of the answer. What is less well understood is the optimal time complexity of solutions to problems using one or more mobile agents under different model restrictions, and the precise interplay between the time and memory complexity of the agent.

5.1 Directions of study for anonymous networks

Time-space tradeoffs. A fundamental and current research problem concerns the interplay between solution time and the memory space required by the agent, for both randomized and deterministic approaches. In Chapter 2 we have pointed out that a deterministic agent equipped with a logarithmic number of bits of memory can explore a graph in $O(n^4 \log n)$ steps, a similar randomized agent only needs $O(n^2 \log n)$ steps, while the best known lower bound on exploration time for both types of agents is only $n^{1+\Omega(1/\log\log n)}$. This allows us to formulate intuitive working hypotheses, for example, that an agent can solve many exploration-based problems significantly faster by applying randomization, but an important complexity gap is still left to be resolved. For the moment, there is still no known framework which could combine known results in the area. It would be of particular interest to combine the $\Omega(n^2)$ lower bound on the cover time of matrix-based random walks [149] and a similar lower bound of $\Omega(n^2)$ on the length of universal tables for deterministic exploration (Section 2.3) into a general lower bound which could apply to randomized algorithms for mobile agents.

Interestingly, for the exploration problem, we have no indication that agents equipped with large memory (say, polynomial in n) are able to explore an anonymous graph in asymptotically less time than algorithms with only $\Theta(\log n)$ memory. This comes in contrast to non-anonymous models, such as JAG-based approaches [T1], where adding more memory to the agent allows us to design faster exploration algorithms. Of course,

this does not mean that the memory size of the agent has no impact on the time of computation. For example, in Chapter 4 we have shown that providing a mobile agent with larger memory provably reduces the completion time for the fundamental problem of anonymous synchronous rendezvous, in the class of trees. For the rendezvous problem in general graphs, the impact of determinism and memory restriction on the time required for finding the solution seems to be even more noticeable, though in this case optimal time bounds on the complexity of the problem are not known and we can only try to draw conclusions based on the performance of the best currently known algorithm for the rendezvous problem.

Another design characteristic of the agent, which influences its computational capabilities, is knowledge of the value of global parameters. We have observed in Chapters 2 and 3 that single-agent problems, such as exploration and map construction, cannot as a rule be solved with termination when no upper bound on the number of nodes of the network is known by the agent. A more subtle question concerns the impact of knowledge of global parameters on the time needed to complete certain tasks without termination, such as covering all the nodes of the network during perpetual exploration. In Section 2.2.2, we have noted that the running time of variants of the Metropolis walk with small memory size are sensitive to knowledge of global parameters. At present, the fastest known randomized exploration algorithms making use of $o(\log\log\log n)$ bits of memory rely on knowledge of an upper bound on n, whereas the fastest known exploration algorithms for agents with more memory (i.e., $\Omega(\log\log\log n)$ bits) do not make use of the value of n. One of our goals in future research will be to obtain a better understanding of the three-way tradeoff between time, space, and knowledge in mobile agent computing, complementing similar studies in "classical" models of distributed computing with processors on nodes [124].

Model variants. Agent-based computing on anonymous networks is, above all, a theoretical model of computation. Different refinements of the model are possible, allowing us to draw general conclusions about the types of resources critical for solving fundamental problems on graphs. Here, we briefly highlight three promising research directions, which have been hinted at in the previous chapters of the manuscript.

First of all, the anonymous network model provides an excellent description for agents whose memory is simply too small to accommodate a single node identifier. By describing the agent as a Mealey automaton, and not as e.g. as a RAM machine operating on memory addresses of size $O(\log n)$, we have a natural setting for studying algorithms with sub-logarithmic memory complexity in the context of classical problems on graphs. The computational power of agents with sublogarithmic memory is not yet well understood. It is a well-established result [159] that an agent requires $\Omega(\log n)$ memory to explore all graphs of at most n nodes, but it is not known if there exists an agent with $o(\log n)$ memory which can explore any tree of at most n nodes and termi-

nate in a finite number of steps. The impact of memory size in the sub-logarithmic range is visible also for randomized algorithms. As noted in Chapter 2, the number of memory bits required by an agent to perform exploration in expected $\tilde{O}(n^2)$ steps is bounded by $O(\log\log n)$. Decreasing the memory size to 0 increases the exploration time to $\Theta(n^3)$, whereas increasing the memory size above $O(\log\log n)$ does not appear to speed up the process asymptotically. In current work-in-progress, we are undertaking an attempt to understand time-space tradeoffs for small memory more fully, and also to find out if any non-trivial topological properties of the anonymous network (related to the structure of its quotient graph) can be decided with high probability by means of randomized agents with sublogarithmic memory.

Another important question concerns the availability of local information to an agent upon entering a node. Whereas the anonymous network model does away with all forms of identifiers, we assume that the agent still knows the degree of the node it is located at, and the port by which this node was entered. Whereas the degree information appears to be indispensable to be able to design any reasonable algorithm, the information about the port of entry has a more pragmatic rationale: without it, we are unable to reverse the last move made by the agent. In the scenario without port-of-entry information, we are still able to perform randomized graph exploration by using the random walk, as well as deterministic graph exploration in polynomial time by following a Universal Traversal Sequence based on the random walk. The question of the existence of faster exploration strategies with $o(n^3)$ expected cover time remains open; in particular, it does not appear to be possible to adapt the Metropolis walk to work without the ability to revert to the last visited node. Whereas it is still possible to perform deterministic exploration with $O(\log n)$ memory carried over links, the construction of Universal Exploration Sequences with local log-space computation [156] is not applicable, and log-space deterministic exploration strategies are only known for the case when certain assumptions are made abut the port labeling in the graph [157]. The quotient graph of the network can still be reconstructed in polynomial time using an adaptation of the method described in Section 3.2, but we do not know how to adapt any of the other methods, in particular, if it is possible to construct the quotient graph carrying $O(\log n)$ memory over links. Likewise, the question of deterministic rendezvous in $O(\log n)$ space remains open.

The idea of removing the ability to backtrack along previously traversed links leads to further generalizations of the anonymous network model to the case of directed graphs. When the digraph in which the agents operate is regular, or at least (almost) Eulerian, then the cover time of the random walk is still polynomial [145]. The complexity of solving deterministic exploration problems, quotient graph construction, and rendezvous remains the same as in the previously discussed case of undirected graphs without knowledge of port-of-entry. For Eulerian digraphs, the question of whether an exploration strategy exists which requires log-space computations in the centralized RAM model has profound theoretical implications; Reingold *et al.* [157] have shown

that the existence of such a strategy would imply that deterministic log-space machines have the same power as randomized log-space machines, proving the equivalence of complexity classes L=RL. The ability to prove the existence of similar deterministic exploration strategies for arbitrary (strongly connected) directed graphs would imply the even stronger relation L=NL. Any collapse or separation of the known hierarchy L $\subseteq$ RL $\subseteq$ NL will certainly be a breakthrough in log-space computability at least comparable to Reingold's [156] proof that L=SL. Most partial attempts to address these problems rely on developments in derandomization techniques starting with the seminal work of Nisan [146], asking if a deterministic log-space algorithm can achieve the same behaviour as a randomized log-space algorithm which it simulates for some small pseudorandom seed. Results in this area have a direct bearing on agent-based exploration algorithms, sometimes requiring only a change of language to transfer from the centralized model to the distributed model.

5.2 Computing in a team

An important line of research in distributed computing theory of the last decade deals with the problem of achieving a given goal using the coordinated effort of a team of agents. Studies have been performed in different models of agent communication and synchronization, and attempted tasks include fundamental problems of distributed computing, such as rendezvous at specific locations, or stabilization on subsets of the network satisfying certain properties. In Chapter 4 we have provided a brief overview of the state-of-the-art for the rendezvous problem in anonymous graphs, also mentioning its applications to problems such as leader election. A completely separate line of study deals with the problem of using a coordinated team of agents to detect a hazardous network fault or a rogue agent, most notably in the so called "black-hole search" [74, 85, 126] and "distributed cops-and-robber search" problems [116, 125].

Apart from the above mentioned tasks whose very definition relies on the existence of multiple agents in the network, an extremely relevant research direction concerns investigations into collaborative solutions to problems such as graph exploration, for which a single agent suffices to obtain a valid solution, but computations in a team can lead to a solution more quickly. Here, the main object of focus is the *speed-up* of the approach, i.e., informally speaking, the decrease in the running time of an algorithm when deploying multiple agents in the network instead of one.

The speed-up depends on many factors, including the initial location of the agents and the possible interactions between them. We illustrate this question with a simple real-world example, which is indicative of the need for agent coordination. Consider a document repository forming an e-book, divided into HTML documents consisting of one page of the book each, with each page containing a hyperlink to the next and previous pages, only. Suppose that an agent is a process which reads a page, and then

follows a random hyperlink on this page. The task of the team of agents is to verify the integrity of the repository, i.e., to check if the book contains all of its pages in order. If not, any one of the agents is expected to signal an error. The goal is to complete the task in the shortest possible time (in expectation). It turns out, that depending on the way the agents coordinate with each other, we may benefit to a varying degree by deploying a larger number of agents to solve the task. Consider, e.g., three different strategies for solving the studied exemplary problem:

A. *The agents explore the book in parallel, starting from page 1.* This strategy achieves little gain: the speedup of execution is very small (logarithmic) compared to the number of deployed agents [9].

B. *The agents process the book in parallel, starting from evenly spread out pages of the book.* In this strategy, the execution speeds up by more than a linear factor in terms of the number of deployed agents (more precisely, the expected speed-up is $\Theta(k^2/\log^2 k)$ [118] for k agents), leading to a highly desirable synergy effect.

C. *The agents process the book in parallel, starting from evenly spread out pages of the book, with the added stipulation that each agent is confined to its own section of the book ("block" of pages).* This strategy achieves even stronger synergy: execution speeds up by a factor of $\Theta(k^2/\log k)$ (the argument follows directly from the probability distributions of hitting times of one-dimensional random walks, cf. [139]).

When performing "uncoordinated" random walks, with agents deployed in a graph independently of each other, the achieved speed-up of exploration has been characterized in [9,80,83] (cf. Section 2.2). The value of this speed-up may sometimes be as low as logarithmic, and sometimes as high as exponential, in terms of k. For many classes of graphs which often appear in networking practice the speed-up is linear in terms of k, but only for small values of k ($k \in O(\log n)$).

Many open questions still remain regarding the parallelization of graph exploration. To begin with, the random walk is not the only exploration strategy for which speed-up in collaborative computations with a team of agents can be measured. For example, our work [T1] indicates that multiple agents following independent Metropolis walks starting from uniformly random network nodes cover a graph much faster than a single Metropolis walk. Currently, there is still no general theory of parallel walks. The question of finding the exploration algorithm which allows k agents acting independently and in parallel to explore a network as quickly as possible is also wide open.

In our current work-in-progress, we have considered the rotor-router mechanism (Section 2.5.1) in the context of multiple agents. Here, the agents moving in the rotor-router system interact with the same set of pointers at nodes, meaning that they cannot be treated as independent walkers in the graph. Still, the obtained values of speed-up

are surprisingly similar to those for the case of multiple random walks. For the case of the ring, we have established [118] that the worst-case cover time of the rotor-router with k agents is precisely $\Theta(\log k)$ times smaller than the corresponding worst case cover time for a single agent. In work currently in preparation, we show that k agents in fact achieve a speed-up of $\Omega(\log k)$ in arbitrary networks. At a very informal level of description, the speed-up of the multi-agent rotor-router on the ring stems from the fact that each agent is associated with a so-called *domain*, i.e., a part of the graph it has already explored, close to its current location and not recently visited by any other agent.

Other techniques may be applied for agents in the anonymous network model which are allowed to communicate with each other. The first theoretical results in the area are due to [92], who show that there exists a strategy which explores a tree with a team of k agents starting from the same node with a speed-up of $\Theta(\log k)$ with respect to the best possible strategy for a single agent. This result holds for small k ($k < \sqrt{n}$) and a communication model which either allows agents to write on whiteboards or to communicate at a distance. The question of the optimality of such an approach stands wide open, with the best known upper bound on speed-up in such a model being $O(k/\log k)$. Very recently, we have also studied the same question of collaborative exploration with much larger team sizes. In [67], we show that in an arbitrary anonymous network with m edges and diameter D, a team of $k > m^{1+\epsilon} D$ agents starting from the same node explores the network in asymptotically optimal time $O(D)$.* This result holds for an even more restrictive model in which two agents can only communicate when located at the same node. For smaller team sizes than the result proposed in [67], establishing the exact tradeoff between the team size k and the exploration time is a highly relevant and challenging open problem. This question can be considered both in the context of anonymous networks, and networks with unique node identifiers known to the agent.

5.3 Some new models and inspirations

The future of mobile agent computing is intrinsically tied to new, emerging areas of application. The last decade has seen intensive research into random walks tuned for applications on the web and in social networks. Approaches based on random walks have been elaborated which shift (bias) the agent towards spending more time at hubs or nodes of high importance, often applying machine learning algorithms to compute according edge weights to guide the agent in its walk [134]. Theoretical foundations have been laid out for network exploration using parallel random walks [9, 80, 83], and several variants of the random walk achieving improved cover time have been put forward [31, 147, 149]. It appears likely that future research will bring answers to more

*Strictly speaking, our results from [67] are not set in the anonymous network model. We thank Jérémie Chalopin for bringing to our attention that they also hold in anonymous networks.

complex questions, combining the elements of theory developed so far. For example, we may ask about the optimal collaborative exploration strategy in order to periodically explore the nodes of a web-type graph with uniform probability, given a set of k agents crawling it in parallel while the network topology is constantly undergoing evolution according to some model of link formation. In addition to the problems of multi-agent protocol design discussed in Section 5.2, the aspect of dynamism is particularly challenging, since our understanding of the behavior of random walks agent-based algorithms in evolving networks is relatively limited for both adversarial and non-adversarial scenarios (cf. [22] for one of the few papers on the topic).

The problem of coping with network evolution is closely linked to challenges related to detecting and evaluating the importance of information appearing in real time in social networks and webs of knowledge. New algorithms for indexing the web need to pick up on hot news items and other resources which are quickly gaining new inbound links in the network, while skillfully eliminating useless background noise and remaining immune to intentional manipulation of the topology of the web (e.g., through so-called "link farms"). In social networks, the patterns of propagation of information are particularly complex [115, 121]. Agent-based protocols may prove to be a viable approach to simulating such processes of rumour spreading, detecting which nodes of the network are capable of starting a cascade of information which will reach a large subset of network nodes, etc.

Another important area where the theory of mobile agent computing has been lagging a little behind real-world designs is that of collaborative multi-robot designs in robotics. When undertaking such challenges, we immediately run into the theoretical challenges of coordinating a team of robots, highlighted in Section 5.2. These problems are additionally aggravated by difficulties of real-world implementation. One pressing aspect is to develop models and algorithms for heterogeneous robots which are cooperating with each other. In the simplest case, some robots may move faster than others. Under the assumption that robots have different maximum speeds, most known theoretical results for problems of search and exploration no longer hold. Even classical problems, such as locating a distinguished point in a one-dimensional terrain (the so-called "cow-path problem"), admit algorithmic solutions with a more involved structure, depending on the relative speeds of the robots [57]. Another example is the problem of patrolling a segment or a circular terrain so as to minimize the maximum time when a point of the circumference is left unguarded. The task admits a simple solution with regular robot trajectories when all the robots have the same speed, but this is no longer the case for robots with different speeds [58, 114]. In fact, the optimal design of trajectories for such robots remains an open problem, even for the simple cases of a segment or a ring. For a set of k robots with known maximum speeds, the solution to the patrolling problem on a circular terrain is closely related to the lonely runner conjecture [33] — an open question in number theory first posed in 1967 (related to the Littlewood, Goldbach,

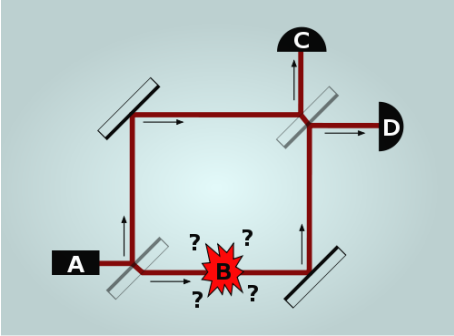

FIGURE 5.1: The Elitzur-Vaidman bomb-testing problem (image source: [169])

and Polignac conjectures), which has recently been partially addressed using methods from dynamical systems theory [106]. Whereas this type of heterogeneous multi-agent problem appears to be very hard in a centralized setting, i.e., when all the robots can coordinate their actions, the distributed setting in which all agents decide on their own actions is not at all understood and definitely deserves further study.

Finally, it is extremely tempting to try to extend of the mobile-agent computing framework to unconventional models of computation, and in particular to quantum computing. This direction appears natural, since in the design of many quantum algorithms, the so-called *quantum walk* serves as a key building block. The quantum walk approach can be regarded as a quantum analogue of the classical random walk, in which the walker's current state is no longer a probability distribution over locations in the graph, but instead a quantum superposition of locations in the graph. Such an approach can be used to obtain quantum algorithms for problems such as element distinctness in a list or detecting if the graph is triangle-free, which are faster by a polynomial factor than the fastest possible classical algorithms for the same problems [14, 140]. By treating the walker as a mobile agent, we could perhaps try to consider quantum analogues of the problems studied in this work: quantum exploration, quantum rendezvous, etc. The first hurdle when undertaking this approach lies in obtaining a precise formulation of, e.g., what it means that two agents meet, given a model of computation in which the location of the agent only becomes well-defined in the measurement phase at the end of the quantum process, and not during its intermediate steps. Since we have yet to provide a way to cope completely with this problem, we close this manuscript with some intuition and a word of caution related to a famous "paradox" in an experiment known as the Elitzur-Vaidman bomb-testing problem [81], which arises in the quantum setting. The experimental set-up is illustrated in Figure 5.1. In the experiment, the light source A emits a single photon (particle), which we can think of as our walker or "agent" in the system. Immediately after leaving the source, the photon passes through a half-plated mirror, and is reflected from the mirror along the upper path with probability $1/2$, or continues through the mirror and along the lower path with probability $1/2$. A box B is

placed on the lower path in the way of the photon. The box may either be a dud (which does nothing and does not affect the path taken by the photon), or a light-triggered bomb which explodes immediately after the photon passes through it. Finally, after passing through another half-plated mirror placed in the way of both the lower and the upper path, the potentially surviving photon is captured and measured by exactly one of the light detectors C or D.

A classical view of the above scenario would tell us that the "agent" chooses either the upper path, or the lower path, each with probability $1/2$. Consequently, it either passes through box B, or does not pass through it, with each of these events occurring with probability $1/2$. This means that in order to be able to identify the box B as a real bomb with certainty, we would need to know that the "agent" has passed through point B, implying that the bomb would always have to explode during such positive identification. Somewhat surprisingly, this analysis no longer holds under the laws of physics, as can be shown in a real-world experiment. Applying quantum mechanics, a more accurate view of the route taken by our agent is not that of a deterministic choice of the upper or lower path, each with probability $1/2$, but instead a quantum superposition of these two paths, which may, somewhat informally, be thought of as a probability wave going through both of these paths at once. As such, the photon never fully goes through point B, even if it interacts with it. By performing a formal analysis, Elitzur and Vaidman [81] obtained that by reading the measurements of the detectors C and D, the box at B may in some runs of the experiment (i.e., with a strictly positive probability, equal to $1/4$) be identified as a real bomb for certain, without actually exploding.

This simple example indicates at least one possible approach which could be used to define the quantum mobile agent framework. Rather than saying that the agent has gone through point B, we can say (and check) that the agent has interacted with point B. Likewise, rather than requiring that two agents achieve quantum rendezvous in some time round, we can require that by the end of the algorithm, the agents will have exchanged some information, thus proving that at some point in time they interacted with each other at the same location. As mentioned before, developing this type of intuition into a consistent theory, and more importantly — one which will yield useful theoretical results — appears to be a challenging, but potentially rewarding research task.

Bibliography

[1] Y. Afek and E. Gafni. Distributed algorithms for unidirectional networks. *SIAM Journal on Computing*, 23(6):1152–1178, 1994.

[2] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[3] S. Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32(1):123–143, 2002.

[4] D. Aldous. Random walks on finite groups and rapidly mixing Markov chains. In *Seminaire de Probabilites XVII*, pp. 243–297. Springer-Verlag, 1983. Lecture Notes in Math. 986.

[5] D. Aldous. Meeting times for independent markov chains. *Stochastic Processes and their Applications*, 38(2):185–193, August 1991.

[6] D. Aldous and J. Fill. *Reversible Markov Chains and Random Walks on Graphs*. http://stat-www.berkeley.edu/users/aldous/RWG/book.html, 2001.

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

[8] A. Almeida, G. Ramalho, H. Santana, P. A. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In A. L. C. Bazzan and S. Labidi, editors, *SBIA*, Lecture Notes in Computer Science vol. 3171, pp. 474–483. Springer, 2004.

[9] N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one. In *SPAA*, pp. 119–128. ACM, 2008.

[10] S. Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995.

[11] S. Alpern. Rendezvous search on labelled networks. *Naval Research Logistics*, 49(3):256–274, 2002.

[12] S. Alpern and S. Gal. *The theory of search games and rendezvous*. International Series in Operations research and Management Science. Kluwer Academic Publishers, 2002.

[13] C. Àlvarez and R. Greenlaw. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity*, 9(2):123–145, 2000.

[14] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007.

[15] C. Ambühl, L. Gasieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. *ACM Transactions on Algorithms*, 7(2):17, 2011.

[16] E. Anderson and S. Fekete. Asymmetric rendezvous on the plane. In *Proceedings of 14th Annual ACM Symposium on Computational Geometry (SoCG)*, pp. 365–373, 1998.

[17] E. Anderson and S. Fekete. Two-dimensional rendezvous search. *Operations Research*, 49(1):107–118, 2001.

[18] E. Anderson and R. Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability*, 27(4):839–851, 1990.

[19] D. Angluin. Local and global properties in networks of processors. In *Proceedings of 12th Symposium on Theory of Computing (STOC)*, pp. 82–93, 1980.

[20] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[21] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988.

[22] C. Avin, M. Koucký, and Z. Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP (1)*, Lecture Notes in Computer Science vol. 5125, pp. 121–132. Springer, 2008.

[23] D. Baba, T. Izumi, F. Ooshita, H. Kakugawa, and T. Masuzawa. Space-optimal rendezvous of mobile agents in asynchronous trees. In *Proceedings of the 17th international conference on Structural Information and Communication Complexity*, SIROCCO'10, pp. 86–100, Berlin, Heidelberg, 2010. Springer-Verlag.

[24] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pp. 635–644, New York, NY, USA, 2011. ACM.

[25] P. Baldi, P. Frasconi, and P. Smyth. *Modeling the Internet and the Web: Probabilistic Methods and Algorithms*. Wiley, 2003.

[26] G. Barnes and U. Feige. Short random walks on graphs. In S. R. Kosaraju, D. S. Johnson, and A. Aggarwal, editors, *STOC*, pp. 728–737. ACM, 1993.

[27] V. Baston and S. Gal. Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. *SIAM Journal on Control and Optimization*, 36(6):1880–1889, 1998.

[28] V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 48(8):722–731, 2001.

[29] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.

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

[31] P. Berenbrink, C. Cooper, R. Elsässer, T. Radzik, and T. Sauerwald. Speeding up random walks with neighborhood exploration. In M. Charikar, editor, *SODA*, pp. 1422–1435. SIAM, 2010.

[32] S. N. Bhatt, S. Even, D. S. Greenberg, and R. Tayar. Traversing directed Eulerian mazes. *Journal of Graph Algorithms and Applications*, 6(2):157–173, 2002.

[33] W. Bienia, L. A. Goddyn, P. Gvozdjak, A. Sebö, and M. Tarsi. Flows, view obstructions, and the lonely runner. *J. Comb. Theory, Ser. B*, 72(1):1–9, 1998.

[34] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *Proceedings of 18th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 181–188, 1999.

[35] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *Proceedings of 15th International Conference on Distributed Computing (DISC)*, LNCS vol. 2180, pp. 33–47, 2001.

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

[37] F. Bonnet and M. Raynal. Anonymous asynchronous systems: the case of failure detectors. *Distributed Computing*, 26(3):141–158, 2013.

[38] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier. Influence maximization in social networks: Towards an optimal algorithmic solution. *CoRR*, abs/1212.0884, 2012.

[39] A. Borodin, W. L. Ruzzo, and M. Tompa. Lower bounds on the length of universal traversal sequences. *J. Comput. Syst. Sci.*, 45(2):180–203, 1992.

[40] M. Brautbar and M. Kearns. Local algorithms for finding interesting individuals in large networks. In A. C.-C. Yao, editor, *ICS*, pp. 188–199. Tsinghua University Press, 2010.

[41] G. Brightwell and P. Winkler. Maximum hitting time for random walks on graphs. *Random Struct. Algorithms*, 1(3):263–276, Oct. 1990.

[42] A. Z. Broder, A. R. Karlin, P. Raghavan, and E. Upfal. Trading space for time in undirected s-t connectivity. *SIAM J. Comput.*, 23(2):324–334, 1994.

[43] N. H. Bshouty, L. Higham, and J. Warpechowska-Gruca. Meeting times of random walks on graphs. *Inf. Process. Lett.*, 69(5):259–265, 1999.

[44] R. Bubley and M. Dyer. Path coupling: A technique for proving rapid mixing in markov chains. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pp. 223–231, Washington, DC, USA, 1997. IEEE Computer Society.

[45] L. Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86:195–282, 1978.

[46] J. Chalopin. *Distributed Algorithms, Local Calculations, and Graph Homomorphisms*. PhD thesis, 2006.

[47] J. Chalopin, E. Godard, Y. MĂ©tivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In A. A. Shvartsman, editor, *OPODIS*, Lecture Notes in Computer Science vol. 4305, pp. 187–201. Springer, 2006.

[48] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4):312–340, 1997.

[49] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, Lecture Notes in Computer Science vol. 2719, pp. 1181–1196. Springer, 2003.

[50] S. Cook and C. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9(3):636–652, 1980.

[51] C. Cooper. Random walks, interacting particles, dynamic networks: Randomness can be helpful. In A. Kosowski and M. Yamashita, editors, *SIROCCO*, Lecture Notes in Computer Science vol. 6796, pp. 1–14. Springer, 2011.

[52] C. Cooper and A. M. Frieze. Crawling on web graphs. In J. H. Reif, editor, *STOC*, pp. 419–427. ACM, 2002.

[53] J. N. Cooper and J. Spencer. Simulating a random walk with constant error. *Combinatorics, Probability and Computing*, 15(6):815–822, 2006.

[54] A. Czumaj, P. Kanarek, M. Kutylowski, and K. Lorys. Delayed path coupling and generating random permutations via distributed stochastic processes. In R. E. Tarjan and T. Warnow, editors, *SODA*, pp. 271–280. ACM/SIAM, 1999.

[55] J. Czyzowicz, D. Dereniowski, L. Gasieniec, R. Klasing, A. Kosowski, and D. Pajak. Collision-free network exploration. Technical report, 2013. http://hal.inria.fr/hal-00736276.

[56] J. Czyzowicz, S. Dobrev, L. Gasieniec, D. Ilcinkas, J. Jansson, R. Klasing, I. Lignos, R. A. Martin, K. Sadakane, and W.-K. Sung. More efficient periodic traversal in anonymous undirected graphs. In S. Kutten and J. Zerovnik, editors, *Proceedings of 16th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS vol. 5869, pp. 167–181. Springer, 2009.

[57] J. Czyzowicz, L. Gasieniec, K. Georgiou, E. Kranakis, and F. MacQuarrie. The beachcombers' problem: Walking and searching with mobile robots. *CoRR*, abs/1304.7693, 2013.

[58] J. Czyzowicz, L. Gasieniec, A. Kosowski, and E. Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In C. Demetrescu and M. M. Halldórsson, editors, *ESA*, Lecture Notes in Computer Science vol. 6942, pp. 701–712. Springer, 2011.

[59] J. Czyzowicz, L. Gasieniec, and A. Pelc. Gathering few fat mobile robots in the plane. *Theor. Comput. Sci.*, 410(6-7):481–499, 2009.

[60] J. Czyzowicz, A. Labourel, and A. Pelc. How to meet asynchronously (almost) everywhere. In *Proceedings of 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 22–30, 2010.

[61] H. K. Dai and K. E. Flannery. Improved length lower bounds for reflecting sequences. In J. yi Cai and C. K. Wong, editors, *COCOON*, Lecture Notes in Computer Science vol. 1090, pp. 56–67. Springer, 1996.

[62] G. D'Angelo, G. D. Stefano, and A. Navarra. *Gathering Asynchronous and Oblivious Robots on Basic Graph Topologies Under the Look-Compute-Move Model*, pp. 197–222. Springer, 2013.

[63] S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the EACTS*, 109:54–69, 2013.

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

[65] N. G. de Bruijn and T. Aardenne-Ehrenfest. Circuits and trees in oriented linear graphs. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 28:203–217, 1951.

[66] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[67] D. Dereniowski, Y. Disser, A. Kosowski, D. Pajak, and P. Uznanski. Fast collaborative graph exploration. In *ICALP 2013*, page 526.

[68] D. Dereniowski, A. Kosowski, and D. Pajak. Distinguishing views in symmetric networks: a lower bound. Technical report, 2013.

[69] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[70] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. In A. W. Richa and C. Scheideler, editors, *SSS*, Lecture Notes in Computer Science vol. 7596, pp. 64–76. Springer, 2012.

[71] Y. Dieudonné and A. Pelc. Deterministic gathering of anonymous agents in arbitrary networks. *CoRR*, abs/1111.0321, 2011.

[72] Y. Dieudonné, A. Pelc, and V. Villain. How to meet asynchronously at polynomial cost. *CoRR*, abs/1301.7119, 2013.

[73] J. Ding, J. R. Lee, and Y. Peres. Cover times, blanket times, and majorizing measures. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pp. 61–70, New York, NY, USA, 2011. ACM.

[74] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. vol. 19, pp. 1–18, 2006.

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

[76] B. Doerr and T. Friedrich. Deterministic random walks on the two-dimensional grid. *Combinatorics, Probability & Computing*, 18(1-2):123–144, 2009.

[77] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[78] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar. Optimal constrained graph exploration. *ACM Transactions on Algorithms*, pp. 380–402, 2006.

[79] J.-P. Duval. Relationship between the period of a finite word and the length of its unbordered segments. *Discrete Mathematics*, 40(1):31–44, 1982.

[80] K. Efremenko and O. Reingold. How well do random walks parallelize? In *APPROX-RANDOM*, LNCS vol. 5687, pp. 476–489, 2009.

[81] A. Elitzur and L. Vaidman. Quantum mechanical interaction-free measurements. *Foundations of Physics*, 23(7):987–997, July 1993.

[82] S. Elouasbi and A. Pelc. Time of anonymous rendezvous in trees: Determinism vs. randomization. In G. Even and M. M. Halldórsson, editors, *SIROCCO*, Lecture Notes in Computer Science vol. 7355, pp. 291–302. Springer, 2012.

[83] R. Elsässer and T. Sauerwald. Tight bounds for the cover time of multiple random walks. In *ICALP (1)*, LNCS vol. 5555, pp. 415–426, 2009.

[84] U. Feige. A spectrum of time-space trade-offs for undirected s-t connectivity. *J. Comput. Syst. Sci.*, 54(2):305–316, 1997.

[85] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro. Searching for black holes in subways. *Theory Comput. Syst.*, 50(1):158–184, 2012.

[86] P. Flocchini, E. Kranakis, D. Krizanc, F. L. Luccio, and N. Santoro. Sorting and election in anonymous asynchronous rings. *J. Parallel Distrib. Comput.*, 64(2):254–265, 2004.

[87] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in a ring. In *Proceedings of 6th Latin American Symposium on Theoretical Informatics (LATIN)*, LNCS vol. 2976, pp. 599–608, 2004.

[88] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan and Claypool, 2012.

[89] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005.

[90] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theor. Comput. Sci.*, 407(1-3):412–447, 2008.

[91] A. S. Fraenkel. Economic traversal of labyrinths. *Mathematics Magazine*, 43:125–130, 1970.

[92] P. Fraigniaud, L. Gąsieniec, D. R. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.

[93] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

[94] P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory. In *Proceedings of 22nd International Conference on Distributed Computing (DISC)*, LNCS vol. 5218, pp. 242–256, 2008.

[95] P. Fraigniaud and A. Pelc. Delays induce an exponential memory gap for rendezvous in trees. In *Proceedings of 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 224–232, 2010.

[96] S. Gal. Rendezvous search on the line. *Operations Research*, 47(6):974–976, 1999.

[97] L. Gasieniec, R. Klasing, R. Martin, A. Navarra, and X. Zhang. Fast periodic graph exploration with constant memory. *Journal of Computer and System Sciences*, 74(5):802–822, 2008.

[98] L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *WG*, Lecture Notes in Computer Science vol. 5344, pp. 14–29, 2008.

[99] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: a case study of unbiased sampling of osns. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pp. 2498–2506, Piscataway, NJ, USA, 2010. IEEE Press.

[100] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Perform. Eval.*, 63(3):241–263, Mar. 2006.

[101] R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *ICALP (2)*, Lecture Notes in Computer Science vol. 5556, pp. 484–495. Springer, 2009.

[102] S. Guilbault and A. Pelc. Asynchronous rendezvous of anonymous agents in arbitrary graphs. In A. F. Anta, G. Lipari, and M. Roy, editors, *OPODIS*, Lecture Notes in Computer Science vol. 7109, pp. 421–434. Springer, 2011.

[103] N. Hanusse, D. Ilcinkas, A. Kosowski, and N. Nisse. Locating a target with an agent guided by unreliable local advice: how to beat the random walk when you have a clock? In A. W. Richa and R. Guerraoui, editors, *PODC*, pp. 355–364. ACM, 2010.

[104] N. Hanusse, E. Kranakis, and D. Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004.

[105] J. M. Hendrickx. Depth of views on symmetric networks: a new bound for guaranteeing equality. Technical report, 2012.

[106] C. H. Horvat and M. Stoffregen. A solution to the lonely runner conjecture for almost all points. Technical report, 2011. arXiv:1103.1662.

[107] S. Ikeda, I. Kubo, N. Okumoto, and M. Yamashita. Fair circulation of a token. *IEEE Trans. Parallel Distrib. Syst.*, 13(4):367–372, 2002.

[108] S. Ikeda, I. Kubo, and M. Yamashita. The hitting and cover times of random walks on finite graphs using local degree information. *Theoretical Computer Science*, 410(1):94–100, 2009.

[109] D. Ilcinkas. Setting port numbers for fast graph exploration. *Theoretical Computer Science*, 401(1-3):236–242, 2008.

[110] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In C. Dwork, editor, *PODC*, pp. 119–131. ACM, 1990.

[111] H. Kakugawa and M. Yamashita. Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. In *SRDS*, pp. 202–211. IEEE Computer Society, 2002.

[112] S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Gathering an even number of robots in an odd ring without global multiplicity detection. In B. Rovan, V. Sassone, and P. Widmayer, editors, *MFCS*, Lecture Notes in Computer Science vol. 7464, pp. 542–553. Springer, 2012.

[113] D. R. Karger, N. Nisan, and M. Parnas. Fast connected components algorithms for the erew pram. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, SPAA '92, pp. 373–381, New York, NY, USA, 1992. ACM.

[114] A. Kawamura and Y. Kobayashi. Fence patrolling by mobile agents with distinct speeds. In K.-M. Chao, T. sheng Hsu, and D.-T. Lee, editors, *ISAAC*, Lecture Notes in Computer Science vol. 7676, pp. 598–608. Springer, 2012.

[115] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pp. 137–146, New York, NY, USA, 2003. ACM.

[116] L. M. Kirousis and C. H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, 1985.

[117] R. Klasing, A. Kosowski, and A. Navarra. Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. *Theor. Comput. Sci.*, 411(34-36):3235–3246, 2010.

[118] R. Klasing, A. Kosowski, D. Pajak, and T. Sauerwald. The multi-agent rotor-router on the ring: A deterministic alternative to parallel random walks. In *PODC 2013*, to appear, 2013.

[119] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. *Theor. Comput. Sci.*, 390(1):27–39, 2008.

[120] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theor. Comput. Sci.*, 384(2-3):201–221, 2007.

[121] J. M. Kleinberg. The flow of on-line information in global networks. In A. K. Elmagarmid and D. Agrawal, editors, *SIGMOD Conference*, pp. 1–2. ACM, 2010.

[122] S. Koenig. Complexity of edge counting. Technical Report Goal-Directed Acting with Incomplete Information. CMU-CS-97-199, Carnegie Mellon University, 1997.

[123] S. Koenig and R. G. Simmons. Easy and hard testbeds for real-time search algorithms. In *Proceedings of 13th National Conference on Artificial Intelligence (AAAI)*, pp. 279–285, 1996.

[124] A. Korman, J.-S. Sereni, and L. Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. In C. Gavoille and P. Fraigniaud, editors, *PODC*, pp. 49–58. ACM, 2011.

[125] A. Kosowski, B. Li, N. Nisse, and K. Suchan. k-chordal graphs: From cops and robber to compact routing via treewidth. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, editors, *ICALP (2)*, Lecture Notes in Computer Science vol. 7392, pp. 610–622. Springer, 2012.

[126] A. Kosowski, A. Navarra, and M. C. Pinotti. Synchronous black hole search in directed graphs. *Theor. Comput. Sci.*, 412(41):5752–5759, 2011.

[127] A. Kosowski and D. Pajak. Fast randomized graph exploration: When a little bit of memory makes all the difference. Technical report, 2013.

[128] M. Koucký. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002.

[129] D. R. Kowalski and A. Malinowski. How to meet in anonymous network. *Theoretical Computer Science*, 399(1-2):141–156, 2008.

[130] E. Kranakis, D. Krizanc, and E. Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Morgan and Claypool, 2010.

[131] E. Kranakis, D. Krizanc, and S. Rajsbaum. *Computing with Mobile Agents in Distributed Networks*. CRC Press, 2001.

[132] E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Mobile agent rendezvous in a ring. In *Proceedings of 23rd International Conference on Distributed Computing Systems (ICDCS)*, pp. 592–599, 2003.

[133] E. Kranakis, D. Krizanc, and J. van der Berg. Computing Boolean functions on anonymous networks. *Information and Computation*, 114(2):214–236, 1994.

[134] M. Kurant, M. Gjoka, C. T. Butts, and A. Markopoulou. Walking on a graph with a magnifying glass: stratified sampling via weighted random walks. *SIGMETRICS Perform. Eval. Rev.*, 39(1):241–252, June 2011.

[135] A. Lamani, M. G. Potop-Butucaru, and S. Tixeuil. Optimal deterministic ring exploration with oblivious asynchronous robots. In B. Patt-Shamir and T. Ekim, editors, *SIROCCO*, Lecture Notes in Computer Science vol. 6058, pp. 183–196. Springer, 2010.

[136] C.-H. Lee, X. Xu, and D. Y. Eun. Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling. *SIGMETRICS Perform. Eval. Rev.*, 40(1):319–330, June 2012.

[137] W. Lim and S. Alpern. Minimax rendezvous on the line. *SIAM Journal on Control and Optimization*, 34(5):1650–1665, 1996.

[138] J. Littlewood and B. Bollobas. *Littlewood's Miscellany*. Cambridge University Press, 1986.

[139] L. Lovász. Random walks on graphs: A survey. *Bolyai Society Mathematical Studies*, 2:353–397, 1996.

[140] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pp. 1109–1117, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[141] N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch. Distributed token circulation in mobile ad hoc networks. *IEEE Trans. Mob. Comput.*, 4(2):154–165, 2005.

[142] G. D. Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

[143] P. Matthews. Covering problems for brownian motion on spheres. *Ann. Probab.*, 16:189–199, 1988.

[144] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[145] R. Montenegro. The simple random walk and max-degree walk on a directed graph. *Random Struct. Algorithms*, 34(3):395–407, May 2009.

[146] N. Nisan. Rl $\subseteq$ sc. In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, editors, *STOC*, pp. 619–623. ACM, 1992.

[147] Y. Nonaka, H. Ono, K. Sadakane, and M. Yamashita. The hitting and cover times of metropolis walks. *Theor. Comput. Sci.*, 411(16-18):1889–1894, 2010.

[148] N. Norris. Universal covers of graphs: Isomorphism to depth N-1 implies isomorphism to all depths. *Discrete Applied Mathematics*, 56(1):61–74, 1995.

[149] H. Ono. Fast random walks on finite graphs and graph topological information. In *ICNC*, pp. 360–363. IEEE Computer Society, 2011.

[150] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. Technical report, 1999.

[151] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999.

[152] P. Panaite and A. Pelc. Impact of topographic information on graph exploration efficiency. *Networks*, 36(2):96–103, 2000.

[153] A. Pelc. Disc 2011 invited lecture: Deterministic rendezvous in networks: Survey of models and results. In D. Peleg, editor, *DISC*, Lecture Notes in Computer Science vol. 6950, pp. 1–15. Springer, 2011.

[154] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[155] V. B. Priezzhev, D. Dhar, A. Dhar, and S. Krishnamurthy. Eulerian walkers as a model of selforganized criticality. *Physics Review Letters*, 77(25):5079–5082, 1996.

[156] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, 2008.

[157] O. Reingold, L. Trevisan, and S. P. Vadhan. Pseudorandom walks on regular digraphs and the rl vs. l problem. In J. M. Kleinberg, editor, *STOC*, pp. 457–466. ACM, 2006.

[158] K. Roh, M. Crochemore, C. S. Iliopoulos, and K. Park. External memory algorithms for string problems. *Fundam. Inform.*, 84(1):17–32, 2008.

[159] H.-A. Rollik. Automaten in planaren Graphen. *Acta Inf.*, 13:287–298, 1980.

[160] T. Sauerwald. Expansion and the cover time of parallel random walks. In *PODC*, pp. 315–324. ACM, 2010.

[161] T. Schelling. *The strategy of conflict*. Oxford University Press, Oxford, 1960.

[162] S. Souissi, X. Defago, and M. Yamashita. Gathering asynchronous mobile robots with inaccurate compasses. In A. A. Shvartsman, editor, *OPODIS*, Lecture Notes in Computer Science vol. 4305, pp. 333–349. Springer, 2006.

[163] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.

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

[165] L. Thomas. Finding your kids when they are lost. *Journal of the Operational Research Society*, 43(6):637–639, 1992.

[166] W. T. Tutte and C. A. B. Smith. On unicursal paths in a network of degree 4. *The American Mathematical Monthly*, 48(4):233–237, 1941.

[167] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Smell as a computational resource — a lesson we can learn from the ant. In *Proceedings of 4th Israel Symposium on Theory of Computing and Systems (ISTCS)*, pp. 219–230, 1996.

[168] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15(5):918–933, 1999.

[169] Wikipedia. Elitzur-Vaidman bomb tester — Wikipedia, The Free Encyclopedia, 2013. [Online; accessed 8-July-2013].

[170] P. Winkler and D. Zuckerman. Multiple cover time. *Random Structures and Algorithms*, 9(4):403–411, 1996.

[171] M. Yamashita and T. Kameda. Computing functions on asynchronous anonymous networks. *Mathematical Systems Theory*, 29(4):331–356, 1996. Erratum: Theory of Computing System 31(1): 109 (1998).

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

[173] M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Trans. Parallel Distrib. Syst.*, 10(9):878–887, 1999.

[174] M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.*, 411(26-28):2433–2453, 2010.

[175] V. Yanovski, I. A. Wagner, and A. M. Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.

[176] X. Yu and M. Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In *Proceedings of 23rd International Colloquium on Automata, Languages and Programming (ICALP)*, LNCS vol. 1099, pp. 610–621, 1996.

[177] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni. A local algorithm for finding well-connected clusters. *CoRR*, abs/1304.8132, 2013.

Attached publications

The offprints of publications [T1–T8] are attached, in order of reference.