A Journey through Dynamic Networks (with Excursions)

Arnaud Casteigts

June 4, 2018

Rapport scientifique relatif à l'obtention d'une

Habilitation à Diriger des Recherches

Présenté devant le jury composé de:

Rapporteur(e)s	Antonio Fernández Anta	DR. IMDEA-Networks (Madrid)
	Clémence Magnien	DR. CNRS-LIP6 (Paris)
	Sébastien Tixeuil	PU. Sorbonne Université-LIP6 (Paris)
Examinateurs	Emmanuel Godard	PU. Univ Aix-Marseille-LIS (Marseille)
	Nicolas Hanusse	DR. CNRS-LaBRI (Bordeaux)
	Philippe Jacquet	DR. Bell-Labs Nokia (Paris-Saclay)
Invité	Joseph Peters	PU. Simon Fraser University (Vancouver)

Preamble

This document is presented with the aim to obtain the "Habilitation à Diriger des Recherches" (hereafter, *habilitation*), a French degree that enables, among other things, the official supervision of PhD students and application to full professorship in the French system. Typical requirements include effective experience in the supervision of PhD students, significant contributions to one or several research areas, and a dissertation surveying these contributions – this document.

Habilitation dissertations are not subject to the same standardization as PhD dissertations. Their writing is regarded as a personal exercise, with little constraints put on formatting (none, in our case). As a result, dissertations vary significantly, both in length and style, and sometimes in purpose. An evident goal is to subject one's work to an evaluation committee. The contributions of a researcher being already characterized (to a reasonable extent) by his/her publications and curriculum vitæ, a recent trend has been to make these essays shorter, giving the candidate an opportunity to showcase synthesis abilities (together with distance and perspective). Perhaps as importantly, the habilitation represents a unique opportunity to revisit and compile one's personal work into a consistent and self-contained document. It legitimates the writing of a document that would otherwise appear as resolutely self-centered, yet so useful to its author if it condenses most techniques and results from the work being reviewed. That second objective goes beyond evaluation, and it is in slight contradiction with conciseness.

In this document, we try to follow a middle path, reviewing chosen parts in details and scanning through the rest more succinctly. The first part offers a complete (yet quintessential) survey of our contributions to the domain of *highly-dynamic networks*, a central and recurrent topic in our research. This part, of about sixty pages, can be seen as an independent dissertation covering four chapters, to be possibly adapted later as a monograph. The second part, half as long, is also organized as a collection of four chapters, each of which covers a set of topics to which we have contributed (sometimes very modestly). While this part is significantly shorter, it covers about the same amount of work as the first part. We made it more succinct because these topics are less central to us and mostly disjoint (unlike the first part, which forms a unified theory). Some of the content in that part also corresponds to series of works which are mainly terminated. Both parts are presented in more details next.

Content of this document

The journey recounted in this document spanned a period of about 10 years, from the end of the PhD (2007) to current (early 2018). The first five years correspond to two postdocs made at the University of Ottawa (Canada), in two different research groups: The first under the supervision of Amiya Nayak and Ivan Stojmenovic; the second under Paola Flocchini and Nicola Santoro, interspersed with short research experiences in other institutions. The rest of the period correspond to the past five years spent at the University of Bordeaux as *Maître de Conférences* (assistant professor). A schematic view of the corresponding timeline together with main topics is given in Figure 1. The document is organized in two parts, each of which drawing content from *both* periods (upper and lower parts of Figure 1, respectively).

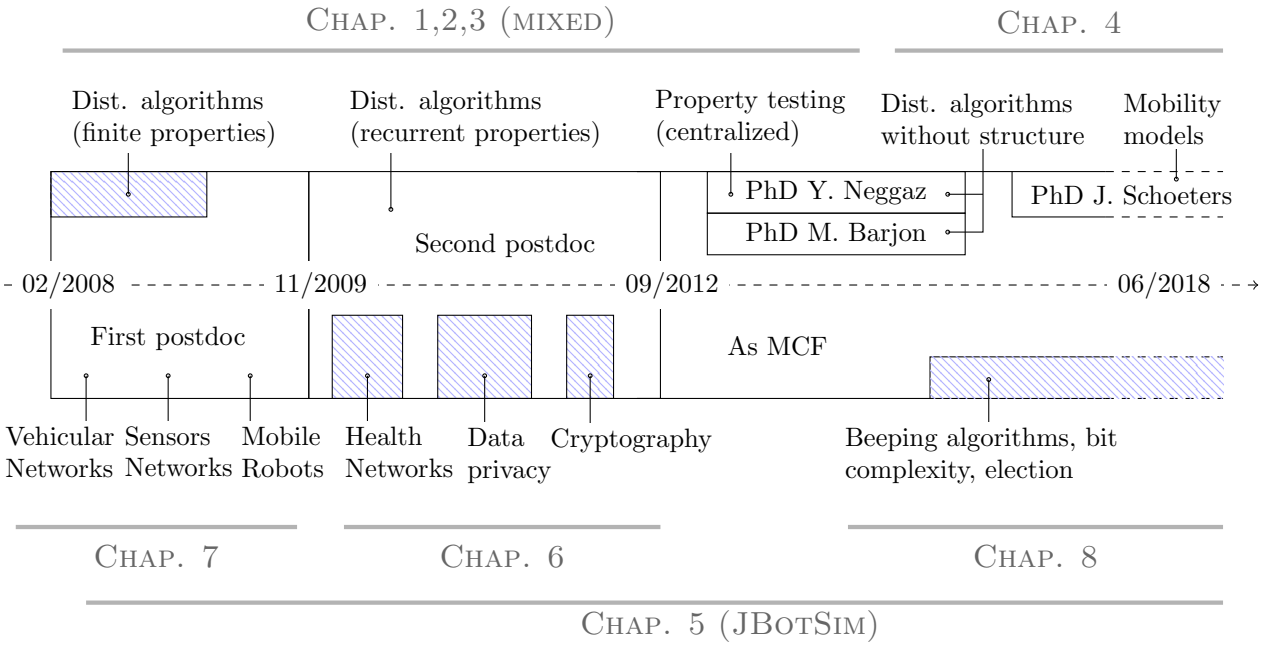
Part I

The first part entitled "Finding Structure in Dynamic Networks" surveys our contributions in the area of highly-dynamic networks. The underlying motivation in this line of work was to compile, organize, and propose concepts and tools that allow a more formal investigation of highly-dynamic networks, a domain which until recently was mainly studied from an applied (empirical) perspective. The content of this part considers both distributed and centralized point of views, unifying some of their common preoccupation under the notion of *structure*. Indeed, highly-dynamic networks often look chaotic at first, but many of them offer a weak (temporal) form of structure that distributed algorithms can exploit (and centralized algorithm can test).

The first chapter starts by mentioning technological contexts (and research topics) in which the study of dynamic networks emerges naturally. While not comprehensive, this gives the reader a foretaste of the variety of these contexts. Then, we present review a number of graph-theoretical representations and temporal concepts which have been used to describe dynamic networks, with a focus on our own contributions and the time-varying graph (TVG) language. The opportunity is taken to pinpoint a strong conceptual unity behind the apparent diversity of models. Next, we discuss the impact that the temporal dimension has had on the very definition of classical problems (whether distributed or combinatorial), many of which are at the basis of the work presented subsequently. This chapter can be viewed as a global introduction.

The second chapter is concerned with the interaction between topological structure and distributed algorithms. Necessary or sufficient conditions for distributed algorithms are classically defined in terms of communication models, knowledge, or structural properties (*i.e.,* restricted classes of graphs). In a dynamic network, the topology changes *during* the computation, and this has a dramatic effect on computations. The impact of these changes onto the feasibility of basic distributed tasks has acted as a general guiding principle behind many of our contributions, leading to the definition of a number of classes of dynamic graphs capturing the requirement of various algorithms in terms of network evolution or interaction (*i.e.,* temporal properties). Some of these properties have a *finite* essence and others a *recurrent* essence, each corresponding to a different period in our research (see Figure 1).

Part I: Finding Structure in Dynamic Networks



Part II: Excursions

Figure 1: Timeline of main activities since the PhD (2007).

In Chapter 3, we get some distance from distributed computing and consider the intrinsic features of (dynamic) graph classes, together with others from the literature. In particular, we review their inter-relations from a set-theoretic point of views, the largest part of which is adapted from one of our article on this topic. We further develop some of the discussions from that article, perhaps with more hindsight, and initiate new discussions as well, connecting this topic with a range of other topics. In particular, a section is devoted to testing temporal properties on given traces of dynamic graphs; another (more prospective) section is devoted to connecting temporal properties to real-world mobility contexts. We conclude with an opening on the emerging topic of discrete movement synthesis for entities that control their own movements (*e.g.,* drones and robots), these movements being possibly tuned to force the resulting network to satisfy desired properties *by design* (ongoing PhD thesis by J. Schoeters).

Finally, Chapter 4 reviews several of our contributions not directly related to the presence of structure in dynamic networks, and yet fully integrated within the same lines of work. In fact, some of them relate explicitly to the *absence* of structure. We first review a series of works targeting the maintenance of a distributed spanning forest in highly-dynamic networks without assumptions on the network dynamics. Next, we present a distributed algorithm called T-Clocks, which makes it possible to measure temporal distances in a continuous-time setting. While this algorithm requires no particular structure for working, it has interesting applications when the network is periodic. Then, we present a theoretical investigation of the expressivity of *time-varying graphs* in terms of automata and languages, in which we show that being able to buffer

information at intermediate nodes (along journeys) has a dramatic impact on the expressivity of the network. We conclude with a personal collection of tidbits (small facts) which we have gleaned over the years and where the dynamic feature seems to have an impact on the very nature of things.

Part II

The second part of this document, entitled "Excursions", recounts our contributions in a number of disjoint topics. The level of detail in this part varies depending on the chapters. Because this document is also intended to be a working document, we devote more detail to those chapters covering topics which are still current to us (Chap. 5 and 6). The last two chapters (Chap. 7 and 8) cover topics which, although of great interest to us, are less actual and summarized more succinctly.

In Chapter 5, we give an overview of the JBotSim library, a tool that we have developed over the past nine years. JBotSim offers basic primitives for prototyping, running, and visualizing distributed algorithms in dynamic networks. We give a few examples of code and use cases, together with a discussion of its main features. While JBotSim was a personal tool over the first few years, we recently opened it to the community and it has a burgeoning ecosystem, briefly presented in the chapter.

In Chapter 6, we review two recent contributions whose targets are static networks. Their common feature is to consider weak distributed communication models where the size and the nature of messages are very limited. In the first contribution, we showed that the bit round complexity of the election problem in arbitrary networks (with identifiers) is $\Theta(D + \log n)$, breaking a long-standing barrier at $O(D \log n)$. In the second, we present a number of design patterns for beeping algorithms, illustrated through several examples of algorithms whose time complexity is analyzed (one of the analyses improves that of the running time of the best known MIS algorithm).

In Chapter 7, we review the main research topics we were involved in during the first postdoc period at the University of Ottawa (2008-09, see Figure 1). The topics are covered in a chronological order. They include vehicular networks, wireless sensor networks, and networks of mobile robots. The chapter also reviews a joint work with another research group on bluetooth networks, the early discussions of which occurred in the end of that period and which fits well within the broad topic of wireless networks.

Finally, Chapter 8 reviews several short-term research missions realized intermittently over the second postdoc period (2010-12). The first consisted of designing a social inquiry together with a sociologist, Louise Bouchard, injecting temporal graph concepts into the study of Canadian health networks. The second mission was commissionned by a physician from the Montfort hospital research institute, Marie-Hélène Chomienne. It involved privacy issues for health data, whose legal setting in Canada is quite specific and for which we applied techniques from the area of differential privacy. The third mission was done together with a cryptographer from the mathematics department of the University of Ottawa in a security-leading company called *Irdeto*, for which we developed a prototype incorporating new white box cryptography protocols.

How to read this document

As explained above, the scope of this document goes slightly beyond evaluation. We intend to use it as a working tool in the coming years, and as such, we tried to record all essential ideas involved in Part I, sometimes through a single sentence. To the possible extent, we tried to avoid unnecessary technical details, aiming at a pleasant (and possibly linear) reading by any colleague with some maturity in graph theory and algorithms. Part II is written in a more narrative form and it may occasionally recount experiences using the first person. This part is more personal and may be of lower interest to a random reader.

The publications associated to every section are generally listed below the title of that section. The introductory wording of a chapter puts some of these publications in context. The names of co-authors are generally mentioned upon the first occurrence of a citation, then "we" is used as a standard rule. No dedicated chapter is devoted to perspectives or future works. However, we disseminate a number of open questions and research avenues along the document (using a joint numbering), suggesting ramifications to the exposed work. We do not intend to pursue all of them ourselves, thus some are "open" for pick-up by the interested readers (who may however want to contact us to check that no one else is currently working on the same problem).

Summary of publications

The work surveyed in this document corresponds to about 25 conference articles (excluding publications done before 2007) published in the proceedings of various venues, including DISC, PODC, OPODIS, SIROCCO, IPDPS, EUROPAR, FCT, and ADHOC-NOW. The main part is also covered by thirteen journal articles, ten of which are published and three are under review. The published ones are in IEEE Transactions on Computer Science, Theoretical Computer Science (Elsevier), The Computer Journal (Oxford Press), Computer Communications (Elsevier), Theory of Computing System (Springer), Int. J. of Foundations of Computer Science (World Scientific), Int. J. of Parallel, Emergent and Distributed Systems (Taylor and Francis), Wireless Networks (Springer), Wireless Communication and Mobile Computing (Wiley), and Global Health Promotion (Sage Pub).

This work was first and foremost a collective adventure. The corresponding publications were co-authored by a number of colleagues, teachers, and friends (with various delimitations), to whom I am grateful. They include, in alphabetical order, Frédéric Amblard, Jérémie Albert, Matthieu Barjon, Lionel Barrère, Jean-Marie Berthelot, Louise Bouchard, Mariette Chartier, Serge Chaumette, Marie-Hélène Chomienne, Swan Dubois, Afonso Ferreira, Paola Flocchini, Ahmed Jedda, Colette Johnen, Guy-Vincent Jourdan, Emmanuel Godard, Nishith Goel, Carlos Gómez-Calzado, Frédéric Guinand, Ralf Klasing, Alberto Lafuente, Mikel Larrea, Bernard Mans, Luke Mathieson, Hussein Mouftah, Yves Métivier, Amiya Nayak, Yessin Neggaz, Joseph Peters, Franck Petit, Yoann Pigné, Walter Quattrociocchi, Mike Robson, Nicola Santoro, Jason Schoeters, Ivan Stojmenovic, Alain Trugeon, Jan Warnke, Mark Yamashita, and Akka Zemmari.

Acknowledgments

Candidates of the habilitation have the privilege to compose their own evaluation committee (with safeguards). I chose to have this document reviewed by Clémence Magnien, Antonio Fernández Anta, and Sébastien Tixeuil. Their own work connect in various ways to the one exposed here. I am grateful for their feedback on various aspects, in particular concerning the hierarchy of classes of dynamic networks (*e.g.,* finite representation, terminology, connections with topology). I am also thankful to Emmanuel Godard, Nicolas Hanusse, Philippe Jacquet, and Joseph Peters for accepting to serve as examiners and for the many insightful questions during the defense. Their presence in the committee was due to their own work and perhaps also to a certain form of freethinking that inspires me.

The work reported in this document is the result of a collective adventure. I am grateful to my co-adventurers (listed on Page 5). Many of these persons have had a tremendous influence on me. In particular, I am deeply grateful to Paola Flocchini and Nicola Santoro, for the chance they offered me in 2009 (in the nick of time) and for being such wonderful collaborators; and to Yves Métivier for his advice, generosity, and constant efforts to focus on the substance behind the form. On a different note, I admire those of us who dare reading more (and writing less), being knowledgeable and happy to share the knowledge disinterestedly – among others, these comments aim at David Renault and Joseph Peters.

I had the pleasure to guide (and be guided by) three PhD students so far, Yessin M. Neggaz, Matthieu Barjon, and Jason Schoeters. The first two were co-supervised with Colette Johnen and Serge Chaumette, and also resulted in a local collaboration with Ralf Klasing, all of whom I thank. I thank Jason for carefully reading an earlier version of this document and for the many constructive comments he sent me back. A significant part of the material in this document has been taught to students in the "Algorithmique de la Mobilité" graduate class at UB, with invaluable feedback from the students over the years. I am also thankful to my colleagues of the "Institut Universitaire de Technologie" (IUT), for the unique atmosphere of friendship and solidarity that we enjoy in this teaching department, and to the members of the "Laboratoire Bordelais de Recherche en Informatique" (LaBRI) – administrative staff, researchers, and students alike, with a special mention to the people of the Atrium.

To my family and friends; to Lucile, Clémentine, and Suzanne, with all my love.

Contents

I Finding Structure in Dynamic Networks **11**

1 Dynamic networks? **13**

1.1 A variety of contexts 13

1.2 Graph representations 14

1.3 Some temporal concepts 18

1.4 Redefinition of problems 21

2 Feasibility of distributed problems **25**

2.1 Basic conditions 25

2.2 Shortest, fastest, and foremost broadcast 29

2.3 Bounding the temporal diameter 32

2.4 Minimal structure and robustness 35

3 Around classes of dynamic networks **39**

3.1 List of classes 39

3.2 Relations between classes 46

3.3 Testing properties automatically 49

3.4 From classes to movements (and back) 55

4 Beyond structure **59**

4.1 Spanning forest without assumptions 59

4.2 Measuring temporal distances distributedly 62

4.3 The power of waiting 65

4.4 Collection of curiosities 68

II Excursions **73**

5 JBotSim **75**

5.1 Introduction 75

5.2 Versatility of use cases 77

5.3 Community 78

6 Bit complexity and related models **81**

6.1 Bit round complexity of leader election 81

6.2 Design patterns in beeping algorithms 83

7 Wireless networks (first postdoc) **87**

7.1 Vehicular networks 87

7.2 Multiratecast in wireless sensor networks 88

7.3 Biconnecting mobile robots 89

7.4 Sensors and actuators networks (book chapters) 90

7.5 Bluetooth Scatternet Formation 91

8 Social sciences, human sciences, data privacy, security **93**

8.1 Study of health networks in Canada 93

8.2 Differential Privacy for Health Data in Canada 94

8.3 White-Box Elliptic Curve Diffie-Hellman 97

Part I

Finding Structure in Dynamic Networks

Chapter 1

Dynamic networks?

Behind the terms "dynamic networks" lies a rich diversity of contexts ranging from near-static networks with occasional changes, to networks where changes occur continuously and unpredictably. In the past two decades, these highly-dynamic networks gave rise to a profusion of research activities, resulting (among others) in new concepts and representations based on graph theory. This chapter reviews some of these emerging notions, with a focus on our own contributions. The content also serves as a definitional resource for the rest of the document, restricting ourselves mostly to the notions effectively used in the subsequent chapters.

1.1 A variety of contexts

A network is traditionally defined as a set of entities together with their mutual relations. It is dynamic if these relations *change* over time. There is a great variety of contexts in which this is the case. Here, we mention two broad categories, communication networks and complex systems, which despite an important overlap, usually capture different (and complementary) motivations.

(Dynamic) communication networks. These networks are typically made of wireless-enabled entities ranging from smartphones to laptops, to drones, robots, sensors, vehicles, satellites, *etc.* As the entities move, the set of communication links evolve, at a rate that goes from occasional (*e.g.,* laptops in a managed Wi-Fi network) to nearly unrestricted (*e.g.,* drones and robots). Networks with communication faults also fall in this category, albeit with different concerns (on which we shall return later).

(Dynamic) complex networks. This category comprises networks from a larger range of contexts such as social sciences, brain science, biology, transportation, and communication networks (as a particular case). An extensive amount of real-world data is becoming available in all these areas, making it possible to characterize various phenomena.

While the distinction between both categories may appear somewhat artificial, it shed some light to the typically different motivations underlying these areas. In particular, research in communication networks is mainly concerned with what can be done *from within* the network through distributed interactions among the entities, while research in complex networks is mainly concerned with defining mathematical models

that capture, reproduce, or predict phenomena observed in reality, based mostly on the analysis of available data in which *centralized* algorithms play the key role.

Interactions between both are strong and diverse. However, it seems that for a significant period of time, both communities (and sub-communities) remained essentially unaware of their respective effort to develop a conceptual framework to capture the network dynamics using graph theoretical concepts. In way of illustration, let us mention the diversity of terminologies used for even the most basic concepts in dynamic networks, *e.g.,* that of *journey* [42], also called *schedule-conforming path* [32], *time-respecting path* [138], and *temporal path* [84, 183]; and that of *temporal distance* [42], also called *reachability time* [127], *information latency* [141], *propagation speed* [132] and *temporal proximity* [142].

In the rest of this chapter, we review some of these efforts in a chronological order. Next, we present some of the main temporal concepts identified in the literature, with a focus on the ones used in this document. The early identification of these concepts, in a unifying attempt, was one of the components of our most influencial paper so far [68]. Finally, we present a general discussion as to some of the ways these new concepts impact the definition of combinatorial (or distributed) problems classically studied in static networks, some of which are covered in more depth in subsequent chapters.

1.2 Graph representations

Standard graphs. The structure of a network is classically given by a *graph*, which is a set of vertices (or nodes) V and a set of edges (links) $E \subseteq V \times V$, *i.e.,* pairs of nodes. The edges, and by extension the graph itself, may be *directed* or *undirected*, depending on whether the relations are unidirectional or bidirectional (symmetrical). Figure 1.1 shows an example of undirected graph. Graph theory is a central topic in discrete mathematics. We refer the reader to standard books for a general introduction (*e.g.,* [185]), and we assume some acquaintance of the reader with basic concepts like paths, distance, connectivity, trees, cycles, cliques, *etc.*

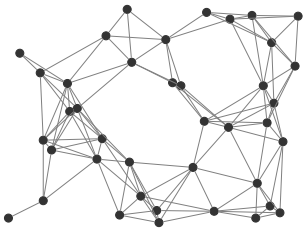


Figure 1.1: A graph depicting communication links in a wireless network.

1.2.1 Graph models for dynamic networks

Dynamic networks can be modeled in a variety of ways using graphs and related notions. Certainly the first approach that comes to mind is to represent a dynamic network as a *sequence* of standard (static) graphs as depicted in Figure 1.2. Each graph of the sequence (*snapshot*, in this document) represents the relations among vertices at a given discrete time. This idea is natural, making it difficult to trace back its first occurrence in the literature. Sequences of graphs were studied (at least) back in the 1980s (see, *e.g.,* [181, 115, 173, 103]) and simply called dynamic graphs. However, in a subtle way, the graph properties considered in these studies still refer to individual snapshots, and typical problems consist of updating standard information about the current snapshot,

like connectivity [173].

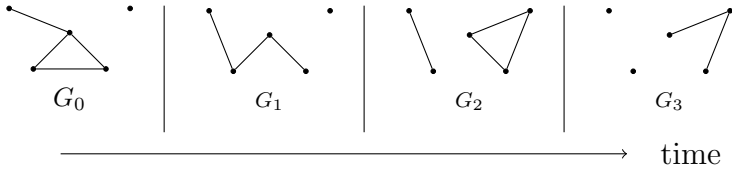


Figure 1.2: A dynamic network seen as a sequence of graphs

A *conceptual shift* occurred when researchers started to consider the whole sequence as being *the* graph. Then, properties of interest became related to temporal features of the network rather than to its punctual structure. In the distributed computing community, one of the first work in this direction was that of Awerbuch and Even in 1984 [15], who considered the broadcasting problem in a network that satisfies a temporal version of connectivity. On the modeling side, one of the first work considering explicitly a graph sequence as being *the* graph seem to be due to Harary and Gupta [125]. Then, a number of graph models were introduced to describe dynamic networks, reviewed here in chronological order.

In 2000, Kempe, Kleinberg, and Kumar [138] defined a model called *temporal network*, where the network is represented by a graph G (the *footprint*, in this document) together with a labeling function $\lambda : E \to \mathbb{R}$ that associates to every edge a number indicating when the two corresponding endpoints communicate. The model is quite basic: the communication is punctual and a single time exist for every edge. Both limitations can be circumvented by considering multiple edges and artificial intermediate vertices slowing down communication.

Independently, Ferreira and Viennot [109] represented a dynamic network by a graph G together with a matrix that describes the *presence schedule* of edges (the same holds for vertices). The matrix is indexed in one dimension by the edges and in the other by integers that represent time. The authors offer an alternative point of view as a sequence of graphs where each graph $G_i(V_i, E_i)$ correspond to the 1-entries of the matrix, this sequence being called an *evolving graph*, and eventually renamed as *untimed* evolving graph in subsequent work (see below).

Discussion 1 (Varying the set of vertices)**.** Some graph models consider a varying set of vertices in addition to a varying set of edges, some do not. From a formal point of view, all models can easily be adapted to fit in one or the other category. In the present document, we are mostly interested in *edge* dynamics, thus we will ignore this distinction most of the time (unless it matters in the context). Also note that an absent node could sometimes be simulated by an isolated node using only edge dynamics.

A similar statement as that of Discussion 1 could be made for *directed* edges versus *undirected* edges. We call on the reader's flexibility to ignore non essential details in the graph models when these details are not important. We state them explicitly when they are.

Ferreira and his co-authors [108, 42, 107] further generalize evolving graphs by considering a sequence of graph $G_i = \{G_1, G_2, ...\}$ where each G_i may span a *non unitary* period of time. The duration of each G_i is encoded in an auxiliary table of times $t_1, t_2, ...$

such that every G_i spans the period $[t_i, t_{i+1})$.[1] A consequence is that the times can now be taken from the real domain, with mild restrictions pertaining to theoretical limitations (*e.g.,* countability or accumulation points). In order to disambiguate both versions of evolving graphs, the earlier version from [109] is qualified as *untimed* evolving graphs. Another contribution of [42] is to incorporate the latency (*i.e.,* time it takes to cross a given edge at a given time) through a function ζ called *traversal time* in [42].

In 2009, Kostakos [142] describes a model called *temporal graphs*, in which the whole dynamic graph is encoding into a single static (directed) graph, built as follows. Every entity (vertex) is duplicated as many times as there are time steps, then directed edges are added between consecutive copies of the same node. One advantage of this representation is that some temporal features become immediately available on the graph, *e.g.,* directed paths correspond to non-strict journeys (defined further down). Recently, the term "temporal graph" has also been used to refer to the temporal networks of [138], with some adaptations (see *e.g.,* [5]). This latter usage of the term seems to become more common than the one from [142].

In the complex network community, where researchers are concerned with the effective manipulation of data, different models of dynamic networks have emerged whose purpose is essentially to be used as a *data structure* (as opposed to being used only as a descriptive language, see Discussion 2 below). In this case, the network history is typically recorded as a sequence of timed links (*e.g.,* time-stamped e-mail headers in [141]) also called link streams. The reader is referred to [101, 141, 189, 128] for examples of use of these models, and to [150] for a recent survey.

As part of a unifying effort with Flocchini, Quattrociocchi, and Santoro from 2009 to 2012, we reviewed a vast body of research in dynamic networks [68], harmonizing concepts and attempting to bridge the gap between the two (until then) mostly disjoint communities of complex systems and networking (distributed computing). Taking inspiration from several of the above models, we defined a formalism called TVG (for *time-varying graphs*) whose purpose was to favor expressivity and elegance over all practical aspects such as implementability, data structures, and efficiency. The resulting formalism is free from intrinsic limitation.

Let V be a set of entities (vertices, nodes) and E a set of relations (edges, links) between them. These relations take place over an interval $\mathcal{T}$ called the *lifetime* of the network, being a subset of $\mathbb{N}$ (discrete) or $\mathbb{R}^+$ (continuous), and more generally some time domain $\mathbb{T}$. In its basic version, a *time-varying graph* is a tuple $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$ such that

- $\rho : E \times \mathcal{T} \to \{0, 1\}$, called *presence* function, indicates if a given edge is available at a given time.
- $\zeta : E \times \mathcal{T} \to \mathbb{T}$, called *latency* function, indicates the time it takes to cross a given edge at a given start time (the latency of an edge could itself vary in time).

The latency function is optional and other functions could equally be added, such as a *node presence* function $\psi : V \times \mathcal{T} \to \{0, 1\}$, a *node latency* function $\varphi : V \times \mathcal{T} \to \mathbb{T}$ (accounting *e.g.,* for local processing times), *etc.*

[1] For French readers: $[a, b)$ is an equivalent notation to $[a, b[$.

Discussion 2 (Model, formalism, language)**.** TVGs are often referred to as a *formalism*, to stress the fact that using it does not imply restrictions on the environment, as opposed to the usual meaning of the word *model*. However, the notion of formalism has a dedicated meaning in mathematics related to formal logic systems. With hindsight, we see TVGs essentially as a *descriptive language*. In this document, the three terms are used interchangeably.

The purely descriptive nature of TVGs makes them quite general and allows temporal properties to be easily expressible. In particular, the presence and latency functions are not *a priori* constrained and authorize theoretical constructs like accumulation points or uncountable 0/1 transitions. While often not needed in complex systems and offline analysis, this generality is relevant in distributed computing, *e.g.,* to characterize the power of an adversary controlling the environment (see [60] and Section 4.3 for details).

Visual representation. Dynamic networks can be *depicted* in different ways. One of them is the sequence-based representation shown above in Figure 1.2. Another is a *labeled graph* like the one in Figure 1.3, where labels indicate when an edge is present (either as intervals or as discrete times). Other representations include chronological diagrams of contacts (see *e.g.,* [128, 119, 150]).

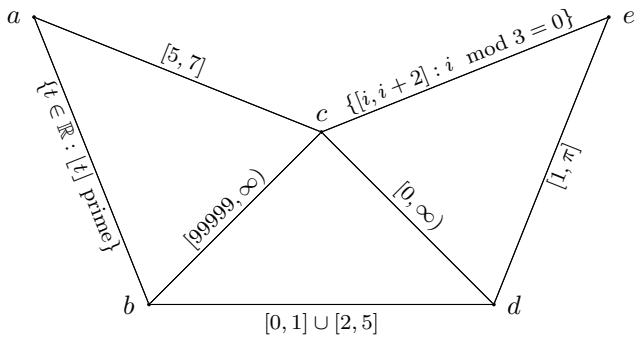


Figure 1.3: A dynamic network represented by a labeled graph. The labels indicates when the corresponding edges are present.

1.2.2 Moving the focus away from models (a plea for unity)

Up to specific considerations, the vast majority of temporal concepts transcend their formulation in a given graph model (or stream model), and the same holds for many algorithmic ideas. Of course, some models are more relevant than others depending on the uses. In particular, is the model being used as a data structure or as a descriptive language? Is time discrete or continuous? Is the point of view local or global? Is time synchronous or asynchronous? Do links have duration? Having several models at our disposal is a good thing. On the other hand, the diversity of terminology makes it harder for several sub-communities to track the progress made by one another. We hope that the diversity of models does not prevent us from acknowledging each others *conceptual* works accross communities.

In this document, most of the temporal concepts and algorithmic ideas being reviewed are independent from the model. Specific models are used to *formulate* these concepts, but once formulated, they can often be considered at a more general level. To stress independence from the models, we tend to refer to a graph representing a dynamic network as just a *graph* (or a network), using calligraphic letters like $\mathcal{G}$ or $\mathcal{H}$ to indicate their dynamic nature. In contrast, when referring to static graphs in a way not clear from the context, we add the adjective *static* or *standard* explicitly and use regular letters like G and H.

1.3 Some temporal concepts

Main articles: ADHOCNOW'11 [67] (long version IJPEDS'12 [68]).

This section presents a number of basic concepts related to dynamic networks. Many of them were independently identified in various communities using different names. We limit ourselves to the most central ones, alternating between time-varying graphs and untimed evolving graphs (*i.e.,* basic sequences of graphs) for their formulation (depending on which one is the most intuitive). Most of the terminology is in line with the one of our 2012 article [68], in which a particular effort was made to identify the first use of each concept in the literature and to give proper credit accordingly. Most of these concepts are now becoming folklore, and we believe this is a good thing.

Subgraphs

There are several ways to restrict a dynamic network $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$. Classically, one may consider a subgraph resulting from taking only a subset of V and E, while maintaining the behavior of the presence and latency functions, specialized to their new domains. Perhaps more specifically, one may restrict the lifetime to a given sub-interval $[t_a, t_b] \subseteq \mathcal{T}$, specializing again the functions to this new domain without otherwise changing their behavior. In this case, we write $\mathcal{G}_{[t_a,t_b]}$ for the resulting graph and call it a *temporal* subgraph of $\mathcal{G}$.

Footprints and Snapshots

Given a graph $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, the *footprint* of $\mathcal{G}$ is the standard graph consisting of the vertices and edges which are present at least once over $\mathcal{T}$. This notion is sometimes identified with the *underlying graph* $G = (V, E)$, that is, the *domain* of possible vertices and edges, although in general an element of the domain may not appear in a considered interval, making the distinction between both notions useful. We denote the footprint of a graph $\mathcal{G}$ by $\mathit{footprint}(\mathcal{G})$ or simply $\cup(\mathcal{G})$. The *snapshot* of $\mathcal{G}$ at time t is the standard graph $G_t = (V, \{e : \rho(e, t) = 1\})$. The footprint can also be defined as the union of all snapshots. Conversely, we denote by $\cap(\mathcal{G})$ the intersection of all snapshots of $\mathcal{G}$, which may be called *intersection graph* or *denominator* of $\mathcal{G}$. Observe that, so defined, both concepts make sense as well in discrete time as in continuous time. In a context of infinite lifetime, Dubois et al. [40] defined the *eventual footprint* of $\mathcal{G}$ as the graph (V, E') whose edges reappear infinitely often; in other words, the *limsup* of the snapshots.

In the literature, snapshots have been variously called layers, graphlets, or instantaneous graphs; footprints have also been called underlying graphs, union graphs, or induced graphs.

Journeys and temporal connectivity

The concept of *journey* is central in highly dynamic networks. Journeys are the analogue of paths in standard graphs. In its simplest form, a journey in $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$ is a sequence of ordered pairs $\mathcal{J} = \{(e_1, t_1), (e_2, t_2) \ldots, (e_k, t_k)\}$, such that $e_1, e_2, ..., e_k$ is a walk in (V, E), $\rho(e_i, t_i) = 1$, and $t_{i+1} \geq t_i$. An intuitive representation is shown on Figure 1.4. The set of all journeys from u to v when the context of $\mathcal{G}$ is clear is denoted by $\mathcal{J}^*_{(u,v)}$.

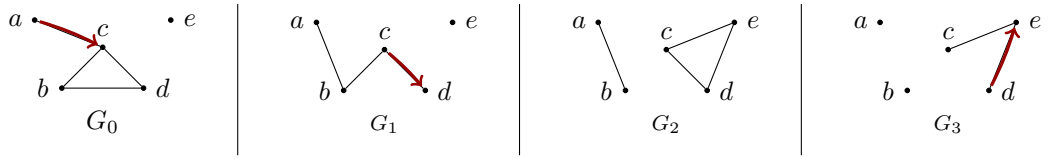


Figure 1.4: Intuitive representation of a journey (from a to e) in a dynamic network.

Several versions can be formulated, for example taking into account the latency by requiring that $t_{i+1} \geq t_i + \zeta(e_i, t_i)$. In a communication network, it is often also required that $\rho(e_i, t) = 1$ for all $t \in [t_i, t_i + \zeta(e_i, t_i))$, *i.e.,* the edge remains present during the communication period. When time is discrete, a more abstract way to incorporate latency is to distinguish between *strict* and *non-strict* journeys [138], strictness referring here to requiring that $t_{i+1} > t_i$ in the journey times. In other words, non-strict journeys correspond to neglecting latency.

Somewhat orthogonally, a journey is *direct* if every next hop occurs without delay at the intermediate nodes (*i.e.,* $t_{i+1} = t_i + \zeta(e_i, t_i)$); it is *indirect* if it makes a pause at least at one intermediate node. We showed that this distinction plays a key role for computing temporal distances among the nodes in *continuous* time (see [63, 65], reviewed in Section 4.2). We also showed, using this concept, that the ability of the nodes to buffer a message before retransmission decreases dramatically the expressive power of an adversary controlling the topology (see [59, 61, 60], reviewed in Section 4.3).

Finally, $departure(\mathcal{J})$ and $arrival(\mathcal{J})$ denote respectively the starting time t_1 and the last time t_k (or $t_k + \zeta(e_k, t_k)$ if latency is considered) of journey $\mathcal{J}$. When the context of $\mathcal{G}$ is clear, we denote the *possibility* of a journey from u to v by $u \rightsquigarrow v$, which does not imply $v \rightsquigarrow u$ even if the links are undirected, because time induces its own level of direction (*e.g.,* $a \rightsquigarrow e$ but $e \not\rightsquigarrow a$ in Figure 1.4). If forall u and v, it holds that $u \rightsquigarrow v$, then $\mathcal{G}$ is *temporally connected* (Class $\mathcal{TC}$ in Chapter 3). Interestingly, a graph may be temporally connected even if none of its snapshots are connected, as illustrated in Figure 1.5 (the footprint must be connected, though).

The example on Figure 1.5 suggests a natural extension of the concept of connected components for dynamic networks, which we discuss in a dedicated paragraph in Section 4.4.

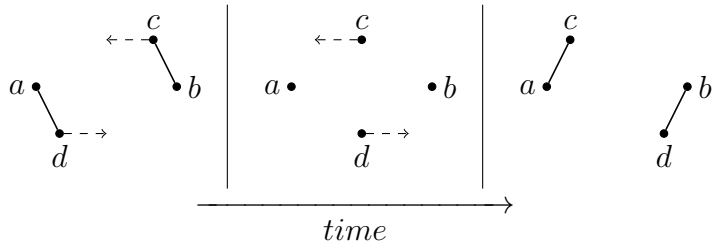


Figure 1.5: Connectivity over time and space. *(Dashed arrows denote movements.)*

Temporal distance and related metrics

As observed in 2003 by Bui-Xuan *et al.* [42], journeys have both a *topological* length (number of hops) and a *temporal* length (duration), which gives rise to several concepts of distance and at least three types of optimal journeys (*shortest*, *fastest*, and *foremost*, covered in Section 2.2 and 4.2). Unfolding the concept of temporal distance leads to that of temporal diameter and temporal eccentricity [42]. Precisely, the temporal eccentricity of a node u at time t is the earliest time that u can reach all other nodes through a journey starting after t. The temporal diameter at time t is the maximum among all nodes eccentricities (at time t). Another characterization of the temporal diameter (at time t) is the smallest d such that $\mathcal{G}_{[t,t+d]}$ is temporally connected. These concepts are central in some of our contributions (*e.g.,* [63, 65], further discussed in Section 4.2). Interestingly, all these parameters refer to time quantities, and these quantities themselves vary over time, making their study (and computation) more challenging.

1.3.1 Further concepts

The number of definitions built on top of temporal concepts could grow large. Let us mention just a few additional concepts which we had compiled in [68, 177] and [51]. Most of these emerged in the area of complex systems, but are of general applicability.

Small-world. A temporal analogue of the small-world effect is defined in [183] based on the *duration* of journeys (as opposed to hop distance in the original definition in static graphs [188]). Perhaps without surprise, this property is observed in a number of more theoretical works considering stochastic dynamic networks (see *e.g.,* [84, 92]). An analogue of *expansion* for dynamic networks is defined in [92].

Network backbones. A temporal analogue of the concept of *backbone* was defined in [141] as the "subgraph consisting of edges on which information has the potential to flow the quickest." In fact, we observe in [65] that an edge belongs to the backbone relative to time t iff it is used by a foremost journey starting at time t. As a result, the backbone consists exactly of the union of all foremost broadcast trees relative to initiation time t (the computation of such structure is reviewed in Section 4.2).

Centrality. The structure (*i.e.,* footprint) of a dynamic network may not reflect how well interactions are balanced within. In [68], we defined a metric of *fairness* as the standard deviation among temporal eccentricities. In the caricatural example of Figure 1.6 (depicting weekly interaction among entities), node c or d are structurally more central, but node a is actually the most central in terms of temporal eccentricity:

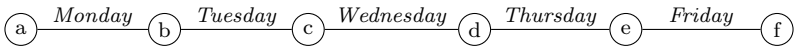


Figure 1.6: Weekly interactions between six people (from [68]).

it can reach all other nodes within 5 to 11 days, compared to nearly three weeks for d and more than a month for f.

Together with a sociologist, Louise Bouchard, in 2010 [51], we proposed to apply these measures (together with a stochastic version of network backbones) to the study of health networks in Canada. (The project was not retained and we started collaborating on another topic.) This kind of concepts, including also *temporal betweenness* or *closeness* (which we expressed in the TVG formalism in [177]), received a lot of attention lately (see *e.g.,* [169, 174]).

Other temporal or dynamic extensions of traditional concepts, not covered here, include *treewidth* [158], *temporal flows* [4], and characteristic temporal distance [183]. Several surveys reviewed the conceptual shift induced by time in dynamic networks, including (for the distributed computing community) Kuhn and Oshman [147], Michail and Spirakis [161], and our own 2012 survey with Flocchini, Quattrociocchi, and Santoro [68].

1.4 Redefinition of problems

Main articles: arXiv'11 [73], DRDC reports'13 [57, 58], IJFCS [66].

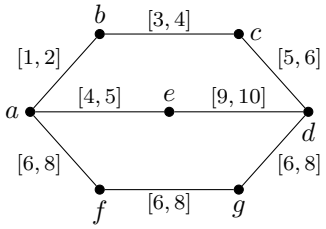
The fact that a network is dynamic has a deep impact on the kind of tasks one can perform within. This impact ranges from making a problem harder, to making it impossible, to redefining the metrics of interest, or even change the whole definition of the problem. In fact, many standard problems *must* be redefined in highly dynamic networks. For example, what is a *spanning tree* in a partitioned (yet temporally connected) network? What is a maximal independent set, and a dominating set? Deciding which definition to adopt depends on the target application. We review here a few of these aspects through a handful of problems like *broadcast*, *election*, *spanning trees*, and classic *symmetry-breaking* tasks (independent sets, dominating sets, vertex cover), on which we have been involved.

1.4.1 New optimality metrics in broadcasting

As explained in Section 1.3, the length of a journey can be measured both in terms of the number of hops or in terms of duration, giving rise to two distinct notions of distance among nodes: the *topological distance* (minimum number of hop) and the *temporal distance* (earliest reachability), both being relative to a source u, a destination v, *and* an initiation time t.

Bui-Xuan, Ferreira, and Jarry [42] define three optimality metrics based on these notions: *shortest* journeys minimize the number of hops, *foremost* journeys minimize

reachability time, and *fastest* journeys minimize the duration of the journey (possibly delaying its departure). See Figure 1.7 for an illustration.



Optimal journeys from a to d (starting at time 0):

- the shortest: a-e-d (only two hops)
- the foremost: a-b-c-d (arriving at $5 + \zeta$)
- the fastest: a-f-g-d (no intermediate waiting)

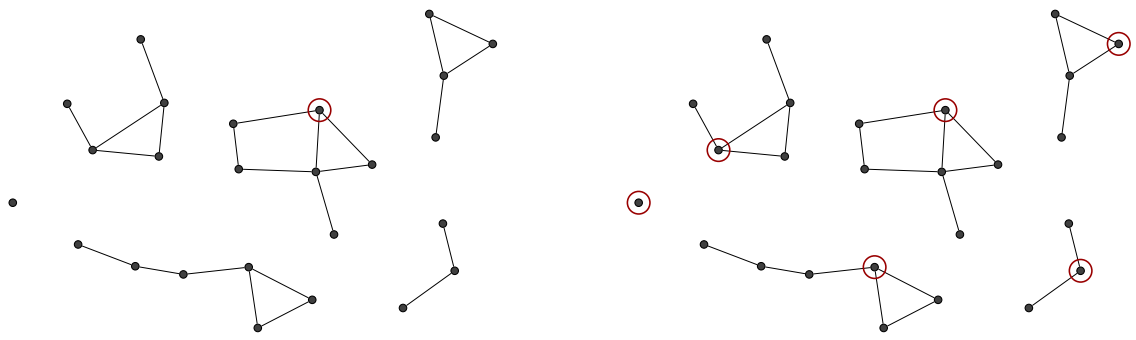
Figure 1.7: Different meanings for length and distance *(assuming latency $\zeta \ll 1$)*.

The *centralized* problem of computing the three types of journeys given full knowledge of the graph $\mathcal{G}$ is introduced (and algorithms are proposed) in [42]. We investigated a distributed analogue of this problem; namely, the ability for the nodes to broadcast according to these metrics without knowing the underlying networks, but with various assumptions about its dynamics [62, 66] (reviewed in Section 2.2).

1.4.2 Election and spanning trees

While the definition of problems like broadcast or routing remains intuitive in highly dynamic networks, other problems are definitionally ambiguous. Consider *leader election* and *spanning trees*. In a static network, leader election consists of distinguishing a single node, the leader, for playing subsequently a distinct role. The spanning tree problem consists of selecting a cycle-free set of edges that interconnects all the nodes. Both problems are central and widely studied. How should these problems be defined in a highly dynamic network which (among other features) is possibly partitioned most of the time?

At least two (somewhat generic) versions emerge. Starting with *election*, is the objective still to distinguish a *unique* leader? This option makes sense if the leader can influence other nodes reasonably often. However, if the temporal connectivity within the network takes a long time to be achieved, then it may be more relevant to elect a leader in *each* component, and maintain a leader per component when components merge and split. Both options are depicted on Figure 1.8.



(a) A single global leader

(b) One leader per connected component

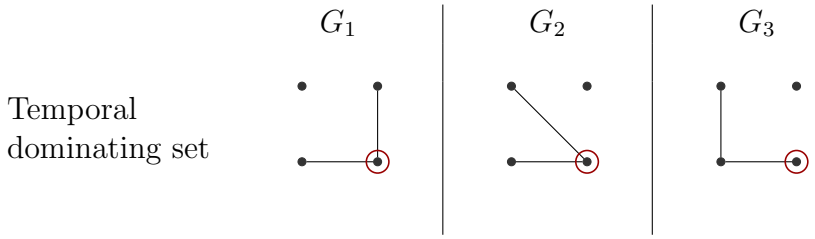
Figure 1.8: Two possible definitions of the leader election problem.

The same declination holds for spanning trees, the options being (1) to build a unique tree whose edges are *intermittent*, or (2) to build a different tree in each component, to be updated when the components split and merge. Together with Flocchini, Mans, and Santoro, we considered the first option in [62, 66] (reviewed in Section 2.2), building (distributedly) a fixed but intermittent broadcast tree in a network whose edges are all recurrent (Class $\mathcal{E}^{\mathcal{R}}$). In a different line of work (with a longer list of co-authors) [45, 168, 53, 24], we proposed and studied a "best effort" principle for maintaining a set of spanning trees of the second type, while guaranteeing that some properties hold *whatever* the dynamics (reviewed in Section 4.1). A by-product of this algorithm is to maintain a single *leader* per tree (the root).

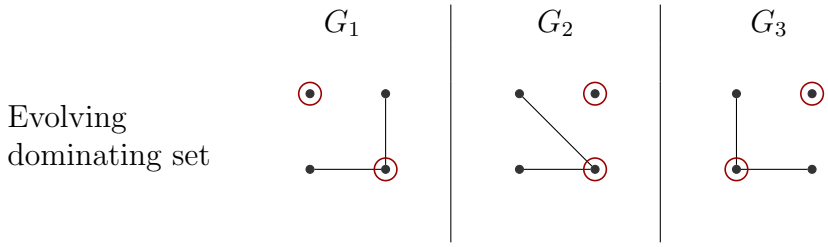
1.4.3 Covering problems

With Mans and Mathieson in 2011 [73], we explored three canonical ways of redefining combinatorial problems in highly-dynamic networks, with a focus on covering problems like *dominating set*. A dominating set in a (standard) graph $G = (V, E)$ is a subset of nodes $S \subseteq V$ such that each node in the network either is in S or has a neighbor in S. The goal is usually to minimize the size (or cost) of S. Given a dynamic network $\mathcal{G} = \{G_1, G_2, \dots\}$, the problem admits three natural declinations:

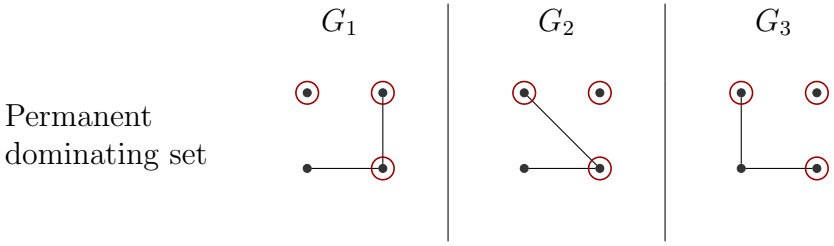
- *Temporal* version: domination is achieved *over time* – every node outside the set must share an edge *at least once* with a node in the set, as illustrated below.



- *Evolving* version: domination is achieved in every snapshot, but the set can vary between them. This version is commonly referred to as "dynamic graph algorithms" in the algorithmic literature (also called "reoptimization").



- *Permanent* version: domination is achieved in every snapshot, with a *fixed* set.



The three versions are related. Observe, in particular, that the temporal version consists of computing a dominating set in $\cup\mathcal{G}$ (the footprint), and the permanent version one in $\cap\mathcal{G}$. Solutions to the permanent version are valid (but possibly sub-optimal) for the evolving version, and those for the evolving version are valid (but possibly sub-optimal) for the temporal version [73]. In fact, the solutions to the *permanent* and the *temporal* versions actually form upper and lower bounds for the *evolving* version. Note that the permanence criterion may *force* the addition of some elements to the solution, which explains why some problems like spanning tree and election do not admit a permanent version.

Open question 1. *Make a more systematic study of the connexions between the temporal, the evolving, and the permanent versions. Characterize the role these versions play with respect to each other both in terms of lower bound and upper bound, and design algorithms exploiting this triality ("threefold duality").*

In a distributed (or online) setting, the permanent and the temporal versions are not directly applicable because the future of the network is not known a priori. Nonetheless, if the network satisfies other forms of regularity, like *periodicity* [112, 110, 137] (Class $\mathcal{E}^{\mathcal{P}}$) or *recurrence of edges* [62] (Class $\mathcal{E}^{\mathcal{R}}$), then such solutions can be built despite lack of information about the future. In this perspective, Dubois *et al.* [100] define a variant of the temporal version in infinite lifetime networks, requiring that the covering relation holds *infinitely often* (*e.g.,* for dominating sets, every node not in the set must be dominated infinitely often by a node in the set). We review in Section 2.4 a joint work with Dubois, Petit, and Robson [56], where we define a new hereditary property in graphs called *robustness* that captures the ability for a solution to have such features in a large class of highly-dynamic networks (Class $\mathcal{TC}^{\mathcal{R}}$, all classes are reviewed in Chapter 3). The robustness of maximal independent sets (MIS) is investigated in particular and the locality of finding a solution in various cases is characterized.

This type of problems have recently gained interest in the algorithmic and distributed computing communities. For example, Mandal et al. [157] study approximation algorithms for the permanent version of *dominating sets*. Akrida *et al.* [6] (2018) define a variant of the temporal version in the case of vertex cover, in which a solution is not just a set of nodes (as it was in [73]) but a set of pairs $(nodes, times)$, allowing different nodes to cover the edges at different times (and within a sliding time window). Bamberger *et al.* [20] (2018) also define a temporal variant of two covering problems (vertex coloring and MIS) relative to a sliding time window.

Concluding remark

Given the reporting nature of this document, we reviewed here the definition of concepts and problems in which we have been effectively involved (through contributions). As such, the content does not aim at comprehensiveness and many concepts and problems were not covered. In particular, the *network exploration* problem received a lot of attention recently in the context of dynamic networks, for which we refer the reader to a number of dedicated works [113, 111, 130, 37, 155].

Chapter 2

Feasibility of distributed problems

A common approach to analyzing distributed algorithms is the characterization of necessary and sufficient conditions to their success. These conditions commonly refer to the communication model, synchronicity, or structural properties of the network (*e.g.,* is the topology a tree, a grid, a ring, *etc.*) In a dynamic network, the topology changes *during* the computation, and this has a dramatic effect on computations. In order to understand this effect, the engineering community has developed a number of *mobility models*, which govern how the nodes move and make it possible to compare results on a relatively fair basis and enable their reproducibility (the most famous, yet unrealistic model is *random waypoint*).

In the same way as mobility models enable the *experimental* investigations of algorithms and protocols in highly dynamic networks, logical properties on the network dynamics, *i.e.,classes of dynamic graphs*, have the potential to guide a more formal exploration of their qualities and limitations. This chapter reviews our contributions in this area, which has acted as a general driving force through most of our works in dynamic networks. The resulting classes of dynamic graphs are then revisited, compiled, and independently discussed in Chapter 3.

2.1 Basic conditions

Main article: SIROCCO'09 [52]

In way of introduction, consider the broadcasting of a piece of information in the dynamic network depicted on Figure 2.1. The ability to complete this task depends on which node is the initial emitter: a and b may succeed, while c is guaranteed to fail. The obvious reason is that no journey (thus no chain of causality) exists from c to a.

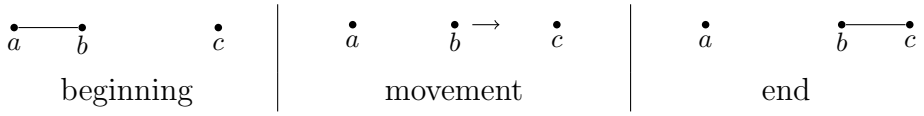


Figure 2.1: A basic mobility scenario.

Together with Chaumette and Ferreira in 2009 [52], we identified a number of such

requirements relative to a few basic tasks, namely broadcasting, counting, and election. For the sake of abstraction, the algorithms were described in a high-level model called *graph relabelings systems* [152], in which interactions consist of *atomic* changes in the state of neighboring nodes. Different scopes of action were defined in the literature for such models, ranging from a single edge to a closed star where the state of all vertices changes atomically [85, 86]. The *expression* of algorithms in the edge-based version is close to that of population protocols [12], but the context of execution is different (especially in our case), and perhaps more importantly, the type of *questions* which are investigated are different.

In its simplest form, the broadcasting principle is captured by a single rule, represented as $I\!-\!N \rightarrow I\!-\!I$ and meaning that if an informed node "interacts" (in a sense that will become clear) with a non-informed node, then the latter becomes informed. This node can in turn propagate information through the same rule. Graph relabeling systems[1] typically take place over a fixed graph, which unlike the interaction graph of population protocols, *is* thought of as a static network. In fact, the main purpose of graph relabeling systems is to abstract *communications*, while that of population protocols is to abstract *dynamism* (through scheduling). In our case, dynamism is not abstracted by the scheduling; relabeling operations take place over a support graph that itself changes, which is fundamentally different and makes it possible to inject properties of the network dynamics into the analysis.

The model. Given a dynamic network $\mathcal{G} = \{G_1, G_2, ...\}$, the model we proposed in [52][2] considers relabeling operation taking place *on top of the sequence*. Precisely, every G_i may support a number of interactions (relabelings), then at some point, the graph transitions from G_i to G_{i+1}. The adversary (scheduler, or daemon) controls both the selection of edges on which interactions occur, and the moment when the system transitions from G_i to G_{i+1}, subject to the constraint that every edge of every G_i is selected *at least once* before transitioning to G_{i+1}. This form of fairness is reasonable, as otherwise some edges may be present or absent without incidence on the computation. The power of the adversary mainly resides in the *order* in which the edges are selected and the number of times they are selected in each G_i. The adversary does *not* control the sequence of graph itself (contrary to common models of message adversaries).

2.1.1 Necessary or sufficient conditions in terms of dynamics

The above model allows us to define formally the concept of necessary and sufficient conditions for a given algorithm, in *terms of network dynamics*. For a given algorithm $\mathcal{A}$ and network $\mathcal{G}$, $\mathcal{X}$ is the set of all possible executions and $X \in \mathcal{X}$ one of them corresponding to the adversary choices.

Definition 1 (Necessary condition)**.** *A property P (on a graph sequence) is* necessary *(in the context of $\mathcal{A}$) if its non-satisfaction on $\mathcal{G}$ implies that no sequence of relabelings can transform the initial states to desired final states. ($\neg P(\mathcal{G}) \implies \forall X \in \mathcal{X}, \mathcal{A}$ fails.)*

[1] This model is sometimes referred to as "local computations", but with a different meaning to that of Naor and Stockmeyer [162], therefore we do not use this term.

[2] In [52], we relied on the general (*i.e.,* timed) version of evolving graphs. With hindsight, untimed evolving graphs, *i.e.,* basic sequences of graphs, are sufficient and make the description simpler.

Definition 2 (Sufficient condition)**.** *A property P (on graph sequences) is* sufficient *if its satisfaction on $\mathcal{G}$ implies that all possible sequences of relabelings take the initial states into desired final states. ($P(\mathcal{G}) \implies \forall X \in \mathcal{X}, \mathcal{A}$ succeeds.)*

In other words, if a necessary condition is not satisfied by $\mathcal{G}$, then the execution will fail *whatever* the choices of the adversary; if a sufficient condition is satisfied, the execution will succeed whatever the choices of the adversary. In between lies the actual power of the adversary. In the case of the afore-mentioned broadcast algorithm, this space is not empty: it is *necessary* that a journey exist from the emitter to all other nodes, and it is *sufficient* that a *strict* journey exist from the emitter to all other nodes.

Discussion 3 (Sensitivity to the model)**.** *By nature, sufficient conditions are dependent on additional constraints imposed to the adversary, namely here, of selecting every edge at least once. No property on the sequence of graph could,* alone*, guarantee that the nodes will effectively interact, thus sufficient conditions are intrinsically* model-sensitive*. On the other hand, necessary conditions on the graph evolution do not depend on the particular model, which is one of the advantages of considering high-level computational model without abstracting dynamism.*

We considered three other algorithms in [52] which are similar to some protocols in [12], albeit considered here in a different model and with different questions in mind. The first is a counting algorithm in which a designated counter node increments its count when it interacts with a node for the first time ($\overset{k}{\bullet}\!\!-\!\!\overset{N}{\bullet} \longrightarrow \overset{k+1}{\bullet}\!\!-\!\!\overset{F}{\bullet}$). An obvious necessary condition to count all nodes is the existence of a direct edge between the counter and every other node (possibly at the same time or at different times). In fact, this condition is also sufficient in the considered model, leaving no power at all to the adversary: either the condition holds and success is guaranteed, or it does not and failure is certain.

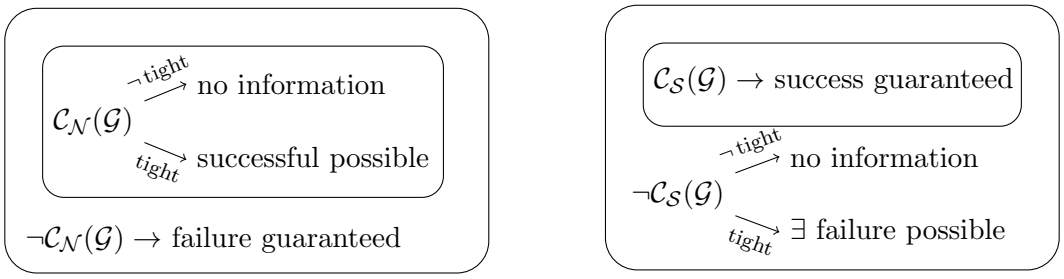
The second counting algorithm has *uniform* initialization: every node has a variable initially set to 1. When two nodes interact, one of them cumulates the count of the other, which is eliminated ($\overset{i}{\bullet}\!\!-\!\!\overset{j}{\bullet} \longrightarrow \overset{i+j}{\bullet}\!\!-\!\!\overset{0}{\bullet}$). Here, a necessary condition to complete the process is that at least one node can be reached by all others through a journey. A sufficient condition due to Marchand de Kerchove and Guinand [159] is that all pairs of nodes interact *at least once* over the execution (*i.e.,* the footprint of $\mathcal{G}$ is complete). The third counting algorithm adds a circulation rule to help surviving counters to meet, with same necessary condition as before.

Open question 2. *Find a sufficient condition for this version of the algorithm in terms of network dynamics.*

2.1.2 Tightness of the conditions

Given a condition (necessary or sufficient), an important question is whether it is *tight* for the considered algorithm. Marchand de Kerchove and Guinand [159] defined a *tightness* criterion as follows. Recall that a necessary condition is one whose non-satisfaction implies failure; it is *tight* if, in addition, its satisfaction does make success *possible* (*i.e.,* a nice adversary *could* make it succeed). Symmetrically, a sufficient condition is tight if

its non-satisfaction does make failure possible (the adversary can make it fail). This is illustrated on Figure 2.2.



(a) Necessary condition $\mathcal{C}_N$ (b) Sufficient condition $\mathcal{C}_S$

Figure 2.2: Tightness of conditions.

It was observed [159] that all of the conditions in [52] are tight. Interestingly, this implies that, while the adversary has no power at all in the case of the first counting algorithm, it has a lot in the case of the second.

2.1.3 Relating conditions to graph classes

Each of the above conditions naturally induces a *class* of dynamic networks in which the corresponding property is satisfied. In fact, if a node plays a distinguished role in the algorithm (non-uniform initialization), then at least two classes of graphs are naturally induced by each property, based on quantification (one existential, one universal). These classes correspond to graphs in which ...

- ... at least one node can reach all the others through a journey $(1 \rightsquigarrow *)$,
- ... all nodes can reach each other through journeys $(* \rightsquigarrow *)$,
- ... at least one node shares at some point an edge with every other $(1 - *)$,
- ... all pairs of nodes share at some point an edge $(* - *)$,
- ... at least one node can be reached by all others through a journey $(* \rightsquigarrow 1)$,
- ... at least one node can reach all the others through a strict journey $(1 \overset{st}{\rightsquigarrow} *)$,
- ... all nodes can reach each other through strict journeys $(* \overset{st}{\rightsquigarrow} *)$.

These classes were assigned an "$\mathcal{F}_i$" number in [46, 52] and for some a distinct "$\mathcal{C}_i$" number in [68]. We revisit all the classes and their names in Chapter 3, and we review algorithms for testing membership of a given graph to each of them.

2.1.4 Towards formal proofs

As explained above, one of the advantages of working at a high level of abstraction with atomic communication (here, graph relabelings) is that impossibility results obtained in these models are general, *i.e.*,, they apply automatically to lower-level models like message passing. Another important feature of such models is that they are well suited for formalization, and thereby for formal proofs.

From 2009 to 2012 [82, 83], Castéran *et al.* developed an early set of tools and methodologies for formalizing graph relabeling systems within the framework of the *Coq* proof assistant, materializing as the Loco library. More recently, Corbineau *et al.* developed the Padec library [8], which allows one to build certified proofs (again with *Coq*) in a computational model called *locally shared memory model with composite atomicity*, much related to graph relabeling systems. This library is being developed intensively and may in the mid term incorporate other models and features. Besides the *Coq* realm, recent efforts were made by Fakhfakh *et al.* to prove the correctness of algorithms in dynamic networks based on the Event-B framework [104, 105]. These algorithms are formalized using graph relabeling systems (in particular, one of them is our *spanning forest* algorithm presented in Section 4.1). Another work pertaining to certifying distributed algorithms for mobile robots in the *Coq* framework, perhaps less directly relevant here, is that of Balabonski *et al.* [19].

Research avenue 3. *Building on top of these plural (and related) efforts, formalize in the framework of Coq or Event-B the main objects involved in this section, namely sequences of graphs, relabeling algorithms, and the fairness condition for edge selection. Use them to prove* formally *that a given assumption on the network dynamics is necessary or sufficient to a given algorithm.*

2.2 Shortest, fastest, and foremost broadcast

Main articles: IFIP-TCS'10 [62] and IJFCS'15 [66]

We reviewed in Section 1.4 different ways in which the time dimension impacts the formulation of distributed and combinatorial problems. One of them is the declination of optimal journeys into three metrics: shortest, fastest, and foremost defined in [42] in a context of centralized offline problems. In a series of work with Flocchini, Mans, and Santoro [62, 66], we studied a distributed version of these problems, namely shortest, fastest, and foremost *broadcast* with termination detection at the emitter, in which the evolution of the network is not known to the nodes, but must obey various forms of regularities (*i.e.,* be in various classes of dynamic networks). Some of the findings were surprising, for example the fact that the three variants of the problems require a gradual set of assumptions, each of which is strictly included in the previous one.

2.2.1 Broadcast with termination detection (TDB)

The problem consists of broadcasting a piece of information from a given source (emitter) to all other nodes, with termination detection at the source (TDB, for short). Only the broadcasting phase is required to be optimal, not the termination phase. The metrics were adapted as follows:

- TDB[$foremost$]: every node is informed at the earliest possible time,
- TDB[*shortest*]: the number of hops relative to every node is minimized,
- TDB[$fastest$]: the time between first emission and last reception is globally minimized.

These requirements hold relative to a given initiation time t, which is triggered externally at the initial emitter. Since the schedule of the network is not known in advance, we examine what minimal *regularity* in the network make TDB feasible, and so for each metric. Three cases are considered:

- Class $\mathcal{E}^{\mathcal{R}}$ (*recurrent* edges): graphs whose *footprint is connected* (not necessarily complete) and every edge in it re-appears infinitely often. In other words, if an edge is available once, then it will be available recurrently.
- Class $\mathcal{E}^{\mathcal{B}} \subset \mathcal{E}^{\mathcal{R}}$ (*bounded-recurrent* edges): graphs in which every edge of the footprint cannot be absent for more than Δ time, for a fixed Δ.
- Class $\mathcal{E}^{\mathcal{P}} \subset \mathcal{E}^{\mathcal{B}}$ (*periodic* edges) where every edge of the footprint obeys a periodic schedule, for some period p. (If every edge has its own period, then p is their least common multiple.)

As far as *inclusion* is concerned, it holds that $\mathcal{E}^{\mathcal{P}} \subset \mathcal{E}^{\mathcal{B}} \subset \mathcal{E}^{\mathcal{R}}$ and the containment is strict. However, we show that being in either class only helps if additional *knowledge* is available. The argument appears in different forms in [62, 66] and proves quite ubiquitous – let us call it a "late edge" argument.

Late edge argument. If an algorithm is able to decide termination of the broadcast in a network $\mathcal{G}$ by time t, then one can design a second network $\mathcal{G}'$ indistinguishable from $\mathcal{G}$ up to time t, with an edge appearing for the first time after time t. Depending on the needs of the proof, this edge may (1) reach new nodes, (2) create shortcuts, or (3) make some journeys faster.

In particular, this argument makes it clear that the nodes cannot decide when the broadcast is complete, unless additional knowledge is available. We consider various combinations of knowledge among the following: number of nodes n, a bound Δ on the reappearance time (in $\mathcal{E}^{\mathcal{B}}$), and the period p (in $\mathcal{E}^{\mathcal{P}}$), the resulting settings are referred to as ${\mathcal{E}^{\mathcal{R}}}_n$, ${\mathcal{E}^{\mathcal{B}}}_\Delta$, etc.

The model. A message passing model in continuous time is considered, where the latency ζ is fixed and known to the nodes. When a link appears, it lasts sufficiently long for transmitting at least one message. If a message is sent less than ζ time before the edge disappears, it is lost. Nodes are notified immediately when an incident link appears (`onEdgeAppearance()`) or disappears (`onEdgeDisappearance()`), which we call a *presence oracle* in the present document. Together with ζ and the fact that links are bidirectional, the immediacy of the oracle implies that a node transmitting a message can detect if it was indeed successfully received (if the corresponding edge is still present ζ time after the emission). Based on this observation, we introduced in [62] a special primitive `sendRetry()` that re-sends a message upon the next appearance of an edge if the transmission fails (or if the edge is absent when called), which simplifies the expression of algorithms w.l.o.g. Finally, a node can identify an edge over multiple appearances, and we do not worry about interferences.

In a subsequent work with Gómez-Calzado, Lafuente, and Larrea [123] (reviewed in Section 2.3), we explored various ways of relaxing these assumptions, in particular the presence oracle. Raynal *et al.* [172] also explored different variants of this model.

2.2.2 Main results

We review here only the most significant results from [62, 66], referring the reader to these articles for missing details. The first problem, TDB[$foremost$], can be solved already in ${\mathcal{E}^{\mathcal{R}}}_n$ by a basic flooding technique: every time a new edge appears locally to an informed node, information is sent onto it. Knowledge of n is not required for the broadcast itself, but for termination detection due to a late edge argument. Using the parent-child relations resulting from the broadcasting phase, termination detection proceeds by sending acknowledgments up the tree back to the emitter every time a new node is informed, which is feasible thanks to the recurrence of edges. The emitter detects termination after $n-1$ acknowledgments have been received.

TDB[$shortest$] and TDB[$fastest$] are not feasible in ${\mathcal{E}^{\mathcal{R}}}_n$ because of a late edge argument (in its version 2 and 3, respectively). Moving to the more restricted class $\mathcal{E}^{\mathcal{B}}$, observe first that being in this class without knowing Δ is indistinguishable from being in $\mathcal{E}^{\mathcal{R}}$. Knowing Δ makes it possible for a node to learn its incident edges *in the footprint*, because these edges *must* appear at least once within any window of duration Δ. The main consequence is that the nodes can perform a *breadth-first search (BFS)* relative to the footprint, which guarantees the shortest nature of journeys. It also makes it possible for a parent to know its definitive list of children, which enables recursive termination using a linear number of messages (against $O(n^2)$ in the termination described above). Knowing both n and Δ further improves the termination process, which now becomes implicit after Δn time units.

Remark 1. *${\mathcal{E}^{\mathcal{B}}}_\Delta$ is strictly stronger than ${\mathcal{E}^{\mathcal{B}}}_n$ in the considered model, because n can be inferred from Δ, the reverse being untrue [66]. In fact, the whole footprint can be learned in ${\mathcal{E}^{\mathcal{B}}}_\Delta$ with potential consequences on many problems.*

TDB[$fastest$] remains unsolvable in $\mathcal{E}^{\mathcal{B}}$. In fact, one may design a network in $\mathcal{E}^{\mathcal{B}}$ where fastest journeys do not exist because the journeys keep improving infinitely many times, despite Δ, exploiting here the continuous nature of time. (We give such a construct in [66].) Being in ${\mathcal{E}^{\mathcal{P}}}_p$ prevents such constructs and makes the problem solvable. In fact, the whole schedule becomes learnable in ${\mathcal{E}^{\mathcal{P}}}_p$ with great consequences. In particular, the source can learn the exact time (modulo p) when it has minimum *temporal eccentricity* (*i.e.,* when it takes the smallest time to reach all nodes), and initiate a foremost broadcast at that particular time (modulo p), which *will* be fastest. Temporal eccentricities can be computed using T-Clocks [64, 65], reviewed in Section 4.2.

Remark 2. *A potential risk in continuous time (identified by E. Godard in a private communication) is that the existence of accumulation points in the presence function might prevent fastest journeys to exist at all even in $\mathcal{E}^{\mathcal{P}}$. Here, we are on the safe side thanks to the fact that every edge appears at least for ζ time units, which is constant is the considered model.*

Missing results are summarized through Tables 2.1 and 2.2. Besides *feasibility*, we characterized the time and message complexity of all algorithms, distinguishing between information messages and other (typically smaller) control messages. We also considered the *reusability* of structures from one broadcast to the next, *e.g.,* the underlying paths in the footprint. Interestingly, while some versions of the problem are harder to solve, they offer a better reusability. Some of these facts are discussed further in Section 3.2.

Metric	Class	Knowledge	Feasibility	Reusability	Result from
Foremost	$\mathcal{E}^{\mathcal{R}}$	$\emptyset$	no	–	\
	$\mathcal{E}^{\mathcal{R}}$	n	yes	no	[62] (long [66])
	$\mathcal{E}^{\mathcal{B}}$	Δ	yes	no	/
	$\mathcal{E}^{\mathcal{P}}$	p	yes	yes	[63] (long [65])
Shortest	$\mathcal{E}^{\mathcal{R}}$	$\emptyset$	no	–	\
	$\mathcal{E}^{\mathcal{R}}$	n	no	–	[62] (long [66])
	$\mathcal{E}^{\mathcal{B}}$	Δ	yes	yes	/
Fastest	$\mathcal{E}^{\mathcal{R}}$	$\emptyset$	no	–	\
	$\mathcal{E}^{\mathcal{R}}$	n	no	–	[62] (long [66])
	$\mathcal{E}^{\mathcal{B}}$	Δ	yes	no	/
	$\mathcal{E}^{\mathcal{P}}$	p	yes	yes	[64] (long [65])

Table 2.1: Feasibility and reusability of TDB in different classes of dynamic networks (with associated knowledge).

Metric	Class	Knowl.	Time	Info. msgs (1^{st} run)	Control msgs (1^{st} run)	Info. msgs (next runs)	Control msgs (next runs)
Foremost	$\mathcal{E}^{\mathcal{R}}$	n	unbounded	$O(m)$	$O(n^2)$	$O(m)$	$O(n)$
	$\mathcal{E}^{\mathcal{B}}$	n	$O(n\Delta)$	$O(m)$	$O(n^2)$	$O(m)$	$O(n)$
		Δ	$O(n\Delta)$	$O(m)$	$O(n)$	$O(m)$	0
		$n\&\Delta$	$O(n\Delta)$	$O(m)$	0	$O(m)$	0
Shortest	$\mathcal{E}^{\mathcal{B}}$	Δ	$O(n\Delta)$	$O(m)$	$2n-2$	$O(n)$	0
either of {		$n\&\Delta$	$O(n\Delta)$	$O(m)$	$n-1$	$O(n)$	0
		$n\&\Delta$	$O(n\Delta)$	$O(m)$	0	$O(m)$	0

Table 2.2: Complexity of TDB in $\mathcal{E}^{\mathcal{R}}$ and $\mathcal{E}^{\mathcal{B}}$ with related knowledge (Table from [66]). Control messages are typically much smaller than information messages, and thus counted separately.

Open question 4. *While $\mathcal{E}^{\mathcal{R}}$ is more general (weaker) than $\mathcal{E}^{\mathcal{B}}$ and $\mathcal{E}^{\mathcal{P}}$, it still represents a strong form of regularity. The recent characterization of Class $\mathcal{TC}^{\mathcal{R}}$ in terms of eventual footprint [40] (see Section 2.4 in the present chapter) makes the inner structure of this class more apparent. It seems plausible to us (but without certainty) that sufficient structure may be found in $\mathcal{TC}^{\mathcal{R}}$ to solve* TDB[$foremost$] *with knowledge n, which represents a significant improvement over $\mathcal{E}^{\mathcal{R}}$.*

2.3 Bounding the temporal diameter

Main article: EUROPAR'15 [123]

Being able to bound communication delays in a network is instrumental in solving a number of distributed tasks. It makes it possible, for instance, to distinguish between a crashed node and a slow node, or to create a form of synchronicity in the network. In highly-dynamic networks, the communication delay between two (remote) nodes may be arbitrary long, and so, even if the communication delay between every two neighbors are bounded. This is due to the disconnected nature of the network, which de-correlates

the global delay (*temporal diameter*) from local delays (*edges latencies*).

In a joint work with Gómez, Lafuente, and Larrea [123], we explored different ways of bounding the temporal diameter of the network and of exploiting such a bound. This work was first motivated by a problem familiar to my co-authors, namely the agreement problem, for which it is known that a subset of sufficient size (typically a majority of the nodes) must be able to communicate timely. For this reason, the work in [123] considers properties that apply among *subsets* of nodes (components). Here, we give a simplified account of this work, focusing on the case that these properties apply to the *whole network*. One reason is to make it easier to relate these contributions to the other works presented in this document, while avoiding many details. The reader interested more specifically in the agreement problem, or to finer (component-based) versions of the properties discussed here is referred to [123]. The agreement problem in highly-dynamic networks has also been studied in a number of recent works, including for example [124, 87].

The model. The model is close to the one of Section 2.2, with some relaxations. Namely, time is continuous and the nodes communicate using message passing. Here, the latency ζ is not a constant, which induces a partial asynchrony, but it remains bounded by some ζ_{MAX}. Different options are considered regarding the awareness that nodes have of their incident links, starting with the use of a *presence oracle* as before (nodes are immediately notified when an incident link appears or disappears). Then, we explore possible replacements for such an oracle, which are described gradually. As before, a node can identify a same edge over multiple appearances, and we do not worry about interferences.

2.3.1 Temporal diameter

Let us recall that the temporal diameter of the network, at time t, is the smallest duration d such that $\mathcal{G}_{[t,t+d]}$ is temporally connected (*i.e.*, $\mathcal{G}_{[t,t+d]} \in \mathcal{TC}$). The objective is to guarantee the existence of a bound Δ such that $\mathcal{G}_{[t,t+\Delta]} \in \mathcal{TC}$ for all t, which we refer to as having a *bounded temporal diameter*. The actual definitions in [123] rely on the concept of Δ-journeys, which are journeys whose duration is bounded by Δ. Based on these journeys, a concept of Δ-component is defined as a set of nodes able to reach each other within every time window of duration Δ.

Networks satisfying a bounded temporal diameter are said to be in class $\mathcal{TC}(\Delta)$ in [123]. For consistency with other class names in this document, we rename this class as $\mathcal{TC}^{\mathcal{B}}$, mentioning the Δ parameter only if need be. (A more general distinction between parametrized and non-parametrized classes, inspired from [123], is discussed in Chapter 3.) At an abstract level, being in $\mathcal{TC}^{\mathcal{B}}$ with a known Δ makes it possible for the nodes to make decisions that depend on (potentially) the whole network following a wait of Δ time units. However, if no additional assumptions are made, then one must ensure that no single opportunity of journey is missed by the nodes. Indeed, membership to $\mathcal{TC}^{\mathcal{B}}$ may rely on specific journeys whose availability are transient. In discrete time, a feasible (but costly) way to circumvent this problem is to send a message in each time step, but this makes no sense in *continuous* time.

This impossibility motivates us to write a first version of our algorithms in [123] using the presence oracles from [62, 66]; *i.e.,* primitives of the type `onEdgeAppearance()` and `onEdgeDisappearance()`. However, we observe that these oracles have no realistic implementations and thus we explore various ways of avoiding them, possibly at the cost of stronger assumptions on the network dynamics (more restricted classes of graphs).

2.3.2 Link stability

Instead of *detecting* edges, we study how $\mathcal{TC}^{\mathcal{B}}$ could be specialized for enabling a similar trick to the one in discrete time, namely that if the nodes send messages at *regular* interval, at least *some* of the possible Δ-journeys will be effectively used. The precise condition quite specific and designed to this sole objective. Precisely, we require the existence of particular kinds of Δ-journeys (called β-journeys in [123]) in which every next hop can be performed with some flexibility as to the exact transmission time, exploiting a stability parameter β on the duration of edge presences. The name β was inspired from a similar stability parameter used by Fernández-Anta et al. [106] for a different purpose.

The graphs satisfying this requirement (for some β and Δ) form a strict subset of $\mathcal{TC}^{\mathcal{B}}$ for the same Δ. The resulting class was denoted by $\mathcal{TC}'(\beta)$ in [123], and called *oracle-free*. We now believe this class is quite specific, and may preferably be formulated in terms of communication model within the more general class $\mathcal{TC}^{\mathcal{B}}$. This matter raises an interesting question as to whether and when a set of assumptions should be stated as a class of *graphs* and when it should not. No such ambiguity arises in static networks, where computational aspects and synchronism are not captured by the graph model itself, whereas it becomes partially so with graph theoretical models like TVGs, through the latency function. These aspects are discussed again in Section 3.1.3.

2.3.3 Steady progress

While making the presence oracle unnecessary, the stability assumption still requires the nodes to send the message regularly over potentially long periods of time. Fernández-Anta *et al.* consider another parameter called α in [106], in a discrete time setting. The parameter is formulated in terms of partitions within the network, as the largest number of consecutive steps, for every partition $(S, \bar{S})$ of V, without an edge between S and $\bar{S}$. This idea is quite general and extends naturally to continuous time. Reformulated in terms of journeys, parameter α is a bound on the time it takes for *every next hop* of *some* journeys to appear, "some" being here at least one between every two nodes (and in our case, within every Δ-window).

One of the consequences of this parameter is that a node can *stop* retransmitting a message α time units after it received it, adding to the global communication bound a second *local* one of practical interest. In [123], we considered α only in conjunction with β, resulting in a new class based on (α, β)-journeys and called $\mathcal{TC}''(\alpha, \beta) \subset \mathcal{TC}'(\beta) \subset \mathcal{TC}^{\mathcal{B}}$ (of which the networks in [106] can essentially be seen as discrete versions). With hindsight, the α parameter deserves to be considered independently. In particular, a concept of α-journey where the time of every next hop is bounded by some duration is

of great independent interest, and it is perhaps of a more *structural* nature than β. As a result, we do consider an α-$\mathcal{TC}^{\mathcal{B}}$ class in Chapter 3 while omitting classes based on β.

Impact on message complexity. Intuitively, the α parameter makes it possible to reduce drastically the number of messages. However, its precise effects in an adversarial context remain to be understood. In particular, a node has no mean to decide which of several journeys *prefixes* will eventually lead to an α-journey. As a result, even though a node can stop re-transmitting a message after α time units, it will have to retransmit the same message again if it receives it again in the future (*e.g.,* possibly through a different route).

Open question 5. *Understand the real effect of α-journeys in case of an adversarial* (i.e., *worst-case) edge scheduling ρ. In particular, does it significantly reduces the number of messages?*

In conclusion, the properties presented above helped us propose in [123] different versions of a same algorithm (here, a primitive called *terminating reliable broadcast* in relation to the agreement problem). Each version lied at different level of abstraction and with a gradual set of assumptions, offering a tradeoff between realism and assumptions on the network dynamics.

2.4 Minimal structure and robustness

Main article: arXiv'17 [56] (submitted)

As we have seen along this chapter, highly dynamic networks may possess various types of internal structure that an algorithm can exploit. Because there is a natural interplay between generality and structure, an important question is whether general types of dynamic networks possess sufficient structure to solve interesting problems. Arguably, one of the weakest assumptions in dynamic networks is that every pair of nodes is able to communicate recurrently (infinitely often) through journeys. This property was identified more than three decades ago by Awerbuch and Even [15]. The corresponding class of dynamic networks (Class 5 in [68] – hereby referred to as $\mathcal{TC}^{\mathcal{R}}$) is indeed the most general among all classes of infinite-lifetime networks discussed in this document. This means that, if a structure is present in $\mathcal{TC}^{\mathcal{R}}$, then it can be found in virtually any network, including always-connected networks ($\mathcal{C}^*$), networks whose edges are recurrent or periodic ($\mathcal{E}^{\mathcal{R}}$, $\mathcal{E}^{\mathcal{B}}$, $\mathcal{E}^{\mathcal{P}}$), and networks in which all pairs of nodes are infinitely often neighbors ($\mathcal{K}^{\mathcal{R}}$). (All classes are reviewed in Chapter 3.) Therefore, we believe that the question is important. We review here a joint work with Dubois, Petit, and Robson [56], in which we exploit the structure of $\mathcal{TC}^{\mathcal{R}}$ to built stable intermittent structures.

2.4.1 Preamble

In Section 1.4.3, we presented three ways of interpreting standard combinatorial problems in highly-dynamic networks [73], namely a *temporal*, an *evolving*, and a *permanent* version. The *temporal* interpretation requires that the considered property (*e.g.,* in case

of dominating sets, being either in the set, or adjacent to a node in it) is realized *at least once* over the execution.

Motivated by a distributed setting, Dubois *et al.* [100] define an extension of the temporal version, in which the property must hold not only once, but *infinitely often* – in the case of dominating sets, this means being either in the set or *recurrently* neighbor to a node in the set. Aiming for generality, they focus on $\mathcal{TC}^{\mathcal{R}}$ and exploit the fact that this class also corresponds to networks whose *eventual footprint* is connected [40]. In other words, if a network is in $\mathcal{TC}^{\mathcal{R}}$, then its footprint contains a connected spanning subset of edges which *are* recurrent, and vice versa.

This observation is perhaps simple, but has profound implications. While some of the edges incident to a node may disappear forever, some others *must* reappear infinitely often. Since a distributed algorithm has no mean to distinguish between both types of edges, Dubois *et al.* [100] call a solution *strong* if it remains valid relative to the actual set of recurrent edges, whatever they be.

2.4.2 Robustness and the case of the MIS

In a joint work with Dubois, Petit, and Robson [56], we revisited these notions, employing the terminology of "robustness" (suggested by Y. Métivier), and defining a new form of heredity in *standard* graphs, motivated by these considerations about *dynamic* networks.

Definition 3 (Robustness)**.** *A solution (or property) is said to be* robust *in a graph G iff it is valid in* all *connected spanning subgraphs of G (including G itself).*

This notion indeed captures the uncertainty of not knowing which of the edges are recurrent and which are not in the footprint of a network in $\mathcal{TC}^{\mathcal{R}}$. Then we realized that it also has a very natural motivation in terms of static networks, namely that some edges of a given network may crash definitely and the network is used so long as it is connected. This duality makes the notion quite general and its study compelling. It also makes it simpler to think about the property.

In [56], we focus on *maximal independent sets* (MIS), which is a maximal set of non-neighbor nodes. Interestingly, robust MISs may or may not exist depending on the considered graph. For example, if the graph is a triangle (Figure 2.3(a)), then only

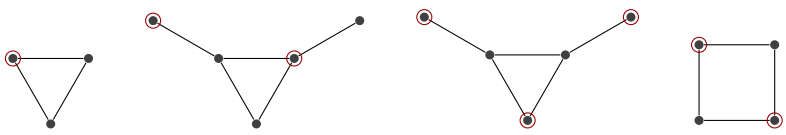


Figure 2.3: Four examples of MISs in various graphs (or footprints).

one MIS exists up to isomorphism, consisting of a single node. However, this set is no longer maximal in one of the possible connected spanning subgraphs: the triangle graph admits no robust MIS. Some graphs do admit a robust MIS, but not all of the MISs are robust. Figures 2.3(b) and 2.3(c) show two MISs in the bull graph, only one of which

is robust. Finally, some graphs are such that *all* MISs are robust (*e.g.,* the square on Figure 2.3(d)).

We characterize exactly the set of graphs in which all MISs are robust [56], denoted $\mathcal{RMIS}^{\forall}$, and prove that it consists *exactly* of the union of complete bipartite graphs and a new class of graphs called *sputniks*, which contains among others things all the trees (for which any property is trivially robust). Graphs not in $\mathcal{RMIS}^{\forall}$ may still *admit* a robust MIS, *i.e.,* be in $\mathcal{RMIS}^{\exists}$, such as the bull graph on Figure 2.3. However, the characterization of $\mathcal{RMIS}^{\exists}$ proved quite complex, and instead of a closed characterization, we presented in [56] an algorithm that finds a robust MIS if one exists, and rejects otherwise. Interestingly, our algorithm has low polynomial complexity despite the fact that exponentially many MISs *and* exponentially many connected spanning subgraphs may exist.

We also turn to the distributed version of the problem, and prove that finding a robust MIS in $\mathcal{RMIS}^{\forall}$ is a *local* problem, namely a node can decide whether or not it belongs to the MIS by considering only information available within $\frac{\log n}{\log \log n} = o(\log n)$ hops in the graph (resp. in the footprint in case of dynamic networks). On the other hand, we show that finding a robust MIS in $\mathcal{RMIS}^{\exists}$ (or deciding if one exists in general graphs) is *not* local, as it may require information up to $\Omega(n)$ hops away, which implies a separation between the MIS problem and the robust MIS problem in general graphs, since the former is solvable within $2^{\mathcal{O}(\sqrt{\log n})}$ rounds in the LOCAL model [166].

Some remarks

Whether a closer characterization of $\mathcal{RMIS}^{\exists}$ exists in terms of natural graph properties remains open. (It might be that it does not.) It would be interesting, at least, to understand how large this *existential* class is compared to its *universal* counterpart $\mathcal{RMIS}^{\forall}$. Another general question is whether large *universal* classes exist for other combinatorial problems than MIS, or if the notion is somewhat too restrictive for the universal versions of these classes. Of particular interests are other symmetry breaking problems like MDS or k-coloring, which play an important role in communication networks.

2.5 Conclusion and perspectives

Identifying necessary or sufficient conditions for distributed problems has been a recurrent theme in our research. It has acted as a driving force and sparked off many of our investigations. Interestingly, the structure revealed through them often turn out to be quite general and of a broader applicability. The next chapter reviews all the classes of graphs found through these investigations, together with classes inferred from other assumptions found in the literature.

We take the opportunity of this conclusion to discuss a matter which we believe is important and perhaps insufficiently considered by the distributed computing community: that of structures available a finite number of times. Some distributed problems may require a certain number k of occurrences of a structural property, like an edge

appearance of a journey. If an algorithm requires k_1 such occurrences and another algorithm requires k_2 such occurrences, an instinctive reaction is to discard the significance of their difference, especially if k_1 and k_2 are of the same order, on the basis that constant factors among various complexity measures is not of utmost importance. The missed point here is that, in a dynamic network, such difference may not only relate to complexity, but also to feasibility! A dynamic network $\mathcal{G}$ may typically enable only a finite number of occurrences of some desired structure, *e.g.,* three round-trip journeys between all the nodes. If an algorithm requires only three such journeys and another requires four, then the difference between both is highly significant.

For this reason, we call for the definition of structural metrics in dynamic networks which may be used to characterize *fine-grained* requirements of an algorithm. Early efforts in this direction, motivated mainly by complexity aspects (but with similar effects), include Bramas and Tixeuil [39], Bramas *et al.* [38], and Dubois *et al.* [100].

Research avenue 6. *Systematize the definition of complexity measures based on temporal features which may be available on a non-recurrent basis. Start comparing algorithms based on the number of occurrences they require of these structures.*

Chapter 3

Around classes of dynamic networks

In the same way as standard graph theory identifies a large number of special classes of graphs (trees, planar graphs, grids, complete graphs, *etc.*), we review here a collection of *classes of dynamic graphs* introduced in various works. Many of these classes were compiled in the context of a joint work with Flocchini, Quattrociocchi, and Santoro in 2012 [68], some others defined in a joint work with Chaumette and Ferreira in 2009 [52], with Gómez-Calzado, Lafuente, and Larrea 2015 [123], and with Flocchini, Mans, and Santoro [62, 66]; finally, some are original generalizations of existing classes. We resist the temptation of defining a myriad of classes by limiting ourselves to properties *used effectively in the literature* (often in the form of necessary or sufficient conditions for distributed algorithms, see Chapter 2).

Here, we get some distance from distributed computing and consider the intrinsic features of the classes and their inter-relations from a set-theoretic point of views. Some discussions are adapted from the above papers, some are new; the existing ones are revisited with (perhaps) more hindsight. In a second part, we review our efforts related to testing automatically properties related to these classes, given a *trace* of a dynamic network. Finally, we discuss the connection between classes of graphs and real-world mobility, with an opening on the emerging topic of movement synthesis.

3.1 List of classes

Main articles: SIROCCO'09 [52], IJPEDS'12 [68], IJFCS'15 [66], EUROPAR'15 [123].

The classes listed below are described mostly in the language of time-varying graphs (see definitions in Section 1.2.1). However, the corresponding properties are conceptually general and may be expressed in other models as well. For generality, we formulate them in a continuous time setting (nonetheless giving credit to the works introducing their discrete analogues). Common restrictions apply in the whole section. In particular, we consider networks with a *fixed* number of nodes n. We consider only edge appearances which lasts sufficiently long for the edge to be used (typically in relation to the latency); for example, when we say that an edge "appears at least once", we mean implicitly for a sufficient duration to be used. The exact meaning of *being used* is vague to accomodate various notions of interaction (*e.g.,* atomic operations or message

exchanges). Finally, we limit ourselves to undirected edges, which impacts inclusion relations among classes.

Classes names. Opportunity is taken to give new names to the classes. Some classes were assigned a $\mathcal{C}_x$ name in some of our previous works and some others a $\mathcal{F}_x$ name (sometimes for the same class). We hope the new names convey mnemonic information which will prove more convenient. In particular, the *base* letter indicates the subject of the property applies: journeys ($\mathcal{J}$), edges ($\mathcal{E}$), paths ($\mathcal{P}$), connectivity ($\mathcal{C}$), temporal connectivity ($\mathcal{TC}$). The superscript provides information about the property itself: recurrent ($\mathcal{R}$), bounded recurrent ($\mathcal{B}$), periodic ($\mathcal{P}$), round-trip ($\circlearrowleft$), one to all ($1\forall$), all to one ($\forall 1$), and so on. Note that the letter $\mathcal{P}$ is used twice in this convention, but its position (base or superscript) makes it non-ambiguous.

3.1.1 Classes based on *finite* properties

We say that a graph property *Prop* is *finite* if $Prop(\mathcal{G}_{[0,t]}) \implies Prop(\mathcal{G})$ for some t. In other words, if by some time the property has been satisfied, then the subsequent evolution of the graph with respect to this property is irrelevant. Among other consequences, this makes the properties satisfiable by graphs whose lifetime is *finite* (with advantages for offline analysis). Below are a number of classes based on finite properties. The contexts of their introduction in briefly reminded in Section 3.1.5.

Class $\mathcal{J}^{1\forall}$ (Temporal source). $\exists u \in V, \forall v \in V, u \rightsquigarrow v$. *At least one node can reach all the others through a journey.*

Class $\mathcal{J}^{\forall 1}$ (Temporal sink). $\exists v \in V, \forall u \in V, u \rightsquigarrow v$. *At least one node can be reached by all others through a journey.*

Class $\mathcal{TC}$ (Temporal connectivity). $\forall u, v \in V, u \rightsquigarrow v$. *Every node can reach all the others through a journey.*

Class $\mathcal{TC}^{\circlearrowleft}$ (Round-trip temporal connectivity). $\forall u, v \in V, \exists \mathcal{J}_1 \in \mathcal{J}^*_{(u,v)}, \exists \mathcal{J}_2 \in \mathcal{J}^*_{(v,u)}, arrival(\mathcal{J}_1) \leq departure(\mathcal{J}_2)$. *Every node can reach every other node and be reached from that node afterwards.*

Class $\mathcal{E}^{1\forall}$ (Temporal star). $\exists u \in V, \forall v \in V, \exists t \in \mathcal{T}, (u, v) \in G_t$. *At least one node will share an edge at least once with every other node (possibly at different times).*

Class $\mathcal{K}$ (Temporal clique). $\forall u, v \in V, \exists t \in \mathcal{T}, (u, v) \in G_t$. *Every pair of nodes will share an edge at least once (possibly at different times).*

Subclasses of $\mathcal{K}$ (Counsils). *Three subclasses of $\mathcal{K}$ are defined by Laplace [149], called $\mathcal{F}_{OC}$ (open counsil), $\mathcal{F}_{PCC}$ (punctually-closed counsil), and $\mathcal{F}_{CC}$ (closed counsil) corresponding to gradual structural constraints in $\mathcal{K}$. For instance, $\mathcal{F}_{OC}$ corresponds to network in which the nodes gather incrementally as a clique that eventually forms a complete graph, each node joining the clique at a* distinct *time.*

Discussion 4 (On the relevance of strict journeys)**.** Some of the classes presented here were sometimes separated into a *strict* and a *non-strict* version, depending on which

kind of journey is guaranteed to exist. However, the very notion of a *strict* journey is more relevant in discrete time, where a clear notion of time step exists. In constrast, an explicit latency is commonly considered in continous time and unifies both versions elegantly (non-strict journeys simply correspond to assuming a latency of 0).

In some of our works (*e.g.,* [52]), we considered strict versions of some classes. We maintain the distinction between both versions in some of the content below, due to (1) the reporting nature of this document, and (2) the fact that many works target discrete time. In particular, we suggest to denote strict versions of the classes using a superscript ">" applied to the class name, such as in $\mathcal{J}^{1\forall>}$, $\mathcal{TC}^{>}$, and $\mathcal{TC}^{\circlearrowleft>}$. Later in the section, we will ignore the disctinction between both, in particular in the updated hierarchy given in Figure 3.2.

3.1.2 Classes based on *recurrent* properties

When the lifetime is *infinite*, temporal connectivity is often taken for granted and more elaborate properties are considered. We review below a number of such classes, most of which were compiled in [68]. For readability, most domains of the variables are not repeated in each definition. By convention, variable t always denotes a time in the lifetime of the network (*i.e.,* $t \in \mathcal{T}$), variables Δ and p denote durations (in $\mathbb{T}$), variable e denotes an edge (in E) and variables u and v denote vertices (in V), all relative to a network $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$.

Class $\mathcal{TC}^{\mathcal{R}}$ (Recurrent temporal connectivity). *For all t, $\mathcal{G}_{[t,+\infty)} \in \mathcal{TC}$. At any point in time, the network will again be temporally connected in the future.*

Class $\mathcal{TC}^{\mathcal{B}}$ (Bounded temporal diameter). $\exists\Delta, \forall t$, $\mathcal{G}_{[t,t+\Delta)} \in \mathcal{TC}$. *At any point in time, the network is temporally connected within the next Δ units of time.*

Class $\mathcal{E}^{\mathcal{R}}$ (Recurrent edges). *The footprint (V, E) is connected and* $\forall e \in E, \forall t, \exists t' > t, \rho(e, t') = 1$. *If an edge appears once, it appears infinitely often (or remains present).*

Class $\mathcal{E}^{\mathcal{B}}$ (Bounded edge recurrence). *The footprint (V, E) is connected and there is a Δ such that* $\forall e \in E, \forall t, \exists t' \in [t, t + \Delta), \rho(e, t') = 1$. *If an edge appears once, then it always re-appears within bounded time (or remains present).*

Class $\mathcal{E}^{\mathcal{P}}$ (Periodic edges). *The footprint (V, E) is connected and* $\forall e \in E, \forall t, \rho(e, t) = \rho(e, t + kp)$, *for some p and all integer k. The schedule of every edge repeats modulo some period. (If every edge has its own period, then p is the least common multiple.)*

Class $\mathcal{P}^{\mathcal{R}}$ (Recurrent paths). $\forall u, v, \forall t, \exists t' > t$, *a path from u to v exists in $G_{t'}$. A classical path will exist infinitely many times between every two vertices.*

Class $\mathcal{C}^{\mathcal{R}}$ (Recurrently-connected snapshots). $\forall t, \exists t' > t, G_{t'}$ *is connected. At any point in time, there will be a connected snapshot in the future.*

Class $\mathcal{C}^{*}$ (Always-connected snapshots). $\forall t, G_t$ *is connected. At any point in time, the snapshot is connected.*

Class $\mathcal{C}^{\cap}$ (T-interval connectivity). *For a given $T \in \mathbb{T}$,* $\forall t, \cap\mathcal{G}_{[t,t+T)}$ *is connected. For all period of length T, there is a spanning connected subgraph which is stable.*

Class $\mathcal{K}^{\mathcal{R}}$ (Complete graph of interaction). *The footprint $G = (V, E)$ is complete, and $\forall e, \forall t, \exists t' > t : \rho(e, t')=1$. Every pair of vertices share an edge infinitely often. (In other words, the eventual footprint is complete.)*

Class α-$\mathcal{TC}^{\mathcal{B}}$ (Steady progress). *There is a duration α such that for all starting time t and vertices u and v, at least one (elementary) journey from u to v is such that every next hop occurs within α time units.*

The next two classes are given for completeness. We argued in Section 3.1.3 that they may preferably not be stated as independent classes, due to their high-degree of specialization, and rather stated as extra stability constraints within $\mathcal{TC}^{\mathcal{B}}$ or α-$\mathcal{TC}^{\mathcal{B}}$.

Class β-$\mathcal{TC}^{\mathcal{B}}$ (Bounded temporal diameter with stable links). *There is a maximal latency ζ_{MAX}, durations $\beta \geq 2\zeta_{MAX}$ and Δ such that for all starting time t and vertices u and v, at least one journey from u to v within $[t, t + \Delta)$ is such that its edges are crossed in disjoint periods of length β, each edge remaining present in the corresponding period.*

Class (α, β)-$\mathcal{TC}^{\mathcal{B}}$. α-$\mathcal{TC}^{\mathcal{B}} \cap \beta$-$\mathcal{TC}^{\mathcal{B}}$.

3.1.3 Dimensionality of the assumptions

The complexity of the above description of β-$\mathcal{TC}^{\mathcal{B}}$ raises a question as to the *prerogatives* of a class. In static graphs, the separation between structural properties (*i.e.,* the network topology) and computational and communicational aspects (*e.g.,* synchronicity) are well delineated. The situation is more complex in dynamic networks, and especially in continuous time, due to the injection of time itself into the graph model. For example, in time-varying graphs, the latency function encodes assumptions about synchronicity directly within the graph, making the distinction between structure and communication more ambiguous.

The down side of this expressivity is a temptation to turn any combination of assumptions into a dedicated class. We do not think this is a reasonable approach, neither do we have a definite opinion on this matter. Some types of assumptions remain non-ambiguously orthogonal to graph models (*e.g.,* size of messages, knowledge available to the nodes, unique identifiers), but it is no longer true that dynamic graph models capture only structural information. A discussion on related topics can also be found in Santoro [176].

Research avenue 7. *Clarify the prerogatives of (dynamic) graph models with respect to the multi-dimensionality of assumptions in distributed computing.*

3.1.4 Parametrized classes

With Gómez-Calzado, Lafuente, and Larrea, we distinguish in [123] between two versions of some classes which admit parameters. Let us extend the discussion to parametrized classes in general, including for example $\mathcal{TC}^{\mathcal{B}}$, $\mathcal{E}^{\mathcal{B}}$, and $\mathcal{C}^{\cap}$, which are all parametrized by a *duration*. Given such a class, one should distinguish between the *instantiated* version

(*i.e.,* for a given value of the parameter) and the *universal* version (union of instantiated versions over all finite parameter values). Taking class $\mathcal{E}^{\mathcal{B}}$ as an example, three notations can be defined: $\mathcal{E}^{\mathcal{B}}$ is the instantiated version with an *implicit* parameter (as used above); $\mathcal{E}^{\mathcal{B}}(x)$ is the instantiated version with an *explicit* parameter; and $\cup\mathcal{E}^{\mathcal{B}}$ is the universal version. Some inclusions relations among classes, discussed next, are sensitive to this aspect, for example the fact that $\mathcal{C}^{\cap} \subseteq \mathcal{C}^{*} \subseteq \cup\mathcal{C}^{\cap}$ brought some confusion in previous versions of the hierarchy (where this distinction was absent).

3.1.5 Background in distributed computing

We review here some of the contexts in which the above classes were introduced. We mainly focus on recurrent properties, the case of finite properties being already covered in Section 2.1.

$\mathcal{TC}^{\mathcal{R}}$ is perhaps the most general class among the ones having infinite lifetime. This property corresponds to class $\mathcal{C}_5$ in our 2012 article [68]. It seems to have been first considered by Awerbuch and Even in the 80s [15]. Being in this class is often implicitly assumed in mobile ad hoc networks, as it captures the ability for the nodes to influence each other recurrently. We describe in Section 2.4 a joint work with Dubois, Petit, and Robson [56], in which the structure of $\mathcal{TC}^{\mathcal{R}}$ is explored.

$\mathcal{TC}^{\mathcal{B}}$ corresponds to the subset of $\mathcal{TC}^{\mathcal{R}}$ in which the communication time is bounded by some value Δ. This class corresponds in essence to classical assumptions made in static distributed computing, when the communication is asynchronous, but interpreted here as a *bounded temporal diameter* of a highly-dynamic network. Together with Gomez-Calzado, Lafuente, and Larrea, we explored in [123] (reviewed in Section 2.3) gradual restrictions of this class that make it possible to exploit the communication bound in realistic and efficient ways. One of the restriction corresponds to Class α-$\mathcal{TC}^{\mathcal{B}}$, where α stands for a parameter introduced by Fernandez-Anta *et al.* in [106], which we re-interpret in terms of the existence of journeys whose wait at *every* intermediate nodes is bounded (steady progress). Interestingly, this property also manifests with high probability in a wide range of edge-markovian dynamic graphs [29], making it possible to stop re-transmission after some time (*parsimonious* broadcast).

Classes $\mathcal{E}^{\mathcal{R}}$, $\mathcal{E}^{\mathcal{B}}$, and $\mathcal{E}^{\mathcal{P}}$ were introduced in a joint work with Flocchini, Mans, and Santoro [62] (journal version [66]). These classes were shown to have a tight relation with the problems of foremost, shortest, and fastest broadcasts (with termination detection). The reader is referred to Section 2.2 for details. These three classes were later considered by Aaron *et al.* in [1], who show that the *dynamic map visitation* problem admit different complexities depending on the class, namely severe inapproximability in $\mathcal{E}^{\mathcal{R}}$, limited approximability in $\mathcal{E}^{\mathcal{B}}$, and tractability in $\mathcal{E}^{\mathcal{P}}$. A number of works on network exploration [112, 130, 110] and [139, 153] also considered periodical networks.

Classes $\mathcal{P}^{\mathcal{R}}$ and $\mathcal{C}^{\mathcal{R}}$ were introduced by Ramanathan *et. al* in [170]; these classes capture, among other things, the ability to wait, for any two given nodes, that a snapshot of the network occurs where these nodes are connected by a standard *path* (Class $\mathcal{P}^{\mathcal{R}}$); or to wait for a snapshot where all the nodes are connected (Class $\mathcal{C}^{\mathcal{R}}$). The original paper does not mention that these features must hold infinitely often (only "one or more" times).

$\mathcal{K}^{\mathcal{R}}$ are graphs from $\mathcal{E}^{\mathcal{R}}$ whose *footprint* is a complete graph. In other words, every two nodes in the network are infinitely often neighbors. The network resulting from the random scheduler from [12] *almost surely* belongs to this class and inspired its definition. However, we take the opportunity to correct here a statement from [68], in which we misattributed the results from [12] to $\mathcal{K}^{\mathcal{R}}$. The results from [12] apply when a fairness criterion is satisfied, namely that every global state (configuration) that is infinitely often reachable is infinitely often reached. The random scheduler in [12] is only a *possible* way of satisfying this constraint, as it is one of the possible ways of generating graphs in $\mathcal{K}^{\mathcal{R}}$. As for the relation between $\mathcal{K}^{\mathcal{R}}$ and fair executions of population protocols, both are in fact *incomparable*.

The characteristic property of $\mathcal{C}^*$ (every snapshot is connected) was considered in [165] in the context of information dissemination. For exemple, this property implies that at any point in time, at least one non-informed node has an informed neighbor (unless dissemination is complete), which bounds the propagation time by $n-1$ rounds in synchronous systems.

Class $\mathcal{C}^{\cap}$ corresponds to a variant of Class $\mathcal{C}^*$ with an additional stability parameter. It was introduced by Kuhn *et al.* in [146] to study problems such as counting, token dissemination, and computation of functions whose input is spread over all the nodes (with adversarial edge scheduling). The running time of some algorithms for these problems in class $\mathcal{C}^{\cap}(T)$ is sped up by a factor of T. In other words, the algorithms profit from stability. Godard and Mazauric [122] consider the particular case that a connected spanning subgraph is *always* present (*i.e.,* $\mathcal{C}^{\cap}(\infty)$), which they refer to as a *statically-connected* dynamic network (see also Section 4.4.1.)

Message adversaries. A significant line of work in distributed computing has considered dynamic transmission faults occurring on top of an (otherwise) static network. These systems are typically synchronous, and an execution is modeled by sequence of graphs, each element of which represents successful transmissions in the corresponding round. The main difficuly is to deal with uncertainty as to what faults will occur in what round, and a typical way of restricting uncertainty is to restrict the *set* of possible graphs, while having little or no control over the order in which these graphs will occur. The reader is referred to Santoro and Widmayer [178] for one of the seminal works in this area and to [171] for a more recent survey. In the past decade, there has been interesting convergences between these models and the kind of highly-dynamic networks discussed here (*e.g.,* [89, 87]). In particular, while the common approach was mainly to restrict the *set* of possible graphs, some works (like the *heard-of* [89] model) have considered constraints in the form of predicates that apply to the sequence *as a whole*, which is what some of the above classes of dynamic networks essentially are.

Research avenue 8. *Establish a methodical comparison between predicates "à la heard-of" and classes of dynamic networks discussed above. The usual restrictions on the set of allowed graphs (*e.g., *standard connectivity) have been relaxed in recent works (*e.g., *[88]), taking the assumptions down to pure temporal reachability achieved on top of possibly disconnected graphs (close in spirit to the above classes). Explicit efforts to unify both areas have also been made by Coulouma et al. [96] and Godard [121].*

3.1.6 Synthetic view of the classes

The classes are summarized in Table 3.1. Besides being dichotomized into *finite* or *recurrent*, some are based on the concept of journeys, others on standard paths (within snapshots), and others on the individual behavior of edges. Some are *uniform*, meaning that every node plays the same role in the definition; some are not. The distinction between strict and non-strict journeys is preserved in this table (it will be dropped subsequently based on Discussion 4 on page 40).

Proposed name	In [68]	In [52]	Other names	Journey-based	Path-based	Edge-based	Finite	Uniform	Visual
$\mathcal{J}^{1\forall}$	$\mathcal{C}_1$	$\mathcal{F}_1$	-	✓	–	–	✓	–	$1 \rightsquigarrow *$
$\mathcal{J}^{1\forall>}$	-	$\mathcal{F}_3$	-	✓	–	–	✓	–	$1 \overset{st}{\rightsquigarrow} *$
$\mathcal{J}^{\forall 1}$	$\mathcal{C}_2$	$\mathcal{F}_7$	-	✓	–	–	✓	–	$* \rightsquigarrow 1$
$\mathcal{J}^{\forall 1>}$	-	-	-	✓	–	–	✓	–	$* \overset{st}{\rightsquigarrow} 1$
$\mathcal{TC}$	$\mathcal{C}_3$	$\mathcal{F}_2$	$\mathcal{TC}$ [5]	✓	–	–	✓	✓	$* \rightsquigarrow *$
$\mathcal{TC}^{>}$	-	$\mathcal{F}_4$	$\mathcal{TC}$ [5]	✓	–	–	✓	✓	$* \overset{st}{\rightsquigarrow} *$
$\mathcal{TC}^{\circlearrowleft}$	$\mathcal{C}_4$	-	-	✓	–	–	✓	✓	$* \circlearrowleft *$
$\mathcal{E}^{1\forall}$	-	$\mathcal{F}_5$	-	–	–	✓	✓	–	$1 – *$
$\mathcal{K}$	-	$\mathcal{F}_6$	-	–	–	✓	✓	✓	$* – *$
Counsils	-	-	$\mathcal{F}_{OC/[P]CC}$ [149]	–	–	✓	✓	–	–
$\mathcal{TC}^{\mathcal{R}}$	$\mathcal{C}_5$	-	ETDN[170], $\mathcal{COT}$[40]	✓	–	–	–	✓	$* \overset{\mathcal{R}}{\rightsquigarrow} *$
$\mathcal{TC}^{\mathcal{B}}$	-	-	$\mathcal{TC}(\Delta)$ [123]	✓	–	–	–	✓	$* \overset{\mathcal{B}}{\rightsquigarrow} *$
α-$\mathcal{TC}^{\mathcal{B}}$	-	-	-	✓	–	–	–	✓	–
$\mathcal{E}^{\mathcal{R}}$	$\mathcal{C}_6$	-	$\mathcal{F}_9$ [46], $\mathcal{R}$[66]	–	–	✓	–	✓	$\overset{\mathcal{R}}{\bullet – \bullet}$
$\mathcal{E}^{\mathcal{B}}$	$\mathcal{C}_7$	-	$\mathcal{B}$[66]	–	–	✓	–	✓	$\overset{\mathcal{B}}{\bullet – \bullet}$
$\mathcal{E}^{\mathcal{P}}$	$\mathcal{C}_8$	-	$\mathcal{P}$[66]	–	–	✓	–	✓	$\overset{\mathcal{P}}{\bullet – \bullet}$
$\mathcal{K}^{\mathcal{R}}$	$\mathcal{C}_{13}$	-	$\mathcal{F}_8$ [46]	–	–	✓	–	✓	$\overset{\mathcal{R}}{* – *}$
$\mathcal{P}^{\mathcal{R}}$	$\mathcal{C}_{12}$	-	ERDN [170]	–	✓	–	–	✓	$\overset{\mathcal{R}}{\bullet – – \bullet}$
$\mathcal{C}^{\mathcal{R}}$	$\mathcal{C}_{11}$	-	ECDN [170]	–	✓	–	–	✓	$\overset{\mathcal{R}}{* – – *}$
$\mathcal{C}^{*}$	$\mathcal{C}_9$	-	-	–	✓	–	–	✓	$\overset{*}{* – – *}$
$\mathcal{C}^{\cap}$	$\mathcal{C}_{10}$	-	-	–	✓	–	–	✓	T-$\overset{*}{* – – *}$

Table 3.1: Summary of key properties of the classes.

Research avenue 9. *Revisit the expression of the classes using* temporal logic. *Many of the definitions make extensive use of temporal adjectifs and quantifiers (*e.g., *"eventually", "over time", "at least once" etc.) making temporal logic a natural choice.*

Subclasses of dynamic networks induced by footprints Dubois et al. [100] define an notion of *induced subclass* in which one restricts a class of dynamic network to the ones whose footprints fall into a given class of (static) graphs $\mathcal{F}$. For example, $\mathcal{TC}^{\mathcal{R}}{}_{|\mathcal{F}}$ is the subset of $\mathcal{TC}^{\mathcal{R}}$ whose footprint is in $\mathcal{F}$. Fluschnik et al. [114] also suggest a classification of dynamic networks based mostly on properties of the footprint (underlying graph) that impact the computational complexity of computing separators [138, 191].

Typesetting the classes in LaTeX

To all intents and purposes, here are some typeset commands:

```
\newcommand{\ER}{\ensuremath{{\cal E^R}}\xspace}
\newcommand{\EB}{\ensuremath{{\cal E^B}}\xspace}
\newcommand{\EP}{\ensuremath{{\cal E^P}}\xspace}
\newcommand{\JOA}{\ensuremath{{\cal J}^{1\forall}}\xspace}
\newcommand{\JAO}{\ensuremath{{\cal J}^{\forall 1}}\xspace}
\newcommand{\RT}{\ensuremath{{\cal TC^{\circlearrowleft}}}\xspace}
\newcommand{\TC}{\ensuremath{{\cal TC}}\xspace}
\newcommand{\TCR}{{\ensuremath{{\cal TC^R}}}\xspace}
\newcommand{\TCB}{{\ensuremath{{\cal TC^B}}}\xspace}
\newcommand{\AC}{\ensuremath{{\cal C^*}}\xspace}
\newcommand{\TINT}{\ensuremath{{\cal C}^{\cap}}\xspace}
\newcommand{\PR}{\ensuremath{\cal P^R}\xspace}
\newcommand{\CR}{\ensuremath{{\cal C^R}}\xspace}
\newcommand{\KG}{\ensuremath{{\cal K}}\xspace}
\newcommand{\EOA}{\ensuremath{{\cal E}^{1\forall}}\xspace}
\newcommand{\KR}{\ensuremath{{\cal K^R}}\xspace}
```

3.2 Relations between classes

Main articles: SIROCCO'09 [52], IJPEDS'12 [68], IJFCS'15 [66], EUROPAR'15 [123].

Formally, each class is a *set* of graphs, and these sets are related to each other through inclusion relations. We review here a number of such relations among the known classes. Most of the relations concerning finite properties are from [46, 52] and most of the ones concerning recurrent properties are from [68]. Here, we give a unified account including also new classes and relations. Then, we discuss relations of a more computational nature which we characterized with Flocchini, Mans, and Santoro in [66]. Finally, some possible uses of such hierarchy are reviewed.

3.2.1 Inclusion relations among classes

Due to the reporting nature of this document, the hiearchies from [52] and [68] are reproduced unchanged (up to the names) on Figure 3.1. We briefly recall the arguments behind these inclusions and present the new relations.

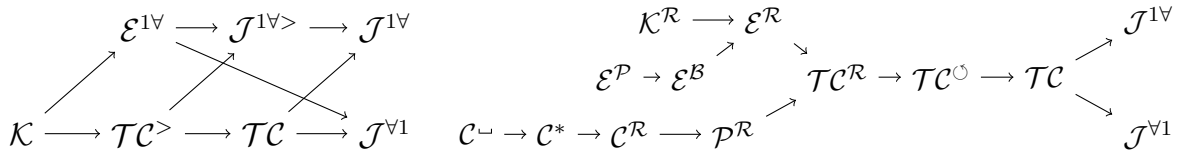


Figure 3.1: Hierarchies from [52] (left) and [68] (right), unified in Figure 3.2.

Finite properties. Unless $n = 0$, it holds that something true for all nodes is also true for at least one node, therefore $\mathcal{TC} \subseteq \mathcal{J}^{1\forall}$, $\mathcal{TC} \subseteq \mathcal{J}^{\forall 1}$, and $\mathcal{K} \subseteq \mathcal{E}^{1\forall}$. Because every edge induces a valid (one-hop) journey, it holds that $\mathcal{E}^{1\forall} \subseteq \mathcal{J}^{1\forall}$ and $\mathcal{K} \subseteq \mathcal{TC}$, and (somewhat reversely) $\mathcal{E}^{1\forall} \subseteq \mathcal{J}^{\forall 1}$. It also holds that $\mathcal{TC}^{\mathcal{R}} \subseteq \mathcal{TC}^{\circlearrowleft} \subseteq \mathcal{TC}$. These inclusions are actually strict, and so are most of the subsequent ones.

Recurrent properties. As expected, $\mathcal{TC}^{\mathcal{R}}$ is the largest class among those based on recurrent properties. In particular, it contains $\mathcal{E}^{\mathcal{R}}$, which itself contains $\mathcal{E}^{\mathcal{B}}$, which in turn contains $\mathcal{E}^{\mathcal{P}}$. Indeed, periodicity of edges is a special case of bounded recurrence of edges, which is a special case of recurrence of edges. The containment of $\mathcal{E}^{\mathcal{R}}$ into $\mathcal{TC}^{\mathcal{R}}$ holds because $\mathcal{E}^{\mathcal{R}}$ additionally requires that the footprint is connected, thus reappearance of edges transitively create recurrent journeys between *all* pairs of nodes. The inclusion is strict, because some graphs in $\mathcal{TC}^{\mathcal{R}}$ may contain non-recurrent edges. In fact, $\mathcal{TC}^{\mathcal{R}}$ is the set of graphs in which a connected spanning subset of edges is recurrent [40] (see also Section 2.4). A special case within $\mathcal{E}^{\mathcal{R}}$ is when the footprint is complete, yielding class $\mathcal{K}^{\mathcal{R}} = \mathcal{K} \cap \mathcal{E}^{\mathcal{R}}$.

The repetition of available paths within snapshots eventually creates journeys between nodes, so $\mathcal{P}^{\mathcal{R}} \subseteq \mathcal{TC}^{\mathcal{R}}$. Since connected snapshots offer paths between all pairs of nodes, it follows that $\mathcal{C}^{\mathcal{R}} \subseteq \mathcal{P}^{\mathcal{R}}$. Next, what is true in each step must also be true infinitely often, thus $\mathcal{C}^{*} \subseteq \mathcal{C}^{\mathcal{R}}$. The case of $\mathcal{C}^{\cap}$ is more subtle because the *universal* version of this class (see Section 3.1.4) equals $\mathcal{C}^{*}$, while *instantiated* versions of it may be strictly smaller than $\mathcal{C}^{*}$. The relations represented on Figures 3.1 and 3.2 are for the instantiated versions of $\mathcal{C}^{\cap}$.

Having a bounded temporal diameter at all times implies that all nodes can recurrently reach each other through journeys, thus $\mathcal{TC}^{\mathcal{B}} \subseteq \mathcal{TC}^{\mathcal{R}}$. The inclusion is actually strict, as one could design a graph in $\mathcal{TC}^{\mathcal{R}}$ whose temporal diameter keeps growing unboundedly with time.

Under mild assumption (including discrete settings and bounded latency continuous settings), it holds that $\mathcal{C}^{*} \subseteq \mathcal{TC}^{\mathcal{B}}$. Finally, the existence of α-journeys imply bounds on the temporal diameter (by $\alpha(n-1)$ time units, plus latencies), implying that α-$\mathcal{TC}^{\mathcal{B}} \subseteq \mathcal{TC}^{\mathcal{B}}$ (and justifying the $\mathcal{B}$ superscript in the name). All these relations are depicted in the updated version of the hierarchy on Figure 3.2.

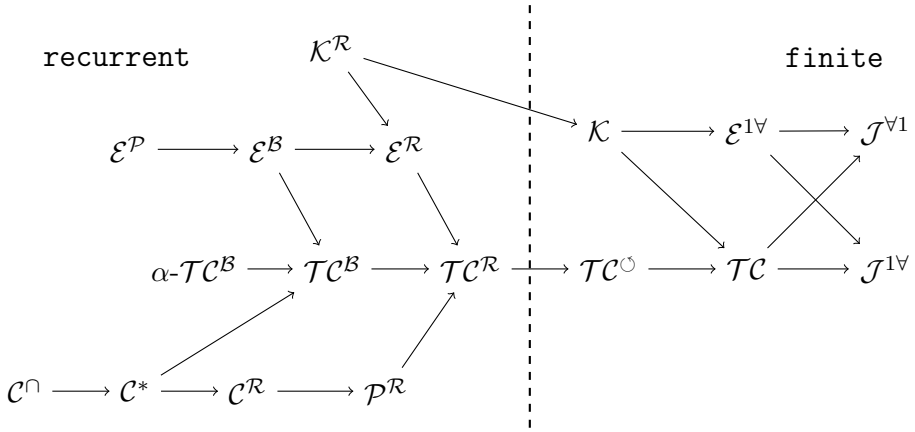


Figure 3.2: Updated hierarchy of the main classes of dynamic networks.

Research avenue 10. *Investigate the relations between directed analogues of these classes,* i.e., *when directed (non-symmetrical) edges can exist. The impact is significant. For example, with undirected edges, the repetition of journeys from a node u to a node v eventually creates backward journeys from v to u, which explains why no recurrent version of $\mathcal{J}^{1\forall}$ was defined ($\mathcal{J}^{1\forall\mathcal{R}}$ would amount to $\mathcal{TC}^{\mathcal{R}}$ itself). This type of "reversibility" argument does not apply to directed networks.*

3.2.2 Computational Relations

Besides set-theoretical relations (*i.e.,* inclusions) among classes of graphs, *e.g.,* $\mathcal{E}^{\mathcal{P}} \subset \mathcal{E}^{\mathcal{B}} \subset \mathcal{E}^{\mathcal{R}}$, we considered with Flocchini, Mans, and Santoro [66] (partially reviewed in Section 2.2), the *computational* relations induced by various combinations of these classes with given knowledge, namely the number n of nodes in the network, a bound Δ on the recurrence time, and the period p. In particular, we considered the relations between $\mathscr{P}(\mathcal{E}^{\mathcal{R}}{}_n)$, $\mathscr{P}(\mathcal{E}^{\mathcal{B}}{}_\Delta)$, and $\mathscr{P}(\mathcal{E}^{\mathcal{P}}{}_p)$, where $\mathscr{P}(\mathcal{C}_k)$ is the set of problems one can solve in class $\mathcal{C}$ with knowledge k. We showed that, in the considered model, it holds that

$$\mathscr{P}(\mathcal{E}^{\mathcal{R}}{}_n) \subset \mathscr{P}(\mathcal{E}^{\mathcal{B}}{}_\Delta) \subset \mathscr{P}(\mathcal{E}^{\mathcal{P}}{}_p) \tag{3.1}$$

In other words, the computational relations between these three contexts form a *strict* hierarchy. The fact that $\mathscr{P}(\mathcal{E}^{\mathcal{R}}{}_n) \subseteq \mathscr{P}(\mathcal{E}^{\mathcal{B}}{}_\Delta)$ comes from $\mathcal{E}^{\mathcal{B}} \subseteq \mathcal{E}^{\mathcal{R}}$ together with the fact that n is learnable in $\mathcal{E}^{\mathcal{B}}{}_\Delta$. In fact, the entire footprint can be learned in $\mathcal{E}^{\mathcal{B}}{}_\Delta$, making it a rather powerful setting. For example, $\mathcal{E}^{\mathcal{B}}{}_\Delta$ can emulate algorithms for synchronous static networks, by confining every round within a Δ window where all neighbors communicate. The strictness of inclusions in Equation 3.1 comes from the existence of at least one problem in each setting which is not solvable in the other, using again the example of foremost, shortest, and fastest broadcast with termination (covered in Section 2.2).

3.2.3 Stochastic Comparison

Some of the classes are incomparable from a set-theoretic (and a fortiori computational) perspective. For example, none of $\mathcal{E}^{\mathcal{R}}$, $\mathcal{TC}^{\mathcal{B}}$, or $\mathcal{C}^{\mathcal{R}}$ could be declared more general than the others, and the same holds for $\mathcal{E}^{1\forall}$ and $\mathcal{TC}$. While incomparable using set theory, one may look at different ordering relations, in particular stochastic ones.

Research avenue 11. *Compare the generality of (otherwise incomparable) classes of dynamic networks by measuring how long it takes for a random network (*e.g., *edge-markovian dynamic graph) to satify the corresponding property. For recurrent properties, one may look instead at how frequently the property is satisfied.*

3.2.4 Conclusion

Several open questions related to the relations between classes have been discussed in this section. In way of conclusion, we will simply mention some of the *uses* one could make of such a hierarchy. First, we believe that these relations have the potential

to guide an algorithm designer. Clearly, one should seek properties that offer sufficient structure to be exploited, while remaining as general as possible. Second, such hierarchy allows one to compare the requirements of several candidate solutions on a rigorous basis. For example, we reviewed in Section 2.1 two counting algorithms, one of which requires the graph to be in $\mathcal{K}$, while the other requires it to be in $\mathcal{J}^{\forall 1}$. Precisely, both algorithms were guaranteed to succeed in $\mathcal{K}$ and to fail outside of $\mathcal{J}^{\forall 1}$; in between, *i.e.,* in $\mathcal{J}^{\forall 1} \setminus \mathcal{K}$, the first algorithm *must* fail, while the second *could* succeed, implying that the second is more general. Third, knowing which class a network belongs to provides immediate information as to what problem can be solved within. Giving again an example based on the model in Section 2.1, if a network is (say) in $\mathcal{TC} \cap \mathcal{J}^{1\forall >}$, then we have immediately that (1) broadcast has some chances of success whatever the emitter (depending on the adversary) and (2) it is guaranteed to succeed for at least one emitter.

This back and forth movement between classes and problems also manifests in more elaborate contexts, as seen above with Equation 3.1, which together with the gradual feasibility of foremost, shortest, and fastest broadcast in these contexts, implies a hierarchy of difficulty among these problems (in terms of topological requirements). We believe hierarchies of this type have the potential to lead more equivalence results and formal comparison between problems, and between algorithms.

3.3 Testing properties automatically

Main articles: SIROCCO'09 [52], ALGOTEL'14 [26, 22], CIAC'15 [69], SIROCCO'17 [70] (long version in Theory of Computing Systems [71]), SNAMAS'11 [177], PhD Y. Neggaz [164].

This section is concerned with the *a posteriori* analysis of connectivity traces using centralized algorithm. While we have not carried ourselves such studies on real data, we have designed a number of algorithms in this perspective. We first discuss methodological aspects, then describe some of our contributions in this area. In particular, we review various basic reductions of temporal properties in dynamic graphs to standard graph properties introduced in [52], then focus on the particular case of computing transitive closures of journeys [26, 22]. Next, we present a general framework for computing parameters of sequence-based dynamic graphs. The framework was introduced in [69] and generalized in [70] (both being combined in [71]). It can be *instantiated* with specific operations to address the computation of various parameters without changing the high-level logic. Finally, we review some suggestions made in [177] concerning the analysis of two-fold dynamic phenomena, namely the (long-term) evolution of (short-term) temporal metrics.

3.3.1 A methodological framework

While not a central topic in [52], one of the perspectives discussed therein suggest a general approach to connect efforts in the analysis of distributed algorithms with practical considerations (usability of the algorithms). The workflow, depicted in Figure 3.3, considers temporal properties resulting from the analysis of distributed algorithms, on the one hand, and dynamic graphs resulting from either real-world measures (traces) or

artificial generation through mobility models. The main motivation is to assist the algorithm designer in making realistic assumptions and the algorithm deployer in chosing the appropriate algorithm for the target mobility context.

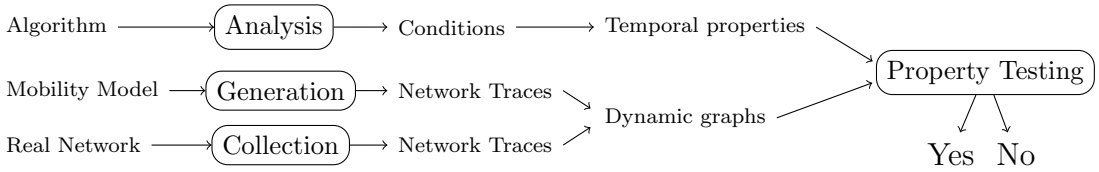


Figure 3.3: Suitability of a distributed algorithm in given mobility contexts.

This architecture has been the guiding principle (motivation) behind most of the work reported in this section.

3.3.2 Around transitive closures of journeys (and paths)

In [35], Bhadra and Ferreira define the concept of *transitive closure* of journeys as the static directed graph $H = (V, A_H)$, where $A_H = \{(v_i, v_j) : v_i \rightsquigarrow v_j)\}$ (see Figure 3.4). We

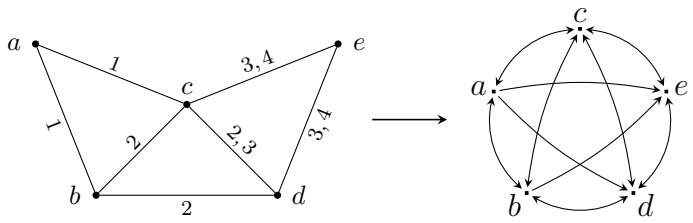


Figure 3.4: Transitive closure of journeys (here in discrete time).

considered in [52] two versions of this notion, whether the arcs in the closure represent strict or non-strict journeys (see Section 1.3 for definitions). Testing membership of a given graph to a number of basic classes listed above (based on *finite* properties) can be done using static derivated structures like this one. Precisely, given a graph $\mathcal{G}$, its footprint G, its transitive closure H, and strict transitive closure H_{strict}, membership to the seven classes discussed in Section 2.1 can be reduced to the following.

- $\mathcal{G} \in \mathcal{J}^{1\forall} \iff H$ contains an out-dominating set of size 1.
- $\mathcal{G} \in \mathcal{TC} \iff H$ is a complete graph.
- $\mathcal{G} \in \mathcal{J}^{1\forall >} \iff H_{strict}$ contains an out-dominating set of size 1.
- $\mathcal{G} \in \mathcal{TC}^{>} \iff H_{strict}$ is a complete graph.
- $\mathcal{G} \in \mathcal{E}^{1\forall} \iff G$ contains a dominating set of size 1.
- $\mathcal{G} \in \mathcal{K} \iff G$ is a complete graph.
- $\mathcal{G} \in \mathcal{J}^{\forall 1} \iff H$ contains an in-dominating set of size 1.

Computing the transitive closure

The problem of computing the transitive closure of a dynamic network varies slightly depending on whether strict or non-strict journeys are considered. We first focus on the case that all journeys must be strict (at most one hop at a time), then we show how to adapt the same solution to non-strict journeys based on a pre-processing trick of the input graph.

Let us first observe that existing algorithms for other problems could be adapted to compute the transitive closure. In particular, one may compute a tree of foremost journeys using the centralized algorithm in Bhadra et al. [42], doing so from *every node* and adding arc (u, v) iff v was reached when computing foremost journeys from u. Using the data structure from [42] (described further down), this amounts to $O(n(m \log k + n \log n))$ time steps, where k is the number of *characteristic* dates in the network (times of appearance or disappearance of edges).

In Barjon et al. [22] (French version [26]), we proposed an alternative algorithm for computing the transitive closure when the input graph is given as a sequence $\mathcal{G} = \{G_1, G_2, ..., G_k\}$, with $G_i = (V, E_i)$. The main feature of our algorithm is that it unifies strict and non-strict through a preprocessing step (described later). The algorithm is basic and consists of scanning the graphs in a chronological order, updating for each edge the union of predecessors on both endpoints nodes (to enforce strictness, the union is based on a copy of the predecessors obtained at the end of the previous time). The final set of predecessors is the transitive closure. Complexity parameters include the number of times k, the maximum *instant* density $\mu = max(|E_i|)$, and the *cumulative* density $m = |\cup E_i|$, yielding an overall cost of $O(k\mu n)$ operations. Depending on the values of the parameters, our algorithm achieves a better or worse complexity than the solution based on foremost trees. In particular, it performs better in sparse scenario where the *instant* density and the number of times are low (since we pay a linear dependency on k). A comparative table is given in [22] for all combination of parameters.

In the case of *non-strict* journeys no previous algorithm existed. Our algorithm is based on a double transitive closure: a *standard* transitive closure (*i.e.,* of paths) applied to each G_i, after which the resulting graph can be processed by the algorithm for strict closure (see Figure 3.5). Interestingly, the pre-processing remains contained within $O(k\mu n)$ operations, resulting in the same running time complexity (up to a constant factor).

More general objects. A generalization of the concept of transitive closure was proposed in Whitbeck et al. [189], which captures the reachability relative to *all* possible initiation times and durations. Once computed, the resulting structure makes it possible to quickly answer queries of the type *"can node u reach node b after time t and within d time units?"*. The complexity of their algorithm (based on an algebra of matrices) is strictly larger than ours (see [22]), but the object is more general.

Discussion 5 (Efficiency of data structures)**.** The algorithms in [42], [189], and [22] all use different data structures to represent the input network. In [42], the input is a *list of temporal adjacencies*, which is similar to an adjacency list augmented with nested lists of presence intervals (one for each edge, see Page 101 of [133] for details). In [189],

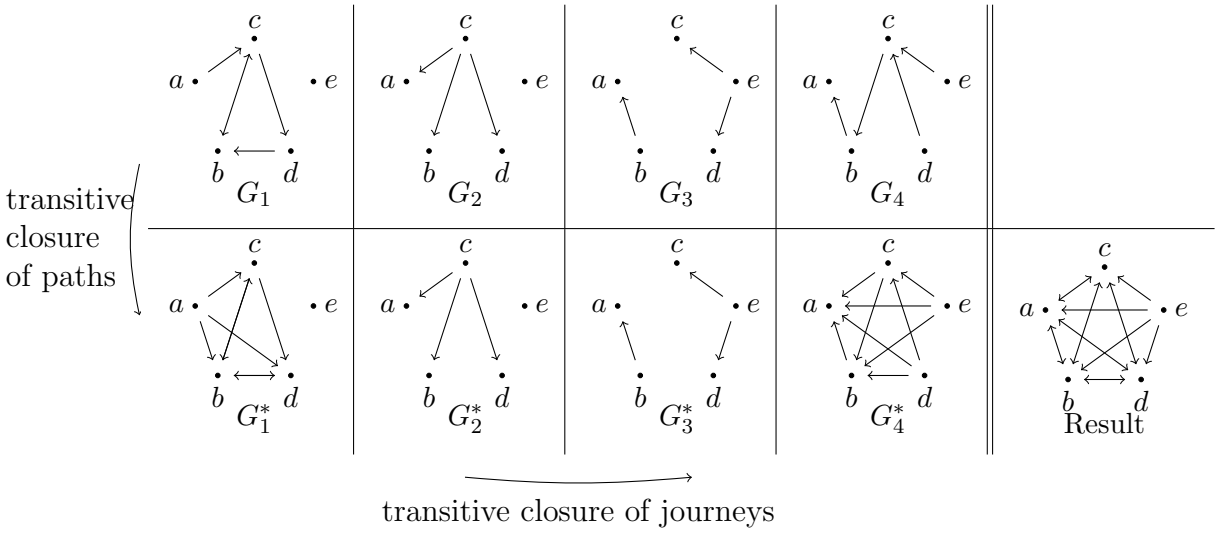


Figure 3.5: Computing the transitive closure of both strict and non-strict journeys.

the input is a time-indexed sequence of events of the type (u, v, UP), $(v, w, DOWN)$. Finally, in [22], the data structure is essentially a sequence of sets of edges $\{E_i\}$. The latter is admittedly a poor choice in terms of performance because it does not exploit stability, the status of an edge being duplicated even when it does not change. Similarly, any change in the network induces a new graph in the sequence. This has tremendous impact on the running time complexity of the above algorithms. Data structures based on sequences of links (link streams) like [189] or [150] often prove more efficient.

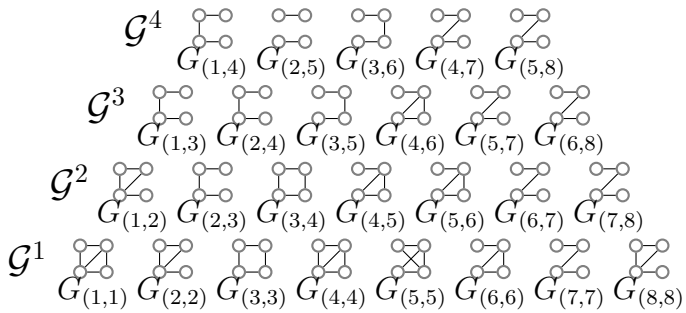
Despite Discussion 5, models based on sequences of graphs may be choices for high-level thinking and algorithmic design, as exemplified by the above pre-processing trick for non-strict journeys that may not be easily expressed in other models. Another illustration is the high-level strategies discussed next, which exploit the specificities of graph sequences.

Open question 12. *If both strict and non-strict journeys are considered, then some graphs cannot be obtained as transitive closure. For example, the graph* $G = (\{u, v, w\}, \{(u, v), (v, u), (v, w), (w, v)\})$ *(two-hop undirected path) cannot be a transitive closure, because it should include either* (u, w) *or* (w, u) *additionally (or both). Characterize the set of graphs which can be obtained as transitive closures.*

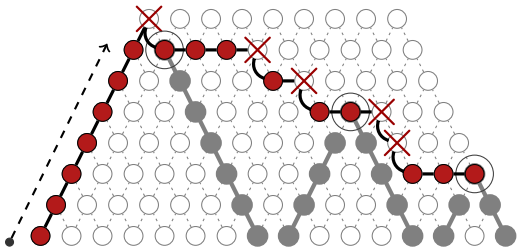
3.3.3 Computing parameters of sequence-based dynamic graphs

With Klasing, Neggaz, and Peters, we presented in [69] a high-level algorithm for finding the largest T such that a given sequence of graphs $\mathcal{G}$ is T-interval connected (see Section 3.1.2 for definitions) and the related decision version for given T. The approach is high-level, *i.e.,* the graphs in the sequence are considered as atomic elements that the algorithm manipulates through two abstract operations: *intersection* (of two graphs), and *connectivity testing* (of one graph). We showed that both the maximization and decision versions of the problem can be solved using only a number of such operations which is linear in the length δ of the sequence. The technique is based on a walk

through a lattice-like meta-graph which we call *intersection hierarchy*, the elements of which represent intersections of various sub-sequences of $\mathcal{G}$ (see Figure 3.6(a)). Then, the largest T for which the network is T-interval connected corresponds to the row in which the walk terminates (3 in both examples).



(a) Intersection Hierarchy



(b) Maximization walk

Figure 3.6: (a) Example of an intersection hierarchy (the first row is the input sequence itself). (b) Example of execution of the maximization algorithm. *The elements in gray and red are the only ones that need to be effectively computed; the gray ones are referred to as ladders.*

Then, with the same co-authors in [70], we generalized this framework for computing other parameters than T-interval connectivity. At a general level, the two operations are referred to as a *composition* operation and a *test* operation. The framework allows us to compute various parameters of dynamic graphs by simply changing the two operations, while using a high-level strategy for the construction and for the walk that are problem-insensitive (up to having one for maximization, one for minimization). The framework is illustrated in [70] through three other problems (in addition to T-interval connectivity, which are *bounded realization of the footprint, temporal diameter,* and *round trip temporal diameter.*

Realization of the Footprint. This property is motivated by Class $\mathcal{E}^{\mathcal{B}}$. Given a footprint G (whose vertex set is identified with that of $\mathcal{G}$), the problem is to find the smallest duration b such that in any window of length b in $\mathcal{G}$, all edges of G appear at least once. Here, the composition operation is the *union* of two graphs and the test operation is the *equality to* G. The row in which the walk terminates is the answer b.

Temporal Diameter. This property is motivated by Classes $\mathcal{TC}$ and $\mathcal{TC}^{\mathcal{B}}$. The problem is to find the smallest duration d such that every pair of node can communicate

through a journey in any window of length d. The hierarchy built here is one of *transitive closures* of journeys. More precisely, the composition operation is the *concatenation* of two transitive closures, and the test operation is *equality to the complete graph*. The row in which the walk terminates is the answer d.

Round-trip Temporal Diameter. This property is motivated by Class $\mathcal{TC}^{\circlearrowleft}$. The problem is to find the smallest duration d such that a *back-and-forth* journey exists between every two nodes. This problem is more complicated than searching two contiguous temporally connected subsequences: we must allow that the arrival of some forth journeys occur after the departure of other back journeys, making the problem more intricate. To solve the problem, we introduce the notion of round-trip transitive closure, in which every present arc (u, v) is additionally labelled with two extra times that indicate the earliest arrival and latest departure of journeys from u to v. The hierarchy built for this problem is one of *round-trip transitive closures*, using a special type of concatenation for composition, and equality to a complete graph for test, with the additional constraint that for the earliest arrival of every edge must be smaller than its latest departure.

Concluding remarks. These algorithms consider strict journeys by default, but they can be adapted to non-strict journeys by using the same pre-processing trick as above (illustrated on Figure 3.5). Other results from [70] include showing that the algorithms can work online with amortized $\Theta(1)$ composition and test operations for each graph of the sequence and proving constructively that some of the problems are in **NC** (NC is Nick's class, containing all problems solvable in polylogarithmic time on parallel machines with polynomially many processors).

3.3.4 Fine-grain dynamics *vs.* Coarse-grain dynamics

Social network analysis is traditionally concerned with measuring parameters related to the network as well as the *evolution* of these parameters over time. The problem of capturing the evolution of structural parameters (such as paths, connectivity, distances, and clustering coefficients) proceeds usually through aggregation windows. One considers a sequence of graphs $G_1, G_2, ...$ such that each G_i corresponds to the aggregation of interactions (often seen as a union of edges) taking place over some interval $[t_i, t'_i]$. In other words, every G_i is the footprint of some $\mathcal{G}_{[t_i,t'_i]}$ and then the main object of study becomes the sequence of these footprints. (Other terminologies in complex systems include *aggregated graphs* or *induced graphs*.) Note that choosing the appropriate duration for time windows in this type of investigation is a typically difficult problem (see *e.g.,* [145, 44, 151]), which we have not considered *per se*.

The case of temporal parameters like journeys or temporal distances is more complex because two different time scales are involved in their studies: (1) their very definition relies on time and does not survive aggregation, and (2) their average features may be themselves subject to long-term evolution. For example, questions like *how does the temporal distance evolve between nodes over large time scales?* embed two different references to time. Same for questions like *how does a network self-organize, optimize,*

or deteriorate over time in terms of temporal efficiency? We introduced in [177] a distinction between *fine-grain* and *coarse-grain* dynamics to capture this distinction. The coarse-grain evolution of fine-grain dynamics is studied through considering a *sequence* of *non-aggregated* graphs $\mathcal{G}_1, \mathcal{G}_2, ...$, each one corresponding to $\mathcal{G}_{[a,b]}$ for some interval $[a, b]$ (see Section 1.3 for definitions). Then temporal metrics can be mesured on each $\mathcal{G}_i$ and their long term evolution examined classically over the sequence.

Concluding remarks. This section reviewed some of our contributions around the question of testing properties of dynamic networks, given a trace of the communication links over some period of time. These contributions are of a *centralized* nature. However, the recognition of properties from a distributed point of view is a relevant question. In particular, if the nodes of a distributed network are able to learn properties about the underlying dynamic network, then this might help solve further problems. Some of our works reviewed in Section 2.2 relate to this topic, for example we explain that the nodes can learn the footprint of the network if the edges reappear within bounded time. From that, they can infer the number of nodes, which helps reduce in turn the message complexity of some problems. We also present in Section 4.2.3 a distributed algorithm for learning temporal distances among the nodes of a periodic network.

Research avenue 13. *Investigate further distributed problems pertaining to the learning of properties of the dynamic network (self-awareness), with the aim to exploit the corresponding structure and gained knowledge in the solving of other problems.*

3.4 From classes to movements (and back)

This section is more prospective; it discusses several links between real-world mobility and properties of dynamic networks. Fragments of these discussions appeared in a joint report with Flocchini in 2013 [57]; other fragments are recurrent in our own talks; however, most of the ideas were not yet systematically explored (whence the "prospective" adjective). In particular, the second part about movement synthesis prefigure a longer-term research program for us, some aspects of which are the object of a starting PhD by Jason Schoeters (since Nov. 2017).

3.4.1 Real-world mobility contexts

Real-world mobility contexts are varied, including for instance sensors, pedestrians, robots, drones, vehicles, or satellites. Each of these networks is of course dynamic, but in its own peculiar way. An interesting question is to understand what properties could be found in each type of network, making it possible to design better distributed algorithms that exploit this structure in more specific ways.

A number of such connections are illustrated in Figure 3.7, using the visual representation of the classes (see Table 3.1 on page 45). For instance, it is generally admitted that satellites have periodic movements (Class $\mathcal{E}^{\mathcal{P}}$), while sensor networks remain in general connected at any instant (Class $\mathcal{C}^*$). Interaction between smartphones may represent interactions between people in a given context, such as in a small company with

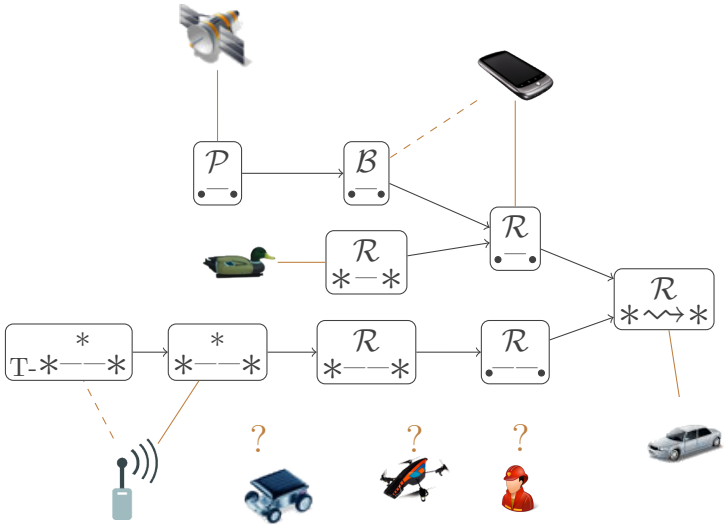


Figure 3.7: Properties of real-world mobility contexts.

bounded recurrence of edges, typically within a week (Class $\mathcal{E}^{\mathcal{B}}$), or in a community with unbounded, yet recurrent interactions (Class $\mathcal{E}^{\mathcal{R}}$). Vehicular networks exhibit an important range of densities and connectivity patterns, but they still offer recurrent connectivity over time and space (Class $\mathcal{TC}^{\mathcal{R}}$). Vehicles also share some traits with pedestrian through having their movements constrained by the environment (roads and pathways).

Through linking mobility contexts to graph properties, we expect to understand better what are the possibilities and limitations of each context, as well as enabling a more systematic transfer of results among the different contexts. It also help understand how networks of different natures could inter-operate.

Research avenue 14. *Conduct a more systematic study of what property (structure) could be expected in what context, using real mobility traces datasets like the ones from the CRAWDAD initiative [180].*

On a related note, studies measuring specific temporal metrics in such traces have been carried out recently, *e.g.,* reachability [189] and density of interaction [118].

3.4.2 Inducing structure through movements

Some types of mobility contexts like drones or robots have the convenient feature that the entities *control* their movements; that is, mobility is active rather than passive. Of course, the entities have a primary mission which determines to a large extent their patterns of movements. However, it is possible to *influence* their movements in order to force the satisfaction of certain properties. This general approach is often considered with respect to classical properties, *e.g.,* swarms of mobile entities moving while remaining connected all the time.

Going further, one may typically relax the connectivity constraint, making it possible for the network to operate in a disconnected mode with *temporal* connectivity constraints. Such properties would make movements less constrained while preserving

the ability to communicate within the group. Several temporal properties may actually be considered. Of particular interest are movements preserving a *bounded temporal diameter* (Class $\mathcal{TC}^{\mathcal{B}}$), because this makes it possible for the nodes to distinguish between a temporary isolated node and a crashed node (if the group does not hear from it for some time).

Research avenue 15. *Design collective mobility patterns whose resulting graph/network satisfies a number of temporal properties, possibly inspired from the classes reviewed in this chapter.*

This topic raises in turn a number of fundamental questions about movements, in particular about the way they should be approached in theoretical computer science. A lot of effort has been devoted to movements synthesis in control theory, with approaches ranging from analytical (here, we use "analytical" in the mathematical sense of analytic functions) to heuristics based on potential fields. We refer here in particular to movements satisfying *kinodynamic* constraints (acceleration, deceleration, and more generally inertia). It seems that little research was conducted in this direction so far from a discrete and combinatorial setting, which is however at the core of our algorithmic culture. (We do not include here a large body of literature about mobile robots in the distributed computing community, where the motivations and constraints are typically different, *e.g.,* forming fixed patterns, breaking symmetry, or gathering at a same point.)

Research avenue 16. *Explore discrete combinatorial models for movement synthesis which satisfy acceleration constraints and yet are simple enough to serve as a basis for theoretical investigation. Consider trajectory problems typically based on heuristics and simulations (*e.g., *path planning, obstacle avoidance), trying to tackle them with a more algorithmic approach (*e.g., *exact algorithms, reductions, approximations).*

In a recent work with M. Raffinot and J. Schoeters, we revisited the TSP problem subject to acceleration constraints modelled by a "Vector Racer"-like model. In such a model, the mobile entities can move by discrete amount (integral vector coordinates), and every next vector cannot diverge by much from the previous one. This approach made it possible for us to design *reductions* from and to the TSP with inertia constraints to other versions of TSP. The years to come will likely see the emergence of an algorithmic science of movements based on similar *discrete* approaches, motivated by the fact that mobile entities with controlled movements are becoming commonplace. On a more personal note, we find the topic quite exciting and forsee many interesting connections to temporal properties being possibly induced by movements in the resulting network. (Another recent work considering such model for an *s*-*t* trajectory optimization is [31].)

Chapter 4

Beyond structure

This chapter reviews several contributions not directly related to the presence of *structure* in dynamic networks, which are however fully integrated within the same lines of work. In fact, some sections relate explicitly to the *absence* of structure. We first review a series of contributions around the topic of maintaining a distributed spanning forest in a highly-dynamic network when no assumption is made on the underlying dynamics. This topic is a recurrent source of interest to us since the introduction of the abstract scheme in [45]; it has since then generated many developments with various co-authors (listed below) [25, 168, 53]), covering different aspects of the problem. Next, we present an algorithm called T-Clocks, elaborated with Flocchini, Mans, and Santoro through [63] and [65], which makes it possible to measure temporal distances in a distributed (and continuous-time) setting. While the algorithm requires no particular structure on the underlying dynamics, it has interesting applications when more structure *is* available (*e.g.,* if the network is periodic). Then, we present a theoretical investigation about the expressivity of time-varying graphs in terms of automata and languages, carried out with Flocchini, Godard, Santoro, and Yamashita in [59, 61, 60], focusing on the impact of being able to buffer information at intermediate nodes (along journeys). We conclude with a small collection of interesting facts which we have collected over the years, in which the dynamic feature seems to have an impact on the very nature of things.

4.1 Spanning forest without assumptions

Main articles: ICAS'06 [45], ALGOTEL'10 [168], ADHOCNOW'13 [53], OPODIS'14 [24] (long version in The Computer Journal [25]), PhD Y. Neggaz [164], PhD M. Barjon [23]

We reviewed in Section 1.4 several ways to reformulate the spanning tree problem in highly-dynamic networks. Among the possible ways, one is as the *maintenance* of a set of trees, containing only edges that exist at the considered time (see Figure 4.1 for an illustration). The formulation of spanning trees as a maintenance problem is quite standard in the area of "dynamic graph algorithms". In constrast, here, the setting is distributed and the network is typically disconnected, meaning that the best configuration is a single tree per component. Furthermore, the nodes may not be given

enough time between the changes to converge to a single tree per component. In fact, it

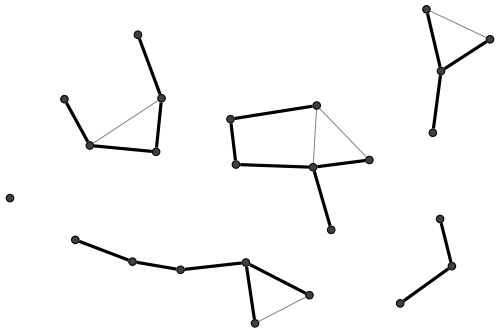


Figure 4.1: A spanning forest in a disconnected graph.

may even be the case that convergence never occurs. This section reviews our attempts at understanding what can still be done in such a context, with a focus on solutions where all decisions are taken *immediately*, without consulting the rest of the component.

4.1.1 Background

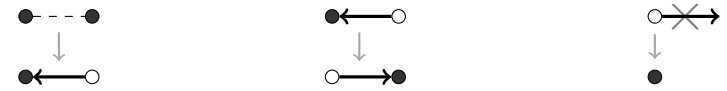
We introduced in 2006 [45] a high level (abstract) algorithm for this problem, whose initial purpose was to illustrate the automatic generation of code based on relabeling rules (a somewhat unrelated topic). Several versions of this scheme were subsequently studied; Guinand and Pigné realized some experimentations on random and deterministic variants, with or without backtracking, leading to a short joint account of experimental results in [168] (in French). Then, we established the correctness of the algorithm in the abstract model three years later in [53]. More recently, together with Johnen and Neggaz, we showed in [24] (long version [25]) that this algorithm can be adapted in the synchronous message-passing model, at the cost of substantial sophistications and a new technique for dealing with non-atomicity. The article includes simulation results obtained together with Barjon and Chaumette, showing that the algorithm is relevant in a practical scenario (based on real mobility datasets).

A number of other works in the literature considered spanning trees in dynamic networks, with various limitations on the dynamics [43, 144, 2, 18, 33, 16], with a special mention to [16], in which the restrictions are very mild (and the computed trees are minimum). We refer the reader to [25] for a more detailed description of these works and how they relate to our solution.

4.1.2 The algorithm

The algorithm in [45] was specified at a high-level of abstraction, based on a coarse grain interaction model inspired from graph relabeling systems [152], where interaction occurs among two neighbors in an atomic way (the model is different from population protocols in that the scheduler does not abstract dynamism, but only communication, see Section 2.1 for a discussion on these two models). The general principle is as follows. Initially every node has a token, meaning that it is the *root* of a tree (initially, its own tree). The first rule (merging rule on Figure 4.2) specifies that if two neighbors having

a token interact, then their trees are merged and one of the node becomes the parent of the other (which also loses its token). In absence of merging opportunity, the tokens execute a walk (typically random, but not necessarily) *within* the tree in the search for other merging opportunities (circulation rule). The circulation of the token flips the direction of relations in the tree, so that the node having the token is always the root of its tree, and the tree is correctly rooted (all relations point towards the root). The fact that the walk uses only edges in the tree is crucial, because it makes it possible to regenerate a token *immediately* when an edge of the tree disappears (regeneration rule, on the child side). Indeed, the child side of a disappeared edge knows that its subtree is token free; it can regenerate a token without concertation and in a seemless way for the rest of its subtree. As a result, both mergings and regenerations are purely localized and immediate decisions.



(a) Merging rule (b) Circulation rule (c) Regeneration rule

Figure 4.2: Spanning forest principle (high-level representation). *Black nodes are those having a token. Black directed edges denote child-to-parent relationships. Gray vertical arrows represent transitions.*

Properties. We proved in [53] that the above scheme guarantees that, at any time, the network remains covered by a set of trees such that 1) no cycle exists, 2) every node belongs to a tree, and 3) exactly one node has a token in every tree and all the edges of the tree are properly directed towards it. These properties hold whatever the chaos of topological changes. One might object here that an algorithm doing basically nothing also satisfies these requirements, which is correct. The above scheme also guarantees that (with slight care given to the implementation), the process eventually converges towards a single tree per connected component if the network stops changing at some point. In some sense, it is a *best effort* algorithm. Correctness was much more difficult to establish in the case of the message-passing version; it occupied us for months and was eventually done successfully [25].

Speed of convergence. Little is known about the speed of convergence of (any version of) this algorithm. As already discussed, it may so happen that the rate of change is too fast for the algorithm to converge. Furthermore, the network may be partitioned at all time steps. We suggested in [168] that an appropriate metric in such cases is the *number of trees per connected component*, considered for example in a stationary regime (in case of probabilistic investigation) or over a real-world mobility trace (in case of simulations, which is exactly what we did in [25]).

Open question 17. *Understand the theoretical complexity of this process. Based on discussions with knowledgeable colleagues in related areas (in particular T. Radzik, D. Sohier, and J.-F. Marckert), even a static analogue of this process based only on the merging and circulation operations has not been considered in standard graphs. These operations thus define a new form of coalescing process whose study remains to be done.*

We made preliminary observations in this direction in [53], however the problem remains mostly open. Because the algorithmic principle is high-level, an ambiguity arises as to the exact behavior of the process: is it continuous or discrete? is it synchronous or asynchronous? what priority rules apply in case of ties? *etc.* These parameters have an impact on the analysis. However, we believe that the choice among them should be precisely driven by simplicity. In the longer term, the objective is to understand the dynamic version (with edge dynamics and regeneration rule) in a stochastic model of dynamic network like edge-markovian dynamic graphs [91], characterizing *e.g.,* the number of trees per connected components in a stationary regime, as a function of birth and death rates for edges. We refer the reader to [21, 131, 7, 94, 41] for references on coalescing processes in general.

4.2 Measuring temporal distances distributedly

Main articles: IPDPS'11 [63], arXiv'12 [64], IEEE Trans. Computers'14 [65]

We mentioned in Chapter 1 a number of temporal concepts, among which that of *temporal distance* that accounts for the time it takes for entities of a dynamic network to reach each others. We present here a joint work with Flocchini, Mans, and Santoro [63] in which we design *distributed* algorithms for the nodes to learn temporal distances to each other in a unstructured dynamic network. The time setting is *continuous*; contacts may have arbitrary durations; and the latency of individual contacts is non-zero, altogether making the problem more complex. Our solution relies on the definition of an abstraction called T-Clocks, which enables to solve other problems in periodically-varying networks.

4.2.1 Overview

The problem of measuring temporal distances in dynamic networks was addressed in a number of works from the field of social network analysis [127, 142, 141]. In particular, Kossinets, Kleinberg, and Watts [141] flip the point of view, asking how "out-of-date" an entity (node) v is with respect to every other node at a time t. They define a concept of *view* as the answer table to this question, the indices being the nodes and the values indicating, for every other node u, the latest time at which u could send a piece of information arriving at v before t (*i.e.,* latest time of influence). The setting is centralized and contacts among entities are punctual, which implies that the views only changes punctually by discrete amounts. Finally, the speed of propagation along a contact (edge latency) is neglecte as well.

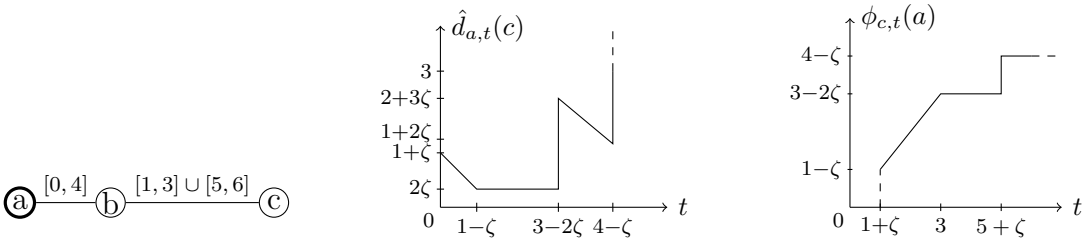
In [63], we considered the *distributed* version of this problem in a *continuous* time setting where contacts can have arbitrary *durations*. We also relaxed the assumption that latency is instantaneous, by considering an arbitrary (but fixed) latency. We construct an abstraction called T-Clocks, which every node can use to know, at any point in time, how out-of-date it is with respect to every other node. The main difficulty stems from considering continuous time together with non punctual contacts, the combination of which induces *continuums* of *direct* journeys (direct journeys are journeys realized without pauses at intermediate nodes) which induces continuous evolution of

the view (more below). The edge latencies also complicates things by making it possible for an direct journey to be actually slower than an indirect one (indirect journeys are such that a pause is made at least at one intermediate node).

The use of T-Clocks is illustrated through solving two problems in periodically varying networks (class $\mathcal{E}^{\mathcal{P}}$), namely building reusable *foremost broadcast trees* (same article [63]), and building *fastest broadcast trees* (in [64]), both being combined in a long version [65]. The latter reduces to the problems of (i) finding minimum time of *temporal eccentricity* (*i.e.,* when the duration needed to reach all other nodes is the smallest), then (ii) building a *foremost* broadcast tree for this particular time (modulo the period p). Next, we briefly discuss some of the key technical aspects.

4.2.2 Temporal views

In [63, 65] we refer to the concept of view as *temporal views* to avoid conflicts with the (unrelated) views of Yamashita and Kameda [190]. As explained, considering non punctual contacts implies the possible co-existence of indirect journeys, and *continuums* of direct journeys on the other hand. While both can be dealt with in the same way in discrete time, their combination in continuous time produces complex patterns of temporal distances among nodes, as illustrated on Figure 4.3 (from [65]).



(a) A dynamic network

(b) Temporal distance from a to c (seen from a)

(c) Temporal view that c has of a (*i.e.,* flipped point of view)

Figure 4.3: Temporal distance and temporal views as a function of time *(with $\zeta \ll 1$)*.

Temporal distance and temporal view actually refer to the same quantity seen from different perspectives: the first is a *duration* seen from the emitter, while the second is a time seen from the receiver.

4.2.3 The T-Clocks abstraction

Direct journeys are often faster than indirect ones, but this is not always true. As a result, the temporal view ϕ that a node has (of another node) could result from either types of journey. The algorithm we introduced tracks the evolution of both types of views separately. While indirect views need only be updated punctually (in a similar way as in [141]), direct views evolve continuously, and are thus inferred, on demand, from the knowledge of the (currently best) number of hops in contemporaneous journeys from a considered node (say u) to the local node v. We call this hop distance the *level* of v with respect to u. The T-Clocks algorithm consists of maintaining up-to-date

information at each node v relative to its *level* (for direct views), and latest *date* of influence (for indirect views) relative to every other node u. This work assumed the same computational model as in [66] (described in Section 2.2); in particular, it relies on presence oracle to detect the appearance and disappearance of incident edges. The resulting abstraction is illustrated on Figure 4.4.

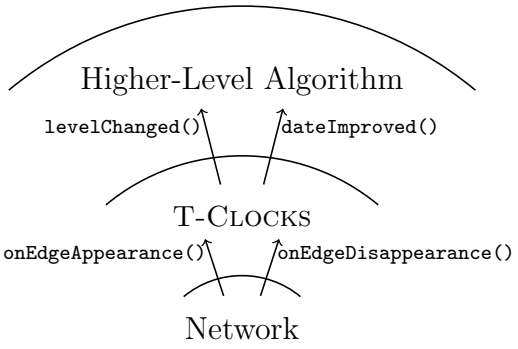


Figure 4.4: T-Clocks as an abstraction to track temporal views. Picture from [65].

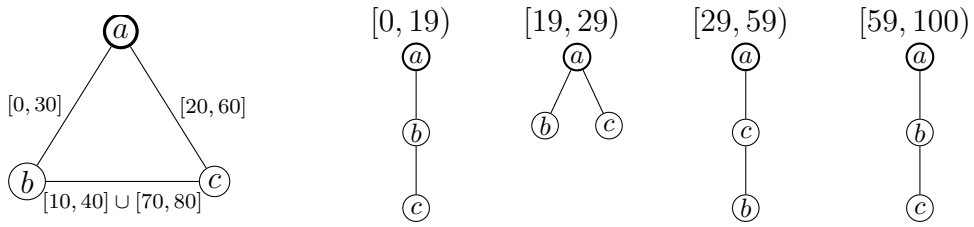
4.2.4 Using T-Clocks in periodically-varying networks

Using T-Clocks, it becomes easier to build foremost or fastest broadcast trees in periodically-varying networks ($\mathcal{E}^{\mathcal{P}}$), thereby completing the results presented in Section 2.2. We briefly review how T-Clocks were used to build foremost broadcast trees in [63] and fastest broadcast trees in [64] (both being combined in [65]).

Foremost broadcast trees in $\mathcal{E}^{\mathcal{P}}$. These trees indicate what structural path (routing choice) a message initiated at a given emitter (here node a) should follow to arrive at every other node (here, b and c) at the earliest possible time. Examples of such trees are shown on Figure 4.5, where the intervals correspond to when the message is *initiated* at the emitter (rather than when the routing is made). Some edges may not be available directly, in which case the message should be sent upon their next appearance. Solving this problem with T-Clocks boils down to monitoring at each node (here, b and c) the evolution of the temporal view with respect to the considered emitter (here, node a). Whether due to direct and indirect journeys, T-Clocks make it possible to track which local neighbors are responsible for providing the best view of the emitter over a complete period p, these neighbors being then selected as foremost parent for the corresponding times (modulo p).

Fastest broadcast trees in $\mathcal{E}^{\mathcal{P}}$. Fastest journeys minimize the span between first emission and last reception (possibly at the cost of delaying the first emission). Unlike the foremost quality, it is not always possible to find a tree in which *every* journey is fastest; this kind of "optimal substructure" feature applied to the foremost metric, making the definition of a tree more natural. We thus define a fastest broadcast tree as one (among the possibly many) which minimizes time between first emission at the emitter and last reception everywhere.

So defined, an interesting observation is that at least one of the *foremost* broadcast trees must also be *fastest*. In fact, the considered problem reduces to (i) finding the time



(a) A periodic graph with period 100.

(b) Foremost broadcast trees from node a, depending on the initiation time (modulo 100), assuming edge latency $\zeta = 1$.

Figure 4.5: Example of foremost broadcast trees from a given node, as a function of the initiation times at the emitter (adapted from [65]).

of minimum temporal eccentricity of the emitter (modulo the period p), and (ii) build a foremost broadcast tree for this initiation time. Since the construction of foremost broadcast tree is already solved by the algorithm presented above, we focus on the problem of determining the time of minimum temporal eccentricity using T-Clocks. Note that this problem is interesting in its own right, and may be used as a building block for other tasks than broadcasting (*e.g.,* electing a leader based on its ability to reach other nodes quickly).

The algorithm consists of monitoring the evolution of temporal views at each node over a complete period (relative to the given emitter). The resulting information is converted back to temporal distance information, which are then aggreated back to the emitter, which eventually knows its eccentricity over a complete period. Finally, the emitter choses any time of minimum eccentricity to initiate the construction of a foremost (and thus here fastest) broadcast tree.

4.3 The power of waiting

Main articles: PODC'12 [59], FCT'13 [61], TCS'15 [60]

Together with Flocchini, Godard, Santoro, and Yamashita in [60] (brief anouncement [59] and conference version [61]), we explored the connections between dynamic networks and computability in terms of formal languages. Focusing on the particular case of time-varying graphs (see Section 1.2.1 for definitions), we showed how the manipulation of a time-varying graph as an *automata* makes it possible to study the power of *buffering* in dynamic networks (that is, the ability for a node to store and carry information before retransmitting it). In summary, we show that the set of languages that can be generated if only direct journeys are allowed (no waiting/buffering) contains all computable languages, whereas the set of languages if waiting is allowed (indirect journeys are possible) it is just the family of *regular* languages. In other words, when waiting is allowed, the expressivity of the environment drops drastically from being as powerful as a Turing machine, to becoming that of a Finite-State machine, which gives a (admittedly abstract) idea of the importance of buffering in dynamic networks.

4.3.1 Automata based on time-varying graphs

Given a dynamic network modeled as a time-varying graph $\mathcal{G}$, a journey in $\mathcal{G}$ can be viewed as a word in the alphabet of the edge labels. In this light, the class of feasible journeys defines the language $L_f(\mathcal{G})$ expressed by $\mathcal{G}$, where $f \in \{wait, nowait\}$ indicates whether or not indirect journeys are considered feasible by the environment. Hence, a TVG $\mathcal{G}$ whose edges are labelled over Σ, can be viewed as a TVG-automaton $\mathcal{A}(\mathcal{G}) = (\Sigma, S, I, \mathcal{E}, F)$ where Σ is the input *alphabet*; $S = V$ is the set of *states*; $I \subseteq S$ is the set of *initial states*; $F \subseteq S$ is the set of *accepting states*; and $\mathcal{E} \subseteq S \times \mathcal{T} \times \Sigma \times S \times \mathcal{T}$ is the set of *transitions* such that $(s, t, a, s', t') \in \mathcal{E}$ iff $\exists e = (s, s', a) \in E : \rho(e, t) = 1, \zeta(e, t) = t' - t$.

Figure 4.6 shows an example of a deterministic TVG-automaton $\mathcal{A}(\mathcal{G})$ that recognizes the context-free language $a^n b^n$ for $n \geq 1$ (using only direct journeys). The presence and latency of the edges of $\mathcal{G}$ are specified in Table 4.1, where p and q are two distinct prime numbers greater than 1, v_0 is the initial state, v_2 is the accepting state, and reading starts at time $t = 1$.

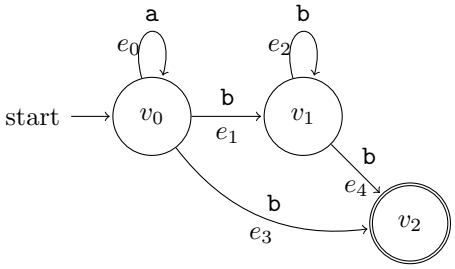


Figure 4.6: A TVG-automaton $\mathcal{A}(\mathcal{G})$ such that $L_{nowait}(\mathcal{G}) = \{a^n b^n : n \geq 1\}$. [60]

e	Presence $\rho(e,t) = 1$ iff		Latency $\zeta(e,t)$ =
e_0	always true		$(p-1)t$
e_1	$t > p$		$(q-1)t$
e_2	$t \neq p^i q^{i-1}, i > 1$		$(q-1)t$
e_3	$t = p$		any
e_4	$t = p^i q^{i-1}$, $i > 1$		any

Table 4.1: Presence and latency functions for the edges of $\mathcal{G}$.

The reader may have noticed the basic principle employed here, which consists of using latencies as a mean to *encode* words into time, and presences as a means to *select* them through opening the appropriate edges at the appropriate time.

4.3.2 Summary of the results

We characterize the two sets of languages $\mathcal{L}_{nowait} = \{L_{nowait}(\mathcal{G}) : \mathcal{G} \in \mathcal{U}\}$ and $\mathcal{L}_{wait} = \{L_{wait}(\mathcal{G}) : \mathcal{G} \in \mathcal{U}\}$, where $\mathcal{U}$ is the set of all time-varying graphs; that is, the languages expressed when waiting is or is not allowed. For each of these two sets, the complexity of recognizing any language in the set (that is, the computational power needed by the accepting automaton) defines the level of difficulty of the environment. We first study

the expressivity of time-varying graphs when waiting is *not* allowed—that is, the only feasible journeys are direct ones—and prove that the set $\mathcal{L}_{nowait}$ contains all computable languages. The proof is constructive and shows how any given Turing machine can be simulated by a time-varying graph.

Corollary 3.3 in [60]. For any computable language L, there exists a time-varying graph $\mathcal{G}$ such that $L = L_{nowait}(\mathcal{G})$.

We next examined the expressivity of time-varying graphs if *indirect* journeys are allowed; that is, entities have the choice to wait for future opportunities of interaction rather than seizing only those that are directly available. In striking contrast with the non-waiting case, the languages $\mathcal{L}_{wait}$ recognized by TVG-automata are precisely the set of *regular* languages. In other words, when waiting is no longer forbidden, the power of the accepting automaton (*i.e.,* the difficulty of the environment, the power of the adversary), drops drastically from being at least as powerful as a Turing machine, to becoming that of a finite state machine. This gap is a measure of the computational power of waiting. (This result may appear counterintuitive, but the fact that the power of the environment/adversary drops is in fact a good news for the algorithm, which is a more intuitive way to think of this result.)

Theorem 4.13 in [60]. $\mathcal{L}_{wait}$ is the set of regular languages.

To better understand the power of waiting, we then turn our attention to *bounded waiting*—that is, when indirect journeys are considered feasible if the pause between consecutive edges in the journeys have a bounded duration $d > 0$. In other words, at each step of the journey, waiting is allowed only for at most d time units. We examine the set $\mathcal{L}_{wait[d]}$ of the languages expressed by TVG in this case and prove the negative result that the complexity of the environment is not affected by allowing waiting for a limited amount of time, that is, for any fixed $d \geq 0$, $\mathcal{L}_{wait[d]} = \mathcal{L}_{nowait}$. As a result, the power of the adversary is decreased only if it has no control over the *length* of waiting, *i.e.,* if the waiting is unpredictable.

Theorem 5.1 in [60]. For any fixed $d \geq 0$, $\mathcal{L}_{wait[d]} = \mathcal{L}_{nowait}$.

The basic idea of the proof is to reuse the same technique as for the nowait case, but with a dilatation of time, *i.e.,* given the bound d, the edge schedule is time-expanded by a factor d (and thus no new choice of transition is created compared to the no-waiting case). As a result, the power of the adversary is decreased only if it has no control over the length of waiting, *i.e.,* if the waiting is unpredictable.

Open question 18. *Understand the significance of this technical result. Depending on the point of view, say (1) network designer or (2) edge presence adversary, the interpretation of the result is fundamentally different. In (1) it is a good news, since it gives more expressivity to design interactions in the network, but in (2) it implies that a distributed algorithm may be forced to act in a way imposed by the environment. The overall implications are not yet well understood.*

Open question 19. *Do these results have analogues in the area of* timed automata*, which are automata where transitions are subject to time constraints. (See* e.g., *[9]).*

4.4 Collection of curiosities

We conclude this part of the document with a small collection of observations regarding concepts whose dynamic version (temporal version) seem different *in essence* to their static analogues, or whose implications are perhaps not yet well understood. Some of these observations are well-known or direct, others less obvious and some raise interesting questions.

4.4.1 Relation between paths and journeys

A first and simple fact is that having a connected footprint does not implies that the network is temporally connected, as exemplified by the graph on Figure 4.7 (from [68]).

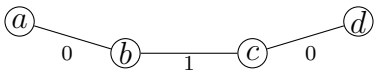


Figure 4.7: A graph whose footprint is connected, but which is not temporally connected.

Clementi and Pasquale ask the reader in [92] whether one can define a graph whose temporal diameter is large despite the fact that every snapshot has small diameter. This is indeed feasible. Using a broadcasting analogy in discrete time, one can design every snapshot as two cliques linked by a single edge, one of which represents informed nodes and the other non-informed nodes. In each step, the construction is updated based on the same rule (see Figure 4.8). Then, every snapshot has diameter 3, while the temporal diameter is $n-1$. Observe that the temporal diameter cannot exceed $n-1$

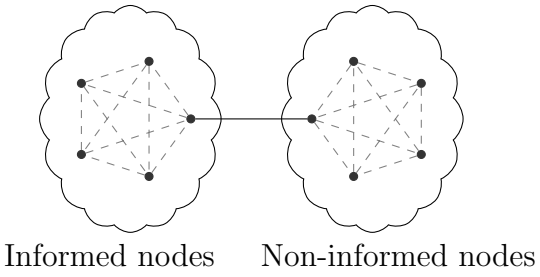


Figure 4.8: Small diameter vs. large temporal diameter.

in this setting, because at least one new node is informed in each time step. The latter argument applies more generally when all the snapshots are connected. In fact, it is the reason why $\mathcal{C}^* \subseteq \mathcal{TC}^{\mathcal{B}}$ in Section 3.2.

Hardness of computing the temporal diameter. Godard and Mazauric [122] consider the problem of computing the temporal diameter of dynamic networks in an offline, but uncertain setting. More precisely, the network is described by an *unknown* sequence of graphs, each snapshot of which belongs to a *known* set of possible graphs. Every next graph in the sequence is independent from the previous (time-homogeneity). In this context, they show that computing the worst-case temporal diameter of the

network is hard. In fact, it is inapproximable in general, and becomes approximable (but still NP-Hard) if all the possible graphs contain a common connected spanning subgraph. In contrast, computing the standard diameter of a standard graph has a low-polynomial complexity.

4.4.2 Connected components

There are several ways to consider the temporal analogue of a component in terms of journeys. If the lifetime is *infinite*, it seems reasonable to define them as maximal sets of nodes, each of which can reach the others *infinitely often* through a journey, either within bounded or unbounded time windows (this is the point of view we adopted in [123], reviewed in Section 2.3). If the lifetime is *finite*, one may rather define components as maximal sets of nodes, each of which can reach the others *at least once*.

The latter point of view is the one adopted by Bhadra and Ferreira in [35], which seems to be the first significant study on the subject (2003). Bhadra and Ferreira further distinguish between *open* and *closed* components, depending on whether the journeys can travel through nodes outside the component. The existence of open components may seem strange, but it follows naturally from the fact that transitivity does not apply to journeys. Bhadra and Ferreira also observe that maximal components may overlap, which we illustrate using a minimal example on Figure 4.9 where $\{a, b, c\}$ and $\{b, c, d\}$ are two maximal components (a and d cannot reach each other). Again, this contrasts

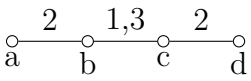


Figure 4.9: Two overlapping maximal components

with static graphs, in which maximal components *partition* the network (*i.e.,* they define equivalence classes among the nodes).

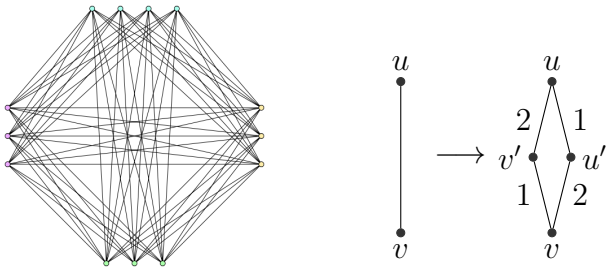
Exponential number?

In a static network (directed or undirected alike), the number of component cannot exceed $O(n)$ because components partition the nodes. The overlapping nature of components in dynamic graphs (see *e.g.,* Figure 4.9) raises the question of their number. In particular, can there be exponentially many maximal components?

Let us first examine the case that only strict journeys are allowed (*i.e.,*, latency is positive). Take a footprint that contains exponentially many maximal cliques, such as a Moon and Moser graph (Figure 4.10(a)), and make all of its edges appear at the same time for a single time unit of time (or ζ time). Then, only one-hop journeys may exist, implying that every maximal clique in the footprint induces a maximal component in the dynamic network. (This idea is inspired from a similar construct in [138] used to show that *every* edge of the footprint might be necessary to achieve temporal connectivity.)

This argument for exponentially many connected components does not hold if non-strict journeys are possible (latency is neglected). With Klasing, Neggaz, Peters, and

Renault [72], we designed a slightly more complex gadget that generalizes this argument to all types of journeys. Take again a Moon and Moser graph and replace every edge (u, v) with four edges (u, u'), (u', v), (v, v'), (v', u), where u' and v' are new nodes (see Figure 4.10). Give these edges presence times $1, 2, 1, 2$, respectively. (We call this a *semaphore* gadget.) Then, the arrival at u from v (resp. at v from u) is now too late for crossing another edge, turning original cliques into maximum components again. Moon and Moser graphs can have up to $3^{n/3} = 2^{\Theta(n)}$ maximal cliques. Our solution induces a quadratic blow up in the number of vertices (two extra vertices per edge), leading to $2^{\Theta(\sqrt{n})}$ maximal components.



(a) Moon and Moser graph (b) Semaphore gadget to replace every edge

Figure 4.10: Dynamic networks can have super-polynomially many maximal components. The argument holds for strict and non-strict journeys alike.

Computational complexity

Bhadra and Ferreira prove in [35] that finding a maximum size component in a dynamic network is NP-hard, based on a reduction from the clique problem. The reduction gadgets are different from our constructs above and no counting argument is invocated. Note that, while finding the largest component is hard, testing if a given set of nodes are temporally connected is clearly not and can be inferred directly from the transitive closure of journeys (itself efficiently computable, see Section 3.3.2).

On a related note, Viard et al. [187] look at the enumeration of temporal cliques, which in that case refer to cliques in the footprint of all $\mathcal{G}_{[t,t+\Delta]}$ for some Δ (sliding window), which are also exponentially many. This is however less surprisingly, since cliques can already be in exponential number in static graphs (unlike components).

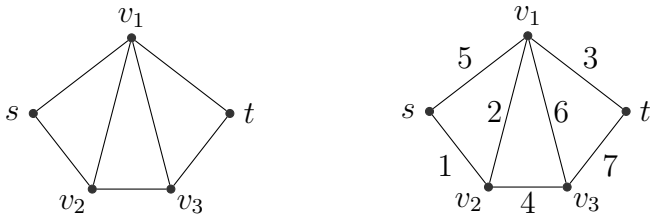
How about recurrent components?

Interestingly, if we consider the above *recurrent* version of connected components, which requires that journeys exist infinitely often among the nodes, then transitivity comes back in the picture and every two overlapping components collapse into one. Connected components in this case amount to (standard) components in the *eventual footprint*. Recurrent components are in this sense much closer to their classical analogue than non-recurrent ones.

Research avenue 20. *Make a more systematic study of these questions.*

4.4.3 Menger's theorem

In the context of static networks, Menger's theorem [160] states that the maximum number of node-disjoint paths between any two nodes s and t is equal to the minimum number of nodes needed to separate them. For example, in the *static* network depicted on Figure 4.11(a), two disjoint paths exist between s and t, namely (s, v_1, t) and (s, v_2, v_3, t), and indeed at least two nodes must be removed to disconnect s and t (*e.g.,* v_1 and v_2). In [32], Berman observe that the natural analogue of Menger's theorem does not hold in dynamic networks. We reproduce here an example from [138] (Figure 4.11(b)), showing a dynamic network in which the theorem fails: any two journeys must share a node, and yet two nodes are needed to separate s and t. Intuitively,



(a) A static networks (b) A dynamic networks

Figure 4.11: Illustration of Menger's theorem in both a static and a dynamic network (adapted from [138])

the times are used to *force* a journey to use certain edges, which prevents journeys whose underlying paths are the same as above. Note that some *footprints* (characterized in [138]) have the nice property that the resulting network remains Mengerian whatever the presence schedule of the edges.

4.4.4 Degrees and density

Bramas and Tixeuil [39] study the problem of data aggregation in a dynamic wireless sensor network where collisions may occur. The goal is to find a collision-free schedule that aggregates data from all the nodes to a single designated node in minimum time. This problem is known to be NP-complete in static networks, and so, even when the degree is restricted to 4 [90] (improved to 3 in [39]). Interestingly, Bramas and Tixeuil prove in the same paper that although the problem is trivial in static networks with degree at most 2, it is NP-hard in dynamic networks whose degree is never more than 2 at a single time. This result again tells us something about essential differences between static and dynamic contexts. In this case, however, it must be pointed out that the *footprint* of the graph used in their reduction does have degree more than 2.

Open question 21. *See if a different reduction exists that limits the degree to* 2 *even in the footprint, making an even more essential separation between static and dynamic networks.*

Cumulated *vs.* instant degrees. On a related note, with Barjon *et al.*, we characterized in [22] the complexity of an algorithm for dynamic networks based on two

types of densities: the maximum *instant* density (largest amount of edges existing at a same time) and the *cumulative* density (number of edges in the footprint), both being possibly much different. The same distinction applies between both types of *degrees*, and in some sense, the above question (Open question 21) embodies this distinction.

4.4.5 Optimality of journeys prefixes

Bhadra et al. observed in [42] that *foremost* journeys have the convenient feature that all prefixes of a foremost journey are themselves foremost journeys. This feature allows us to define a distributed version of the problems in [42] in terms of foremost broadcast *trees*, all parts of which are themselves foremost (reviewed in Sections 2.2 and 4.2). Unfortunately (and interestingly), this feature does not apply to the *fastest* metric, pushing us to define fastest broadcast trees in a different way (minimizing the overall span between first emission and last reception). We give in Figure 4.12 a concrete example where the fastest quality is not prefix-stable, which adapts a continuous-time example we gave in [65] into the discrete time setting (for simplicity). To see this,

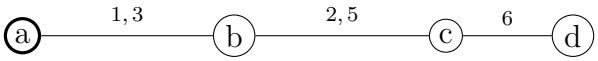


Figure 4.12: A graph in which prefixes of fastest journeys are not themselves fastest.

consider the fastest journey from node a to node d, namely $\{(ab, 3), (bc, 5), (cd, 6)\}$. One of the prefixes of this journey, namely $\{(ab, 3), (bc, 5)\}$ is not itself a fastest journey, since $\{(ab, 1), (bc, 2)\}$ is faster. (Note that using the latter as an alternative prefix towards d would lead to a non-fastest journey.)

Part II

Excursions

Chapter 5

JBotSim

Main article: SimuTOOLS (2015) [48].
Other resources: `http://jbotsim.io`,
`https://github.com/acasteigts/jbotsim`

JBotSim is a java library which I developed over the past ten years. It offers basic primitives for prototyping, running, and visualizing distributed algorithms in dynamic networks. With JBotSim, one can implement an idea in minutes and interact with it (*e.g.,* add, move, or delete nodes) while the algorithm is running. It is well suited to prepare live demonstrations of algorithms to colleagues or students, as well as to evaluate the performance of algorithms at a high-level (number of messages, number of rounds, etc.). Unlike most tools, JBotSim is not a runnable software, but a *library* to be used in (otherwise independent) programs. This chapter gives an overview of JBotSim's distinctive features and ecosystem.

5.1 Introduction

JBotSim is an open source (LGPL) simulation library dedicated to distributed algorithms in dynamic networks. It is not a competitor of mainstream simulators such as NS3 [126], OMNet [186], or The One [140], in the sense that it does not implement real-world networking stacks nor does it aim at physical and technological realism. Quite the opposite, JBotSim aims to be *technology-insensitive* and used mostly at the algorithmic level. In this respect, it is closer to the ViSiDiA project [28, 97], a visualization tool for graph-based distributed algorithms in *static* networks, or to multi-agent simulation tools.

Whether an algorithm is centralized or distributed, the natural way of programming in JBotSim is event-driven: algorithms are defined as methods to be executed when particular events occur (reception of a message, appearance or disappearance of a communication link, timer pulse, *etc.*). Movements of the nodes can be controlled either by program or by means of live interaction with the mouse (adding, deleting, or moving nodes around with the mouse). Interactivity is an important aspect of JBotSim, allowing for these movements to be performed during the execution, in order to test and visualize the behavior of the algorithm in various configurations. Besides its features, the main asset of JBotSim is its simplicity of use.

5.1.1 Basic use

One can obtain JBotSim by fetching the last official release as a JAR package from the official website (`http://jbotsim.io`), or by cloning the source code on GitHub (`https://github.com/acasteigts/jbotsim`). Here is a minimal program.

Listing 1 HelloWorld with JBotSim

```
import jbotsim.Topology;
import jbotsim.ui.JViewer;

public class HelloWorld{
    public static void main(String[] args){
        Topology topology = new Topology();
        new JViewer(topology);
    }
}
```

This example shows two central components in JBotSim's architecture. The `Topology` plays the role of container for objects of type `Node` and `Link`; in other words, it *is* the graph. It also organizes the life of the system, managing the internal clock and the message engine, notifying listeners from changes in the system, and updating the links if need be according to various policies (*e.g.,* distance between nodes). The `JViewer` is a graphical component which allows one to visualize the topology and interact with it based on an underlying `JTopology` object that renders the topology and manages interactions. Among the interactions, one can add nodes (left click), delete them (right click), or move them around (drag & drop) with the mouse. Significantly, these actions can be performed *during* the execution and induce algorithmic reactions.

5.1.2 Distributed *vs.* centralized algorithms

Distributed algorithms (*aka* node algorithms) are created in two steps: (1) Create a class that extends class `Node`, and (2) Override some methods, as illustrated in Listing 2. Most algorithms can be specified using 2 or 3 such methods. JBotSim will automatically call them in appropriate moments (similarly to the methods of an `Activity` in an android application). Other basic events to be overridden in class `Node` include `onStop`, `onPreClock`, `onPostClock`, `onMove`, `onLinkAdded`, `onLinkRemoved`, `onSensingIn`, and `onSensingOut`. The user can extend these with extra variables and methods. Then a simulation involves a number of nodes of one or several such types.

If the target algorithm (or part of it) is centralized, then the above events can be observed by means of dedicated interfaces to be "implemented" explicitly. Listing 3 gives an example of a centralized algorithm that records mobility traces into a given format. Technically, this program listens to the appearance and disappearance of nodes and links, and to the nodes movements. Upon these events, it records a string representation of the event using a human readable format (*e.g.,* DGS [101], Gephi [27], or Link stream [150]).

The reader is referred to many examples and tutorials available online on JBotSim's website (`http://jbotsim.io`), as well as a dozen of online lessons given in the context

Listing 2 A typical skeleton of distributed algorithm

```
public class BasicNode extends Node{
    ... // declare your variables
    @Override
    public void onStart() {
        // initialize your variables
    }
    @Override
    public void onClock() {
        // code to be executed in each round
    }
    @Override
    public void onMessage(Message message) {
        // process a received message
    }
    @Override
    public void onSelection() {
        // perform something special if this node is selected (made distinguished)
    }
}
```

of the "Algorithmique de la Mobilité" graduate class at the University of Bordeaux (`http://www.labri.fr/perso/acasteig/algomob/`). Details concerning the internals of the library (inner design and API) are available mainly through the *javadoc* and *GitHub* repository.

5.2 Versatility of use cases

JBotSim is quite versatile and the contexts in which it has been used are varied. Nodes may or may not possess wireless communication capabilities, sensing abilities, or self-mobility. They may also differ in clock frequency, communication range, icons, color, or any other user-defined property. The links between the nodes may be directed or undirected; they can be wired or wireless – in the latter case JBotSim's topology updates the links automatically according to various (possibly user-defined) policies. Finally, the algorithms may be distributed or centralized, as already discussed.

Figure 5.1 shows a number of screenshots illustrating this diversity of contexts. A dedicated channel on YouTube also shows a number of related videos. The examples include vehicular networks, robots and drones, wireless sensor networks, graph theory, as well as more complex and heterogeneous scenarios where several types of nodes are used (*e.g.,* park cleaning and fire-fighting aircrafts). Figures 5.1(l) and 5.1(o) also show use cases where JBotSim is embedded as a component into a different software – enabling this kind of features is the reason why JBotSim comes as a *library*. Finally, examples 5.1(f) and 5.1(k) exemplify the modularity of JBotSim, in the first case with the creation of a toroidal space where nodes at both extremities can communicate, and in the second case allowing the definition of obstacles which prevent communication links to be created (and nodes to move across obstacles).

In fact, nearly all components of JBotSim can be extended or replaced. This allows one to comply to a variety of theoretical model for distributed computing, through

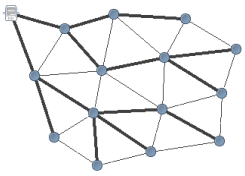
Listing 3 Example of a mobility trace recorder

```
public MyRecorder implements TopologyListener, ConnectivityListener, MovementListener{
   public MyRecorder(Topology tp){
      tp.addTopologyListener(this);
      tp.addConnectivityListener(this);
      tp.addMovementListener(this);
   }

   // TopologyListener
   public void nodeAdded(Node node) {
      println("an␣" + node + "␣x:" + node.getX() + "␣y:" + node.getY());
   }
   public void nodeRemoved(Node node) {
      println("dn␣" + node);
   }

   // ConnectivityListener
   public void linkAdded(Link link) {
      println("ae␣" + link + "␣" + link.source + "␣" + link.destination);
   }
   public void linkRemoved(Link link) {
      println("de␣" + link);
   }

   // MovementListener
   public void nodeMoved(Node node) {
      println("cn␣" + node + "␣x:" + node.getX() + "␣y:" + node.getY());
   }
}
```

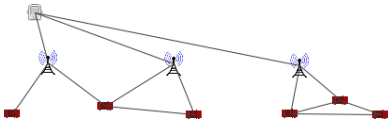
writing, for instance, different message engines (synchronous or asynchronous), link managers (toroidal, obstacle-based, partially random), internal scheduling (various level of fairness), or computational models (message-based, graph-based). The inner engines are independent from the GUI, making it possible to write alternative interfaces, *e.g.,* an android viewer, a java applet, or no GUI at all (from a shell script).

5.3 Community

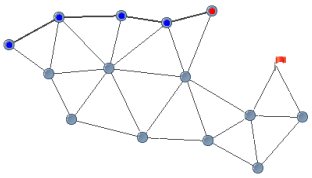
JBotSim's community of user is modest but growing. The official release (as a `JAR` file) was downloaded 150 times in 2015, 900 times in 2016, and 1100 times in 2017 (SourceForge statistics). These numbers do not include downloads directly through the source code on GitHub, for which we have no statistics. Its development was recently open to external contributors, with significant contributions from Quentin Bramas, Emmanuel Godard, and Vincent Klein.



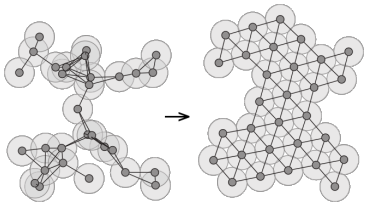
(a) Data aggregation



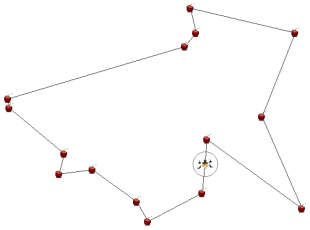
(b) Vehicular networks



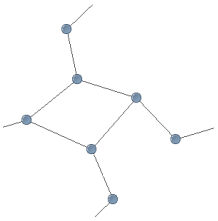
(c) Geograhical routing



(d) Deployment by virtual forces

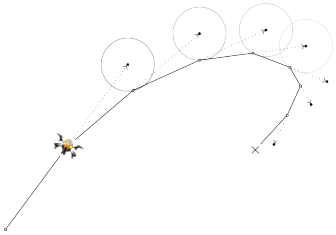


(e) Travelling Salesman Problem

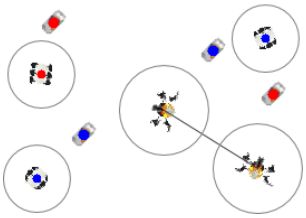


(f) Toroidal space

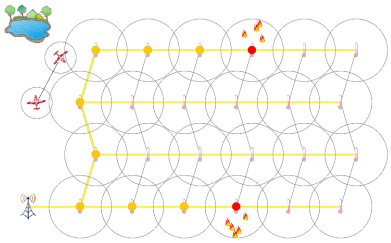
(g) Territory sharing based on Voronoi diagrams



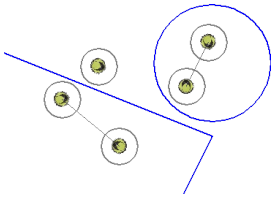
(h) Acceleration constraints



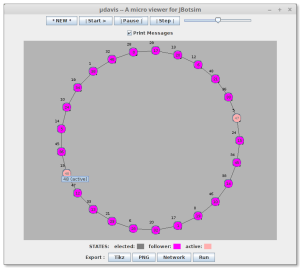
(i) Heterogeneous park cleaning



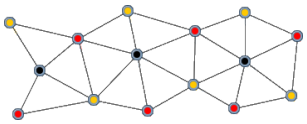
(j) Fire-fighting aircrafts



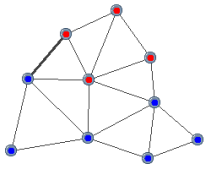
(k) Obstacle management



(l) DAVIS (embedding)



(m) Graph algorithms



(n) Population protocols

(o) Navigation (embedding)

Figure 5.1: Examples of uses of the JBotSim library.

JBotSim is or has been used for research and teaching in a growing number of places, including of course at Bordeaux by Cyril Gavoille (teaching), Alessia Milani (student projects), Serge Chaumette (research), Colette Johnen (research and teaching); in Marseille (Emmanuel Godard, see below, and Shantanu Das and Arnaud Labourel), at LIP6 in Paris (Sébastien Tixeuil, research), Strasbourg (Quentin Bramas, teaching), Ottawa (Giuseppe Di Luna and Paola Flocchini, teaching), and San Sebastian (Carlos Gomez-Calzado, Mikel Larrea, research). I personnally use it at all levels (teaching, projects, and research). In particular, a graduate class given at the University of Bordeaux, "Algorithmique de la Mobilité", relies entirely on JBotSim for the practical part (see `http://www.labri.fr/perso/acasteig/teaching/algomob/`).

A number of extensions are (or have been) developed by colleagues, including a remote viewer (Carlos Gómez Calzado), a collection of mobility models (Jason Schoeters), an obstacle module (Matthieu Barjon), an Android Viewer (Kinda Al Chahid), and a dynamic topology generator (group of students at ENSEIRB). These extensions meet a number of other existing extensions, including `Tikz` export for topologies; a collection of generic topologies (rings, complete graphs, grids), engines for recording and playing mobility traces, stochastic dynamic graph generators (based on Markovian evolving graphs), an asynchronous message engine, and other basic mobility models (some of which by Vincent Autefage).

Emmanuel Godard and his research group in Marseille are currently developing a set of tools based on JBotSim called Davis. These tools include a collection of computational models for distributed computing that *specialize* the API to force the developer to use only those primitives allowed by the model. Each model is subsequently mapped into JBotSim's primitives in a seamless way. Davis also includes a viewer with additional features for distributed algorithms (based on the `JTopology`, see above), such as menus to load algorithms, to load predefined network topologies, and control buttons for the execution.

5.3.1 Concluding remarks

I my view, JBotSim is a *kernel* that encapsulates a small number of generic features whose purpose is to be used with simplicity in other programs. In the first few years, I developed it as a hobby and with a form of obsessive pursuit of simplicity (which led to re-develop it from scratch several times). The API has now stabilized and JBotSim is ready to be released under version 1.0. My plan in the mid-term is to help others define extensions that suit their needs, but the kernel itself should not change much.

Chapter 6

Bit complexity and related models

Distributed algorithms are expressed with respect to various models and assumptions. In Part I, we considered assumptions pertaining to the network dynamics. When the network is static, other assumptions include the structure of the network (*e.g.,* is it a tree, a ring, a complete graph?), the knowledge available to the nodes (*e.g.,* network size, identifiers, bound on the diameter), and the communication model (*e.g.,* synchronous, asynchronous, limited message size). Here, we present two recent works with Métivier, Robson, and Zemmari, in which the communication model is synchronous, but otherwise drastically weakened. In [76], we consider constant size messages; that is, $O(1)$ bits can be exchanged in each round. In this context, we show that the bit round complexity of the election problem in arbitrary networks (with identifiers) is $\Theta(D + \log n)$, breaking a long-standing barrier on this well-known problem. In [75], the model is even weaker and the nodes can only beep or listen, without being able to identify which neighbor beeped when a beep is heard. We present a number of design patterns for such models; illustrate their use through several examples; and analyse their complexity (one of the resulting bound improving significantly upon the best known).

6.1 Bit round complexity of leader election

Main articles: DISC'16 [76] (long version [78]).

The election problem in a network consists of distinguishing a unique node, the leader, which can subsequently act as coordinator, initiator, and more generally perform distingued operations in the network (see [182] p. 262). Once a leader is established, many problems become simpler, which makes this problem one of the most studied in the distributed computing literature [98] (together with consensus).

A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process (or node) is in the *elected* state and all others are in the *non-elected* state. The vast body of literature on election (see [13, 156, 175, 184] and references therein) covers a number of different topics, which can be grouped according to three main directions: (1) The feasibility of deterministic election in *anonymous* networks, starting with the seminal paper of Angluin [11] and the key role of coverings (graph homomorphisms that prevent symmetry breaking, and thereby the uniqueness of a leader); (2) The complexity of deterministic election in *identified*

networks (*i.e.,* every processor has a unique identifier); and (3) The complexity of *probabilistic* election in anonymous or identified networks (identifiers play secondary roles here).

Our contribution is in the second category. Each node is identified by a unique integer of size $O(\log n)$. The nodes exchange messages with their neighbors in synchronous rounds. The exact complexity of deterministic leader election in this setting has proven elusive for decades and even simple questions remain open [148]. In particular, Peleg asks in [167] whether an algorithm could be both optimal in time and in number of messages, and Fusco and Pelc [116] suggest to use finer-grain complexity measures to look at the complexity of election.

We present an algorithm called $\mathcal{STT}$ whose bit round complexity breaks the $O(D \log n)$ barrier, using a new pipelining technique for spreading the identifiers. Our algorithm requires only $O(D + \log n)$ bit rounds and we show that this is optimal by combining a lower bound from [148] and a recent communication complexity result by Dinitz and Solomon [99]. This work is thus a step forward in understanding election, and a first (positive) answer to whether optimality can be achived both in time and in the *amount* of information exchanged. (As opposed to measuring time on the one hand, and the number of messages *of a given size* on the other hand.) Incidentally, our results also illustrates the benefits of studying optimality under the unified lenses of bit round complexity.

	Time	Number of messages	Message size	Bit round complexity
Awerbuch [14]	$O(n)$	$\Theta(m + n \log n)$	$O(\log n)$	$O(n \log n)$
Peleg [167]	$\Theta(D)$	$O(D\, m)$	$O(\log n)$	$O(D \log n)$
$\mathcal{STT}$ [76]	$O(D + \log n)$	$O((D + \log n)m)$	$O(1)$	$\Theta(D + \log n)$

Table 6.1: Complexities of best solutions according to various measures.

6.1.1 Main technical insight

The global architecture of our algorithm follows a (now) classical principle, similar to that used *e.g.,* by Gallager [117] or Peleg [167]. It consists of a competition of spanning tree constructions that works by extinction of those trees originating at nodes with lower identifiers (see also Algorithm 4 in [13] and discussion therein). Eventually, a single spanning tree survives, whose root is the node with highest identifier. This node becomes elected when it detects termination (recursively from the leaves up the root). Here the difficulty (and thus main contribution) arises from designing such algorithms with the extra constraint that only constant size messages can be used. Of course, one might simulate $O(\log n)$-size messages in the obvious way paying $O(\log n)$ bit rounds for each message. But then, the bit round complexity remains $O(D \log n)$. In contrast, by introducing new pipelining techniques whose applicability extends the scope of the sole election problem, we take the complexity down to $O(D + \log n)$. This technique relies, among others, on a trick for encoding the identifiers.

The encoding. Given an identifier Id, we define $\alpha(Id)$ as the unary representation of the binary length of Id, followed by 0, followed by the binary representation of Id, i.e.:

$$\alpha(Id) = base1(|base2(Id)|) \cdot 0 \cdot base2(Id).$$

Ex. $Id = 25 \stackrel{2}{=} 11001$, then $\alpha(Id) = 11111011001$

Remark 3. *Let u and v be two nodes with identifiers Id_u and Id_v. Then:*

$$Id_u < Id_v \Leftrightarrow \alpha(Id_u) \prec \alpha(Id_v).$$

The main feature of this encoding is that it extends the natural order between identifiers to their lexicographic order. That is, the definitive order between two identifiers Id_u and Id_v can be decided upon the first letter which differs in $\alpha(Id_u)$ and $\alpha(Id_v)$. This property is key to our algorithm, in which the spreading of identifiers progresses bitwise based on comparisons. Recently, our encoding has been reused by Beauquier *et al.* in [30].

6.2 Design patterns in beeping algorithms

Main articles: OPODIS'16 [75] (long version [77]).

Weak models of communication called *beeping models* have been studied recently [95, 179, 3, 129, 134, 120]. In these models, the nodes can only *beep* or *listen* to communicate. Several versions have been defined, depending on the ability for the nodes to detect collision while they beep or listen. In [77], we denote the ability to detect collision while beeping (internal collision) by B_{cd} and that of detecting collision while listening (peripheral collision) by L_{cd}. The absence of such ability is denoted by B and L, respectively. The cartesian product of both features captures all existing models: The basic model introduced in [95] is BL; the model considered by Afek et al. in [3] (Section 6) and Jeavons et al. in [134] is $B_{cd}L$; the model considered in [179] and in Section 4 of [3] is BL_{cd}. As of $B_{cd}L_{cd}$, we consider it in [74] along with all others, showing gradual complexity improvements as the detections features become enabled.

While some beeping models are stronger than others, all of them remain weak in essence. Beyond theory, these models account for natural real-world phenomena like the features of a network at the lowest level (physical or MAC layer), and communication patterns in biology [93, 3, 163].

6.2.1 Main contributions

We present four types of contributions in [75], each one summarized in a separate paragraph below. The first is to identify generic building blocks (*design patterns*) which often occur in beeping algorithms. Based on them, we present a number of algorithms for several graph problems which improve upon previous solutions. Then, we propose an emulation technique for using collision detection primitives even when they are not available (probabilistically). The complexity analyses we provide are integral part of

the contribution; in particular one of them improves the bound for the MIS algorithm in [134].

Design patterns. We identify a number of common building blocks in beeping algorithms, including *multi-slot phases*: the fact of dividing the main loop into a (typically constant) number of slots having specific roles (*e.g.,* contention among neighbors, collision detection, termination detection); *exclusive beeps*: the fact of having a single node beep at a time in a neighborhood (within one or two hops, depending on the needs); *adaptive probability*: increasing or decreasing the probability of beeping in order to favor exclusive beeps; *internal* (resp. *peripheral*) collision detection: the fact of detecting collision while beeping (resp. listening); and *emulation* of collision detection: the fact of detecting collisions even when it is not available as a primitive. Relying on these patterns makes it possible to formulate algorithms in a concise and elegant way.

Algorithms. We present algorithms for a number of basic graph problems, including coloring, 2-hop coloring, degree computation, maximal independent set (MIS) and 2-hop MIS. Quite often, the design of algorithms is easier and more natural if collision detection is assumed as a primitive, *e.g.,* in $B_{cd}L_{cd}$ or $B_{cd}L$. Furthermore, emulation techniques enable safe and automatic translations of algorithms into weaker models such as BL. Therefore, our algorithms are expressed using whatever model is the most convenient and the analyses are formulated in the same model. These algorithms are of type Las Vegas (*i.e.,* guaranteed result, uncertain time), as opposed to Monte Carlo (*i.e.,* guaranteed time, uncertain result). First, we present a coloring algorithm in the $B_{cd}L$ model which requires no knowledge about G but may result in a large number of colors. If the nodes know an upper bound K on the maximum degree in the network Δ, then a different strategy is proposed that uses only $K+1$ colors. A particular case is when Δ itself is known, resulting in $\Delta+1$ colors, which is a significant difference with the algorithm of Cornejo and Kuhn [95] that results in $O(\Delta)$ colors with similar time complexity (both are not really comparable, however, since our algorithm is Las Vegas in $B_{cd}L$ and their algorithm is Monte Carlo in BL. Then, we show how to extend these algorithms to 2-hop versions in the $B_{cd}L_{cd}$ model, using the fact that a 2-hop coloring in a graph G is equivalent to a coloring in the *square* of the graph G^2 (*i.e.,* G with extra edges for all paths of length two). Based on the observation that degree computation is strongly related to 2-hop coloring, we present a further adaptation of the algorithm to this problem. Finally, we turn our attention to the 2-hop MIS problem, which combines features from 2-hop coloring and from the MIS algorithm by Jeavons et al. [134].

Analyses. Design patterns make it clearer that the high-level behavior of some algorithms is actually general. For instance, the running time of our coloring algorithm is nothing but the first moment when *every* node has produced an exclusive beep. Characterizing this time is however complex due to the *adaptive probability* pattern. Our first analysis shows that such a process takes $O(\Delta+\log n)$ time to complete with high probability. The analysis relies on a martingale technique with non-independent random variables based on an old result by Azuma [17]. We prove a matching $\Omega(\Delta+\log n)$ lower bound that establishes that our algorithm and analysis are essentially optimal. We show that the upper bound analysis applies to 2-hop coloring and degree computa-

Problem	Article	Model	# slots (*w.h.p.*)	Knowledge	Comment
Coloring	this work	$B_{cd}L$	$\Theta(\Delta + \log n)$	-	$O(\Delta + \log n)$ colors
	Cornejo *et al.* [95]	BL	$O(\Delta \log n)$	$K = O(\Delta)$	MC & $O(\Delta)$ colors
	this work	$B_{cd}L$	$O(K \log n)$	$K \geq \Delta$	$K+1$ colors
2-hop coloring	this work	$B_{cd}L_{cd}$	$O(\Delta^2 + \log n)$	-	$O(\Delta^2 + \log n)$ colors
	this work	$B_{cd}L_{cd}$	$O(\Delta^2 \log n)$	$K = O(\Delta)$	K^2+1 colors
Degree comput.	this work	$B_{cd}L_{cd}$	$O(\Delta^2 + \log n)$	-	-
MIS & 2-hop MIS	Jeavons *et al.* [134]	$B_{cd}L$	$\leq 2\, e^{25} \log n$	-	$\Omega(\log n)$ slots [143]
	this work	$B_{cd}L$	$\leq 152 \log n$	-	
2-hop MIS	this work	$B_{cd}L$	$\leq 228 \log n$	-	-
Emulation of $B_{cd}L_{cd}$ (or $B_{cd}L$ or BL_{cd})	this work	BL	$\Theta(\log n)$ factor	$N = O(n^c)$	LV $\rightarrow$ MC

Table 6.2: Beeping algorithms on graphs with n nodes, where Δ denotes the maximum degree in the graph; LV stands for Las Vegas; and MC stands for Monte Carlo. Unless otherwise mentioned, all algorithms are LV.

tion almost directly, resulting in a $O(\Delta^2 + \log n)$ running time. Then, we present a new analysis of the MIS algorithm by Jeavons *et al.* [134]. In terms of patterns, the MIS algorithm also relies on *exclusive beeps* and *adaptive probability*, but it completes faster because an exclusive beep by one node causes its *whole* neighborhood to terminate. We prove that the complexity of this algorithm is less than $76 \log n$ phases of two slots each (thus $152 \log n$ slots) *w.h.p.*, which improves dramatically upon the analysis presented in [134], which results in an upper bound of e^{25} phases of two slots each. Although constant factors are not the main focus in general, the gap here is one between practical and impractical running times. Thus, our result confirms that the MIS algorithm by Jeavons et al. is practical. This analysis extends to the 2-hop MIS algorithm, resulting in $76 \log n$ phases of three slots each (thus $228 \log n$ slots). We also observe that the $\Omega(\log n)$ lower bound for coloring in the bit-size message passing model [143] applies to the MIS problem in beeping models, making the algorithm from [134] asymptotically optimal (up to a constant factor). Finally, we characterize the running time of the $(K+1)$-coloring algorithm with known $K \geq \Delta$. This analysis is slightly less general because the high-level structure of the algorithm is strongly dependent on this particular problem. We show that $O(K \log n)$ slots are sufficient *w.h.p.*, which indeed corresponds to the same $O(\Delta \log n)$ as in [95] if $K = O(\Delta)$. By a similar argument as above, this analysis extends to the 2-hop (K^2+1)-coloring in $O(K^2 \log n)$ time. All complexities are summarized in Table 6.2.

Collision detection and emulation techniques. Classical considerations on symmetry breaking in anonymous beeping networks, see for example [3] (Lemma 4.1), imply that there is no Las Vegas internal collision detection algorithm in the beeping models BL and BL_{cd}. Likewise, there is no Las Vegas peripheral collision detection algorithm in the beeping models BL and $B_{cd}L$. Since collision detection must detect exclusive beeps with certainty, and this pattern is central in most algorithms, thus a large range of algorithms cannot exist in a Las Vegas version in these models. Inspired by a technique used in Algorithm 3 of [3], we study the cost of detecting collision when it is not

available and we present generic procedures to transform Las Vegas algorithms with collision detection into Monte Carlo algorithms in BL. We show that any collision in the neighborhood of a *given* node can be detected in $O(\log(\epsilon^{-1}))$ slots with error at most ϵ, or in $O(\log n)$ slots *w.h.p.* Then, this is true for *all* nodes using $O(\log(\frac{n}{\epsilon}))$ slots with error ϵ or $O(\log n)$ slots *w.h.p.* We prove that this technique is asymptotically optimal (up to a constant factor) by a matching lower bound to break symmetry *w.h.p.* in some topologies.

Chapter 7

Wireless networks (first postdoc)

This chapter reviews a number of research topics in which I was involved during the 2008-2009 period (18 months) corresponding to my first postdoc at the University of Ottawa, under the supervision of Amiya Nayak and Ivan Stojmenović. The topics are covered in chronological order. They include vehicular networks, wireless sensor networks, and networks of mobile robots. The chapter also reviews a joint work with another group of researchers from Ottawa on bluetooth networks, the early discussions of which occurred in the end of that period and which fits well within the broad topic of wireless networks.

7.1 Vehicular networks

Main article: Wireless Communications and Mobile Computing'11 [81].

Vehicular networks are envisioned for large scale deployment. Standardization bodies, car manufacturers, and academic researchers are solving a variety of related challenges. In 2008, with Nayak and Stojmenović, we prepared a large-scale project proposal for ORF (Ontario Research Fund) on vehicular ad hoc networks. The project was eventually not granted, but we compiled the body of literature we had surveyed into an invited tutorial article [81].

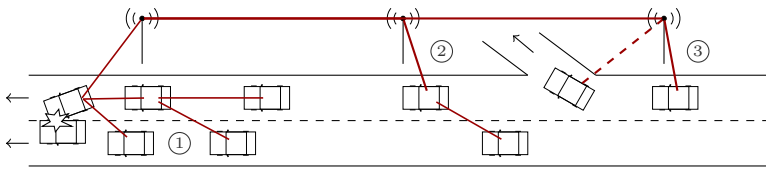


Figure 7.1: A typical collision scenario leading to various reactions: (1) *Fast forward of instant warnings* to arriving cars, (2) infrastructure-assisted *warning delivery* to incoming traffic, and (3) wider *notifications* for traffic re-routing (from [81]).

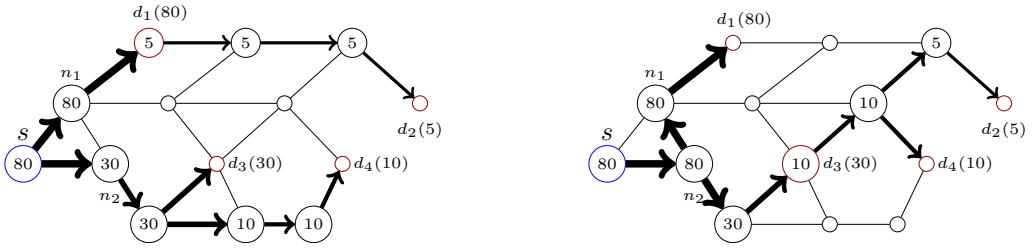
After a brief description of the societal context and of the main architectures envisioned for intelligent transportation systems, our tutorial presented four particular aspects of vehicular networks, which are (1) the potential for a large set of applications, including safety scenarios like the one depicted in Figure 7.1, (2) a review of the main

approaches to modeling both roads and traffic, and finally two important communication primitives, that are (3) data dissemination via broadcasting/geocasting, and (4) routing in highway and urban environments. A particular emphasis was set on recent (by then) protocols that realistically consider the inherently complex nature of vehicular mobility, such as intermittent connectivity, speed variability, and the impact of intersections.

7.2 Multiratecast in wireless sensor networks

Main article: CSA'09 [154].

Later in 2008, in a joint work with Liu, Goel, Nayak, and Stojmenović [154], we studied the multicast problem in wireless sensor networks, where the source can send data to a fixed number of destinations (actuators) at a different rate (which we called *multiratecast*). A typical motivation for this problem is to enable fault tolerant monitoring where data is collected at more than one actuators using different rates, which decrease with the distance, so that if the closest actuator fails, others can take over. To this end, we introduced a new metric for the cost of rate-aware multicast paths. Indeed, the usual *hop count* and *retransmission number* metrics do not reflect real efficiency in this context. This is illustrated on Figure 7.2, where two different paths are proposed to serve a given set of destinations, each one having its own rate requirement. Here the 'longer' path in terms of the two classical metrics (Path A, 11 hops, 9 retransmissions) is actually less expensive than the 'shorter' one (Path B, 9 hops, 7 retransmissions) in terms of number of messages to be effectively sent; *i.e.,* the *cumulative rate* of retransmissions.



(a) Path A, total rate cost 255 (b) Path B, total rate cost 295

Figure 7.2: Impact of the rate on a multicast path cost. Four destinations (d_1, d_2, d_3, and d_4) are considered, each one having a different required rate (between parenthesis). Numbers in circles indicate the effective retransmission rate.

In this context, we propose two multiratecast routing protocols: Maximum Rate Multicast (MRM) and Optimal Rate Cost Multicast (ORCM), which are the first localized position-based protocols specifically designed for this problem. The first, MRM, selects the next forwarding neighbor(s) in order to favor destinations requiring the highest rates, while the second, ORCM, evaluates several possible choices and selects the best one according to a *cost over progress* ratio criterion. Both are compared by simulation using the above metric, showing essentially that ORCM performs better if the number of destinations is small, and MRM if it is large. MRM also behaves better as the variance

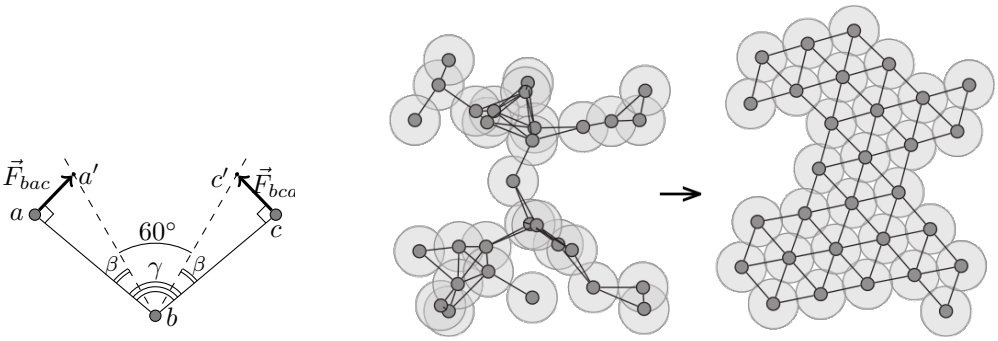
among the rates becomes important. We refer the reader to the original article for details.

7.3 Biconnecting mobile robots

Main articles: VTC-Fall'10 [49] (long version in Computer Communications [50]).

In 2009, Nayak and Stojmenović offered me to work on a project involving mobile robots. The objective was to deploy the robots from an arbitrary (but connected) initial configuration in such a way that their collective coverage is maximized and their network topology is biconnected (*i.e.,* it remains connected if any robot crashes). Over the same period, a PhD student from Bordeaux, Jérémie Albert, visited me for one month and took part on the project. This resulted in an article [50], in which we propose a new solution to the problem of self-deploying a network of wireless mobile robots with consideration to several criteria, namely fault-tolerance (*biconnectivity*), *coverage*, *diameter*, and *quantity of movement* required by the deployment. Various subsets of these criteria had been addressed in previous works, we proposed an elegant solution to address all of them at once.

The approach in [50] is based on combining two complementary sets of virtual forces: *spring* forces, whose properties are well known to provide optimal coverage at reasonable movement cost, and *angular* forces, causing two *angularly consecutive* neighbors of a node to get closer to each other when the corresponding angle is larger than 60° (see Figure 7.3). Angular forces have the global effect of biconnecting the network and reducing its diameter at the same time, while not affecting the benefits obtained by spring forces on coverage. We gave a detailed description of both types of forces, whose



(a) Angular forces (b) Initial and final configurations

Figure 7.3: Combining angular with spring forces biconnects the network (from [50]).

combination posed a number of technical challenges. We also provided a message-based adaptation of these forces relying on position exchanges within two hops.

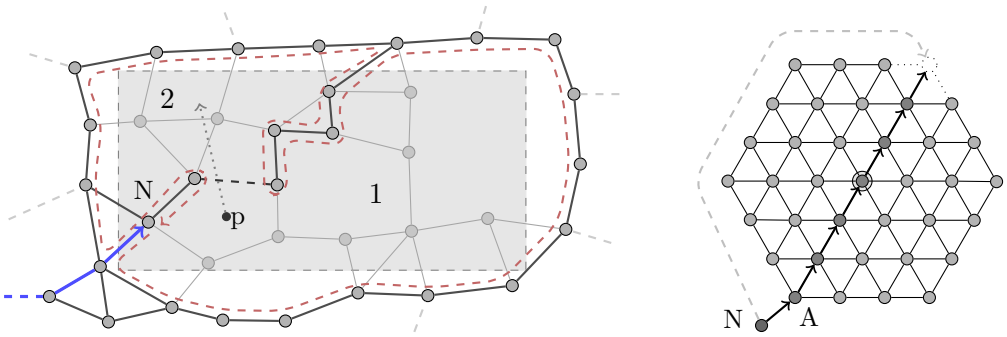
Extensive simulations were performed using the JBotSim library [48] to evaluate the solution against all criteria (coverage, biconnectivity, quantity of movements, and diameter) and compare it to prior solutions. We refer the reader to the original article for details and to JBotSim's YouTube channel for a video of the algorithm.

7.4 Sensors and actuators networks (book chapters)

Main articles: Chapters 5 [79] and 7 [80] of "Wireless sensor and actuator networks: algorithms and protocols for scalable coordination and data communication" (Nayak and Stojmenović Eds., Wiley, 2010).

During 2009, my supervisors Nayak and Stojmenović were working towards editing a book on wireless sensor and actuator networks. In this context, they offered me to write two chapters based on a given corpus of articles. My work consisted in studying the corpus, synthesizing the articles and then organizing (and writing) the resulting content into chapters [79] and [80].

The first chapter [79] is entitled "Multicasting, Geocasting and Anycasting in Sensor and Actuator Networks". In this chapter, we review the scenarios where a given message is sent from a single source to possibly several destinations. These scenarios can be subdivided into multicasting, geocasting, multiratecasting, and anycasting. In *multicasting*, a given message must be routed from one node to a number of destinations whose locations may be arbitrary and spread over the network. *Geocasting* destinations are all nodes located in a given geographical area (see Figure 7.4(a)). *Multiratecasting* is a generalization of multicasting, where regular messages are sent from a source to several destinations, possibly at a different rate for each destination (reviewed above in Section 7.2). Finally, in *anycasting* scenarios, a source must send a message to any node among a given set of destinations, preferably only one. Each of these scenarios corresponds to a typical use case in sensor and actuator networks.



(a) Geocasting in sensor networks (from [79]) (b) Robot relocation (from [80])

Figure 7.4: Two examples of scenarios covered in the chapter (one for each). Both pictures were generated using JBotSim `Tikz` export with semi-automated decoration routines.

The second chapter [80] is entitled "Topology Control in Sensor, Actuator and Mobile Robot Networks". In such networks, the efficiency often depends on characteristics of the underlying connectivity, and the length and density of links. It is therefore common practice to control such parameters, *i.e.,* to *control* the topology. Topology control can be achieved by modifying the transmission radii, selecting a given subset of the links, or moving some nodes (if such functionality is available). This chapter reviews some of these problems, and related solutions applicable to the context of sensor and actuator networks. Spanning structures and minimum weight connectivity are first discussed, and some applications for power efficient and delay bounded data aggregation are described.

The issue of detecting critical nodes and links to build a biconnected topology is also investigated, with the aim of providing fault-tolerance to the applications, and some recent and prospective works considering biconnectivity of mobile sensors/actuators (reviewed above in Section 7.3). Related deployment of sensors, augmentation, area and point coverage are also discussed (see Figure 7.4(b) for an example of robot relocation scenario).

7.5 Bluetooth Scatternet Formation

Main articles: ISCC'13 [135] (long version in Wireless Networks'14 [136]).

In the beginning of 2010, I started to work with a local PhD student (Ahmed Jedda) who had done his masters under the supervision of Guy-Vincent Jourdan and had started a PhD under Hussein Mouftah. Ahmed was working on Bluetooth protocols, by then a popular research topic. While not a regular activity, I often discussed with him and eventually helped him write his results on the Bluetooth Scatternet Formation (BSF) problem, resulting in a joint article with Jedda, Jourdan, and Mouftah [136].

The BSF problem consists of interconnecting piconets (local networks organized around a single master node) in order to form a multi-hop topology. While a large number of algorithms had been proposed, only few addressed the construction time as a key parameter. Also few were tested under realistic circumstances; in particular, the baseband and link layers of Bluetooth are highly specific and known to have crucial impacts on performance. In [136], we revisited performance studies for a number of time-efficient BSF algorithms, among which BlueStars, BlueMesh, and BlueMIS II. We also presented a new time-efficient BSF algorithm called *BSF-UED* (for BSF based on Unnecessary Edges Deletion), which forms connected scatternets deterministically and limits the outdegree of nodes to 7 heuristically.

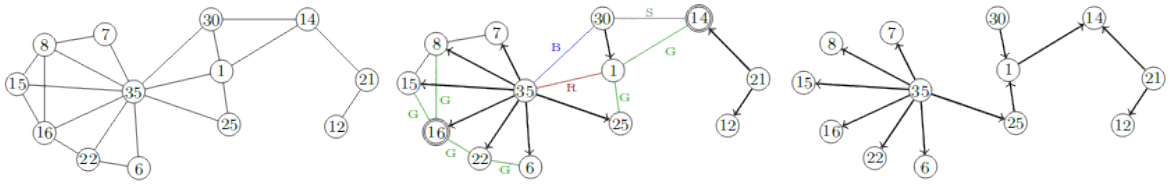


Figure 7.5: Some intermediate steps in the computation of the scatternet (from [136]).

The performance of the algorithm was evaluated through detailed simulations that take into account the low-level specificities of Bluetooth. We show that BSF-UED provides an excellent balance between the various metrics used. In particular, it compares favorably against BlueMesh while requiring only 1/3 of its execution time. Only BlueStars is faster than BSF-UED, at the cost of a very large number of slaves per master (much more than seven), which is impractical in many scenarios.

Chapter 8

Social sciences, human sciences, data privacy, security

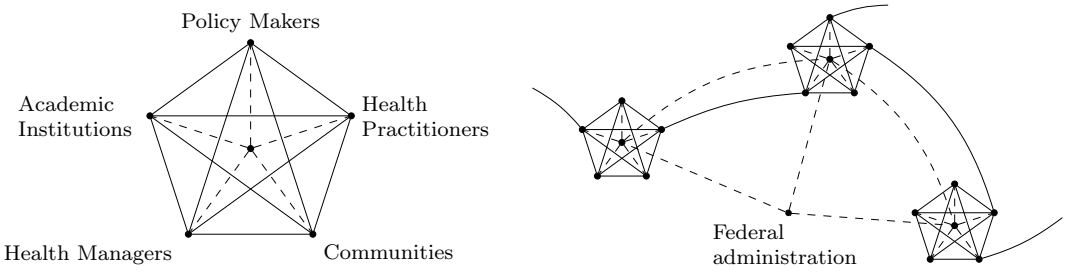
The works reviewed here corresponds to several short-term research missions (of typically 2 to 4 months) realized intermittently over the period 2010-2012. The initial motivation was one of financial necessity, due to gaps in the core funding of my second postdoc (whose topics are covered in Part I of this document). However, these experiences turned out to be very inspiring and intellectually rewarding; it opened us to new topics and new professional universes, making a great respiration within other activities. The first mission in 2010 was not so far from the comfort zone. It consisted of designing a social studies inquiry together with a sociologist, Louise Bouchard, injecting temporal graph concepts into the study of Canadian health networks, leading to joint report [51]. The second mission, later in 2010, was commissionned by a physician from the Montfort hospital research institute, Marie-Hélène Chomienne. It involved privacy issues for health data, whose legal setting in Canada is quite specific, for which we presented a solution based on recent cryptographic techniques from the area of differential privacy, leading to joint publications with Chomienne, Bouchard, and Jourdan [54, 55, 34]. In the summer of 2011, together with a cryptographer from the mathematics department of the University of Ottawa, in the context of a partnership with the University, we participated in a research mission in a security-leading company called *Irdeto* in Kanata. This mission led me to learn the basics of white box cryptography and eventually to develop a prototype [47].

8.1 Study of health networks in Canada

Main article: Technical report [51].

In 2001, the "Comité Consultatif des Communautés Francophones en Situation Minoritaire (CCCFSM)" delivered a report to Canada's health agency describing the conditions under which one million of French-speaking Canadadian in minority situation (mainly in Ontario) accessed health care. It was shown that about 55 % of them had virtually no access to health services in French, which is not consistent with Canada's constitution. Among the actions resulting from this report, health Canada created a $10M fund to develop networking initiatives among the existing players at the provin-

cial scale (with a national office). The retained model was the one pushed forward by the World Health Organization called *Towards unity for health* [36], aiming to foster equitable community-oriented health services education, research, and policy, through bringing together innovative health care organizations, universities, educational institutions, and individuals committed to contribute to the improvement of health in their communities (see Figure 8.1(a)). The envisioned approach was to develop at least one such network in each province, together with a national coordinating office, leading to a global network like the one in Figure 8.1(b).

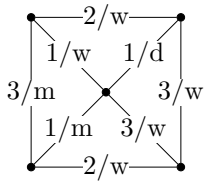


(a) Provincial networks

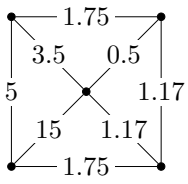
(b) Interconnection of provincial networks

Figure 8.1: Envisioned organization of health networks by Canada's health agency (pictures from [51]).

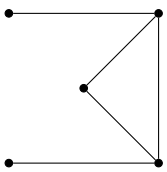
Together with a sociologist, Louise Bouchard, we developed in [51] (2010) a proposal of survey study in order to map these networks in reality ten years after the beginning of the initiative, and assess their quality based on a number of questions. Some of the questions where standard network analysis questions; more specifically, we suggested to apply new temporal concepts from dynamic networks and data mining literature. One of the questions of interest was to identify network backbones (concept from [141], see also Section 4.2), which we adapted in a *stochastic* version based on average interaction rates among participants, as illustrated on Figure 8.2. Other concepts related to



(a) Average rate of contacts (over months, weeks, or days)



(b) Average temporal distance



(c) Resulting backbone

Figure 8.2: Backbone of a dynamic network based on average rate of contacts (from [51]).

temporal *centrality* were also considered. (The project was not retained and we started collaborating on another topic, reviewed next.)

8.2 Differential Privacy for Health Data in Canada

Main articles: Technical report [54], DPM'12 [55], Global Health Promotion'14 [34]

Research in population health studies the impact of various factors (*determinants*) on health, with the long-term objective of yielding better policies, programs, and services. Among these determinants, researchers of Official Language Minority Communities (OLMCs) in Canada focus on that of speaking a minority (and yet official) language such as English in Quebec, or French in Ontario (our focus). Investigations of this type require the *linkage* of two types of data: health data and linguistic data. Unfortunately, in Canada, health data is managed at the provincial level (ICES agency, in Ontario), whereas linguistic data of quality is only available at the federal level (Statistics Canada agency, or SC). Permanent linkage of both is not feasible from a legal point of view. It can be done punctually for a given study but is impractical for use on a regular basis, which is what OLMC researchers need.

Together with Chomienne, Bouchard, and Jourdan, we presented a technical solution based on recent advances in *differential privacy* which makes it possible to link the data on a transient and automated basis, with statistical guarantees as to the amount of information leaked in the process. We give a simplified account here, referring the reader to [55] for details. In particular, language variables at SC are multi-dimensional and thus more complex than $\{French, English\}$.

Count queries. Given a sample of individuals s, and a property p, a *count query* $q(s, p)$ consists of counting the number of individuals in s that satisfy p, as illustrated on Figure 8.3. Count queries are basic tools in the area of databases. Used properly,

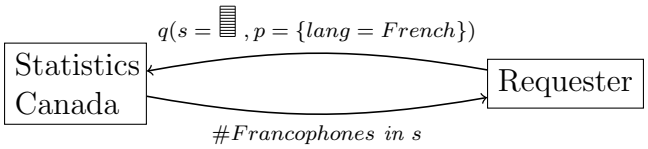


Figure 8.3: A basic count query.

they would suffice for OLMC researchers to study possible correlations. Consider for instance the following question: *In Ontario, what is the average rate of angioplasty among Francophones? (vs. Anglophones)* To obtain the answer, first generate (at ICES) a representative sample of individuals having undergone angioplasty (how this sample can itself be protected is discussed in [55]), then request SC for a count of Francophones within this set. By normalizing the answer over the more general ratio of Francophones in Ontario (which is known), one answers the question. Unfortunately, a malicious use of count queries makes it possible to leak information. For instance, if two queries differ only by one element, then the information concerning this element is leaked. (More sophisticated but similar constructs can be done which are harder to detect.) Even though the language is not as sensitive as other types of information could be (*e.g.,* health or income), we were explained that SC's policy in regard of residual information is absolutely strict.

Our proposal. We suggested to (1) add noise to the answers and (2) modify the global workflow of the queries. Both are illustrated on Figure 8.4. The proposed workflow (right) consists of a *tripartite* interaction between researchers, ICES, and SC, such that

1. SC does not know what health criteria the sample is associated with
2. Health researchers do not know what sample is generated
3. ICES does not know the result of the language query

(a) Adding noise to the answer (b) Making the workflow tripartite

Figure 8.4: Two aspects of our solution.

The most technical part was the one concerned the addition of noise to the answers. We found recent techniques from the field of differential privacy (*e.g.,* [102]) which matched our needs exactly. Differential privacy is concerned with understanding the impact of adding noise in privacy data analysis. Thanks to these techniques, we were able to characterize exactly the interplay between 1) the magnitude of the noise (Laplacian), 2) the level of statistical leakage tolerated by SC, and 3) the number of queries that can be performed. This tradeoff is illustrated by the volume of Figure 8.5.

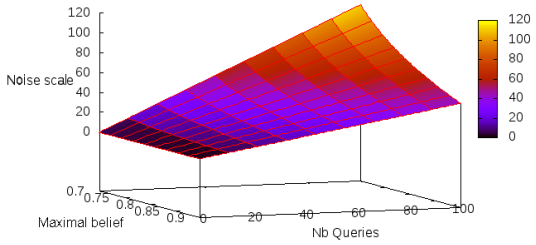


Figure 8.5: Interplay between noise scale, queries number, and maximal belief (from [55]).

While the theoretical feasibility of this scheme was admitted by the concerned agencies, its deployment was judged too difficult (at least in the short term). However, ICES amplified its efforts in collecting linguistic information at the provincial level, and my OLMC colleagues were then advised to wait and see if this data suited their needs before considering linkage with federal data.

8.3 White-Box Elliptic Curve Diffie-Hellman

Main articles: Technical report [47].

In 2011, together with another postdoctoral fellow, we participated in a four-months research project between Irdeto Canada and the University of Ottawa, under the SME4SME applied research program to foster works between companies and researchers at UO. Irdeto is the world leader in digital platform security, best known for their digital right management technologies and white box encryption solutions. The contact was from Guy-Vincent Jourdan, a professor at UO, with whom I had already collaborated (see previous section).

Contrary to classical security, in which an attacker typically stands *between* the entities of a system, intercepting traffic and possibly pretending to be one of the entities (Man-in-the-Middle attack), white box attacks refer to the case that the attacker has access to the computing resource, and in particular, can read the host memory while a program executes. If this program makes use of cryptographic primitives, such as key agreement primitives, then the private keys become exposed. The aim of our project was to design and implement a prototype of white box encryption for Elliptic Curve Diffie-Hellman (ECDH); in particular, making it possible to perform the three basic operations of ECDH key agreement – *Key Pair Generation*, *Shared Secret Computation*, and *Key Derivation* – without exposing the private keys.

The project timeline was split into two periods of two months. The first one involved the other postdoctoral fellow, Mizanur Rahman, who was then doing research in cryptography in the mathematics department at UO. Mizanur's mission was to design the cryptographic scheme at the math level. Then, I stepped in for another period of two months, with the aim to understand his scheme and to integrate it within Irdeto's tools and libraries. This required diving into the security area, in particular cryptography, by then totally new to me. The key step in the scheme was to break up critical computations into a sequence of operations, some of which involve random numbers generated at *compile time* in an obfuscated way, together with operations which are themselves obfuscated in a consistent way with the numbers. The reader is referred to my technical report [47], which presents the background and main features of the mathematical scheme (as pedagogically as possible), and then describe its integration within Irdeto's tool suit.

At the end of this mission, both Mizanur Raman and I were offered a position at Irdeto (which Mizanur accepted). I declined, mainly because one of our research proposal with Paola Flocchini and Nicola Santoro had been granted meanwhile, securing one year of uninterrupted research on my main topics of interest, by then an unprecedented opportunity!

Bibliography

[1] Eric Aaron, Danny Krizanc, and Elliot Meyerson. Dmvp: foremost waypoint coverage of time-varying graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 29–41. Springer, 2014.

[2] Sheila Abbas, Mohamed Mosbah, and Akka Zemmari. Distributed computation of a spanning tree in a dynamic graph by mobile agents. In *Proc. of IEEE Int. Conference on Engineering of Intelligent Systems (ICEIS)*, pages 1–6, 2006.

[3] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, 2013.

[4] Eleni C Akrida, Jurek Czyzowicz, Leszek Gasieniec, Lukasz Kuszner, and Paul G Spirakis. Temporal flows in temporal networks. In *International Conference on Algorithms and Complexity*, pages 43–54. Springer, 2017.

[5] Eleni C Akrida, Leszek Gasieniec, George B Mertzios, and Paul G Spirakis. On temporally connected graphs of small cost. In *International Workshop on Approximation and Online Algorithms (WAOA)*, pages 84–96. Springer, 2015.

[6] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *CoRR*, abs/1802.07103, 2018.

[7] David Aldous and Jim Fill. Reversible markov chains and random walks on graphs, 2002.

[8] Karine Altisen, Pierre Corbineau, and Stéphane Devismes. A framework for certified self-stabilization. *Logical Methods in Computer Science*, 13(4), 2017.

[9] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[10] Frédéric Amblard, Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. On the Temporal Analysis of Scientific Network Evolution. In *International Conference on Computational Aspects of Social Networks (CASoN)*, pages 169–174, 2011.

[11] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on Theory of Computing*, pages 82–93, 1980.

[12] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[13] H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics.* John Wiley & Sons, 2004.

[14] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In *Proc. of 19th Symp. on Theory of Computing, 1987, New York, USA*, pages 230–240, 1987.

[15] B. Awerbuch and S. Even. Efficient and reliable broadcast is achievable in an eventually connected network. In *Proceedings of the 3rd ACM symposium on Principles of distributed computing (PODC)*, pages 278–281, Vancouver, Canada, 1984. ACM.

[16] Baruch Awerbuch, Israel Cidon, and Shay Kutten. Optimal maintenance of a spanning tree. *J. ACM*, 55(4):18:1–18:45, September 2008.

[17] K. Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.

[18] Hichem Baala, Olivier Flauzac, Jaafar Gaber, Marc Bui, and Tarek El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *Journal of Parallel and Distributed Computing*, 63:97–104, 2003.

[19] Thibaut Balabonski, Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Certified gathering of oblivious mobile robots: Survey of recent results and open problems. In *joint 22nd Int. Workshop on Formal Methods for Industrial Critical Systems - and - 17th Int. Workshop on Automated Verification of Critical Systems (FMICS-AVoCS)*, pages 165–181, 2017.

[20] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. *arXiv preprint arXiv:1802.10199*, 2018.

[21] Judit Bar-Ilan and Dror Zernik. Random leaders and random spanning trees. In *Workshop on Distributed Algorithms (WDAG)*, volume 392 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 1989.

[22] M. Barjon, A. Casteigts, S. Chaumette, C. Johnen, and Y.M. Neggaz. Testing temporal connectivity in sparse dynamic graphs. In *2nd AETOS Int. conference on Research challenges for future RPAS/UAV systems*, 2014.

[23] Matthieu Barjon. *Autour des groupes tolérants aux délais dans les flottes mobiles communicantes. (On Delay-Tolerant Groups in Communicating Mobile Fleets).* PhD thesis, University of Bordeaux, France, 2016.

[24] Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Y. Neggaz. Maintaining a spanning forest in highly dynamic networks: The synchronous case. In *18th Int. Conference on Principles of Distributed Systems (OPODIS)*, volume 8878 of *LNCS*, pages 277–292, 2014.

[25] Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Y. Neggaz. Maintaining a distributed spanning forest in highly dynamic networks. *The Computer Journal*, 2018. (In Press.).

[26] Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Yessin M. Neggaz. Un algorithme de test pour la connexité temporelle dans les graphes dynamiques de faible densité. In *16e Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (ALGOTEL)*, 2014.

[27] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *Proc. 3rd International AAAI Conference on Weblogs and Social Media*, San Jose, 2009. AAAI Press.

[28] Michel Bauderon and Mohamed Mosbah. A unified framework for designing, implementing and visualizing distributed algorithms. *Electronic Notes in Theoretical Computer Science*, 72(3):13–24, 2003.

[29] H. Baumann, P. Crescenzi, and P. Fraigniaud. Parsimonious flooding in dynamic graphs. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 260–269, Calgary, Canada, 2009. ACM.

[30] Joffroy Beauquier, Janna Burman, Fabien Dufoulon, and Shay Kutten. Fast beeping protocols for deterministic mis and (Δ+1)-coloring in sparse graphs. In *IEEE INFOCOM 2018*, 2018.

[31] Michael A Bekos, Till Bruckdorfer, Henry Förster, Michael Kaufmann, Simon Poschenrieder, and Thomas Stüber. Algorithms and insights for racetrack. In *Proc. of 8th Int. Conf. on FUN with algorithms, LIPIcs-Leibniz*, volume 49. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[32] K.A. Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks*, 28(3):125–134, 1996.

[33] Thibault Bernard, Alain Bui, and Devan Sohier. Universal adaptive self-stabilizing traversal scheme: Random walk and reloading wave. *J. Parallel Distrib. Comput.*, 73(2):137–149, 2013.

[34] Jean-Marie Berthelot, Louise Bouchard, Arnaud Casteigts, Mariette Chartier, Alain Trugeon, and Jan Warnke. Les systèmes d'information sociosanitaire à l'appui de la planification locale de la santé: défis et enjeux. *Global Health Promotion*, 21(1):15,22, 2014.

[35] S. Bhadra and A. Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Proceedings of the 2nd International Conference on Ad Hoc, Mobile and Wireless Networks (AdHoc-Now)*, pages 259–270, Montreal, Canada, 2003. Springer.

[36] C. Boelen. Towards unity for health. *Case studies. Geneva: World Health Organization*, 2001.

[37] Marjorie Bournat, Swan Dubois, and Franck Petit. Computability of perpetual exploration in highly dynamic rings. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 794–804. IEEE, 2017.

[38] Quentin Bramas, Toshimitsu Masuzawa, and Sébastien Tixeuil. Distributed Online Data Aggregation in Dynamic Graphs. Research report, Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris. ; Osaka University, Japan, January 2016.

[39] Quentin Bramas and Sébastien Tixeuil. The complexity of data aggregation in static and dynamic wireless sensor networks. *Information and Computation*, 2016.

[40] Nicolas Braud-Santoni, Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. The next 700 impossibility results in time-varying graphs. *International Journal of Networking and Computing*, 6(1):27–41, 2016.

[41] Nicolas Broutin and Jean-François Marckert. A new encoding of coalescent processes: applications to the additive and multiplicative cases. *Probability Theory and Related Fields*, 166(1-2):515–552, 2016.

[42] B. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, April 2003.

[43] Janna Burman and Shay Kutten. Time optimal asynchronous self-stabilizing spanning tree. In Andrzej Pelc, editor, *Distributed Computing*, volume 4731 of *Lecture Notes in Computer Science*, pages 92–107. Springer Berlin Heidelberg, 2007.

[44] Rajmonda Sulo Caceres and Tanya Berger-Wolf. Temporal scale of dynamic networks. In *Temporal Networks*, pages 65–94. Springer, 2013.

[45] Arnaud Casteigts. Model driven capabilities of the DA-GRS model. In *Proc. of 1st Intl. Conference on Autonomic and Autonomous Systems (ICAS)*, pages 24–32. IEEE, 2006.

[46] Arnaud Casteigts. *Contribution à l'Algorithmique Distribuée dans les Réseaux Mobiles Ad Hoc - Calculs Locaux et Réétiquetages de Graphes Dynamiques*. PhD thesis, University of Bordeaux, September 2007.

[47] Arnaud Casteigts. White-box elliptic curve diffie-hellman. Technical report, Irdeto Canada, 2011.

[48] Arnaud Casteigts. JBotSim: a tool for fast prototyping of distributed algorithms in dynamic networks. In *Proceedings of the 8th international ICST conference on simulation tools and techniques (SIMUTools)*, 2015.

[49] Arnaud Casteigts, Jérémie Albert, Serge Chaumette, Amiya Nayak, and Ivan Stojmenovic. Biconnecting a network of mobile robots using virtual angular forces. In *Proc. 72nd IEEE Vehicular Technology Conference (VTC'Fall)*, 2010.

[50] Arnaud Casteigts, Jérémie Albert, Serge Chaumette, Amiya Nayak, and Ivan Stojmenovic. Biconnecting a network of mobile robots using virtual angular forces. *Computer Communications*, 35(9):1038–1046, 2012.

[51] Arnaud Casteigts and Louise Bouchard. Intégration de la dimension temporelle dans l'analyse des réseaux de santé en français. Technical report, Institut of Population Health of Ottawa, 2010.

[52] Arnaud Casteigts, Serge Chaumette, and Afonso Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *Proc. of 16th Intl. Conference on Structural Information and Communication Complexity (SIROCCO)*, volume 5869 of *LNCS*, pages 126–140. Springer, 2009.

[53] Arnaud Casteigts, Serge Chaumette, Frédéric Guinand, and Yoann Pigné. Distributed maintenance of anytime available spanning trees in dynamic networks. In *Proc. of 12th conf. on Adhoc, Mobile, and Wireless Networks (ADHOC-NOW)*, LNCS, pages 99–110, 2013.

[54] Arnaud Casteigts, Marie-Hélène Chomienne, Louise Bouchard, and Guy-Vincent Jourdan. Enabling dynamic linkage of linguistic census data at statistics canada. Technical report, Research Institute of Montfort Hospital, 2011.

[55] Arnaud Casteigts, Marie-Hélène Chomienne, Louise Bouchard, and Guy-Vincent Jourdan. Differential privacy in tripartite interaction: A case study with linguistic minorities in canada. In *Proc. of 7th International Workshop on Data Privacy Management (DPM)*, volume 7731 of *LNCS*, pages 75–88. Springer, 2012.

[56] Arnaud Casteigts, Swan Dubois, Franck Petit, and John Michael Robson. Robustness in highly dynamic networks. *CoRR*, abs/1703.03190, 2017.

[57] Arnaud Casteigts and Paola Flocchini. Deterministic algorithms in dynamic networks: Formal models and metrics. Technical report, Commissioned by Defense Research and Development Canada (DRDC), 2013.

[58] Arnaud Casteigts and Paola Flocchini. Deterministic algorithms in dynamic networks: Problems, analysis, and algorithmic tools. Technical report, Commissioned by Defense Research and Development Canada (DRDC), 2013.

[59] Arnaud Casteigts, Paola Flocchini, E. Godard, Nicola Santoro, and Masafumi Yamashita. Brief announcement: Waiting in dynamic networks. In *Proceedings of the 31th ACM Symposium on Principles of Distributed Computing (PODC)*, 2012.

[60] Arnaud Casteigts, Paola Flocchini, E. Godard, Nicola Santoro, and Masafumi Yamashita. On the expressivity of time-varying graphs. *Theoretical Computer Science*, 590:27–37, 2015.

[61] Arnaud Casteigts, Paola Flocchini, Emmanuel Godard, Nicola Santoro, and Masafumi Yamashita. Expressivity of time-varying graphs. In *19th International Symposium on Fundamentals of Computation Theory (FCT)*, 2013.

[62] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility. In *6th IFIP International Conference on Theoretical Computer Science (TCS)*, pages 111–124, 2010.

[63] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Measuring temporal lags in delay-tolerant networks. In *25th IEEE Intl. Parallel & Distributed Processing Symposium (IPDPS)*, pages 209–218, 2011.

[64] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Building fastest broadcast trees in periodically-varying graphs. *CoRR*, abs/1204.3058, 2012.

[65] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Measuring temporal lags in delay-tolerant networks. *IEEE Transactions on Computers*, 63(2), 2014.

[66] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Shortest, fastest, and foremost broadcast in dynamic networks. *Int. Journal of Foundations of Computer Science*, 26(4):499–522, 2015.

[67] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In *10th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW)*, pages 346–359, 2011.

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

[69] Arnaud Casteigts, R. Klasing, Yessin Neggaz, and J. Peters. Efficiently testing T-Interval connectivity in dynamic graphs. In *9th Int. Conference on Algorithms and Complexity (CIAC)*, volume 9079 of *LNCS*, pages 89–100, 2015.

[70] Arnaud Casteigts, Ralf Klasing, Yessin Neggaz, and Joseph Peters. A generic framework for computing parameters of sequence-based dynamic graphs. In *Proc. of 24th Intl. Conference on Structural Information and Communication Complexity (SIROCCO)*, 2017.

[71] Arnaud Casteigts, Ralf Klasing, Yessin Neggaz, and Joseph Peters. Computing parameters of sequence-based dynamic graphs. *Theory of Computing Systems*, 2018. (In Press.).

[72] Arnaud Casteigts, Ralf Klasing, Yessin Neggaz, Joseph Peters, and David Renault. Private communication, 2016.

[73] Arnaud Casteigts, Bernard Mans, and Luke Mathieson. On the feasibility of maintenance algorithms in dynamic graphs. *CoRR*, abs/1107.2722, 2011.

[74] Arnaud Casteigts, Yves Métivier, John Michael Robson, and Akka Zemmari. Counting in one-hop beeping networks. *CoRR*, abs/1605.09516, 2016.

[75] Arnaud Casteigts, Yves Métivier, John Michael Robson, and Akka Zemmari. Design patterns in beeping algorithms (extended abstract). In *20th Int. Conference on Principles of Distributed Systems (OPODIS)*, volume xx of *LIPIcs*, 2016.

[76] Arnaud Casteigts, Yves Métivier, John Michael Robson, and Akka Zemmari. Deterministic leader election in $O(D + log n)$ time with messages of size $O(1)$. In *30th Int. Symposium on Distributed Computing (DISC)*, volume 9888 of *LNCS*. Springer, 2016.

[77] Arnaud Casteigts, Yves Métivier, John Michael Robson, and Akka Zemmari. Design patterns in beeping algorithms: Examples, emulation, and analysis. *CoRR*, abs/1607.02951v2, 2017.

[78] Arnaud Casteigts, Yves Métivier, John Michael Robson, and Akka Zemmari. Deterministic leader election takes $\Theta(D + \log n)$ bit rounds. *CoRR*, abs/1605.01903, 2017.

[79] Arnaud Casteigts, Amiya Nayak, and Ivan Stojmenovic. *Multicasting, Geocasting and Anycasting in Sensor and Actuator Networks*, chapter 5 of Wireless Sensor and Actuator Networks - Algorithms and Protocols for Scalable Coordination and Data Communication, Nayak, A. and Stojmenovic, I. (Eds.). Wiley, 2010.

[80] Arnaud Casteigts, Amiya Nayak, and Ivan Stojmenovic. *Topology Control in Sensor, Actuator and Mobile Robot Networks*, chapter 7 of Wireless Sensor and Actuator Networks - Algorithms and Protocols for Scalable Coordination and Data Communication, Nayak, A. and Stojmenovic, I. (Eds.). Wiley, 2010.

[81] Arnaud Casteigts, Amiya Nayak, and Ivan Stojmenovic. Communication protocols for vehicular ad hoc networks. *Wireless Communication and Mobile Computing*, 11(5):567–582, 2011.

[82] P. Castéran, V. Filou, and M. Mosbah. Certifying distributed algorithms by embedding local computation systems in the coq proof assistant. In *Proc. of Symbolic Computation in Software Science (SCSS'09)*, 2009.

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

[84] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot. The diameter of opportunistic mobile networks. *Communications Surveys & Tutorials*, 10(3):74–88, 2008.

[85] Jérémie Chalopin. *Algorithmique distribuée, calculs locaux et homomorphismes de graphes*. PhD thesis, Bordeaux 1, 2006.

[86] Jérémie Chalopin and Daniël Paulusma. Graph labelings derived from models in distributed computing: A complete complexity classification. *Networks*, 58(3):207–231, 2011.

[87] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 528–539. Springer, 2015.

[88] Bernadette Charron-Bost and Shlomo Moran. The firing squad problem revisited. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 96. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[89] Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[90] Xujin Chen, Xiaodong Hu, and Jianming Zhu. Minimum data aggregation time problem in wireless sensor networks. In *International conference on mobile ad-hoc and sensor networks*, pages 133–142. Springer, 2005.

[91] A. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proceedings of the 27th ACM Symposium on Principles of distributed computing (PODC)*, pages 213–222, Toronto, Canada, 2008. ACM.

[92] A. Clementi and F. Pasquale. Information spreading in dynamic networks: An analytical approach. In S. Nikoletseas and J.D.P. Rolim, editors, *Theoretical Aspects of Distributed Computing in Sensor Networks*, chapter 19, pages 591–619. Springer, 2011.

[93] J. Collier, N. Monk, P. Maini, and J. Lewis. Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling. *Journal of Theoretical Biology*, 183(4):429–446, 1996.

[94] Colin Cooper, Robert Elsässer, Hirotaka Ono, and Tomasz Radzik. Coalescing random walks and voting on graphs. In *Proceedings of the 31st ACM symposium on Principles of distributed computing (PODC)*, pages 47–56. ACM, 2012.

[95] A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. In *DISC*, pages 148–162, 2010.

[96] Étienne Coulouma, Emmanuel Godard, and Joseph G. Peters. A characterization of oblivious message adversaries for which consensus is solvable. *Theor. Comput. Sci.*, 584:80–90, 2015.

[97] Bilel Derbel. A brief introduction to visidia, 2007.

[98] Y. Dinitz, S. Moran, and S. Rajsbaum. Bit complexity of breaking and achieving symmetry in chains and rings. *Journal of the ACM*, 55(1), 2008.

[99] Y. Dinitz and N. Solomon. Two absolute bounds for distributed bit complexity. *Theor. Comput. Sci.*, 384(2-3):168–183, 2007.

[100] Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. Enabling minimal dominating set in highly dynamic distributed systems. In *17th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 51–66, 2015.

[101] A. Dutot, F. Guinand, D. Olivier, and Y. Pigné. Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. *EPNACS: Emergent Properties in Natural and Artificial Complex Systems*, 2007.

[102] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. *Proc. of 3rd Theory of Cryptography Conference (TCC)*, pages 265–284, 2006.

[103] David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert E. Tarjan, Jeffery R. Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. In *Proc. 1st Symp. Discrete Algorithms*, pages 1–11. ACM and SIAM, January 1990.

[104] Faten Fakhfakh, Mohamed Tounsi, Ahmed Hadj Kacem, and Mohamed Mosbah. Towards a formal model for dynamic networks through refinement and evolving graphs. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2015*, pages 227–243. Springer, 2016.

[105] Faten Fakhfakh, Mohamed Tounsi, Mohamed Mosbah, Ahmed Hadj Kacem, and Dominique Mery. A formal approach for maintaining forest topologies in dynamic networks. In *16th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2017)*, 2017.

[106] Antonio Fernández Anta, Alessia Milani, Miguel A. Mosteiro, and Shmuel Zaks. Opportunistic information dissemination in mobile ad-hoc networks: the profit of global synchrony. *Distributed Computing*, 25(4):279–296, 2012.

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

[108] Afonso Ferreira. On models and algorithms for dynamic communication networks: The case for evolving graphs. In *In Proc. ALGOTEL*, 2002.

[109] Afonso Ferreira and Laurent Viennot. A note on models, algorithms, and data structures for dynamic communication networks. Technical Report 4403, INRIA, 2002.

[110] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50(1):158–184, 2012.

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

[112] P. Flocchini, B. Mans, and N. Santoro. Exploration of periodically varying graphs. In *Proceedings of 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

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

[114] Till Fluschnik, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *arXiv preprint arXiv:1803.00882*, 2018.

[115] Greg N Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985.

[116] E. G. Fusco and A. Pelc. Knowledge, level of symmetry, and time of leader election. *Distributed Computing*, 28(4):221–232, 2015.

[117] R. G. Gallager. Finding a leader in a network with $o(e + n \log n)$ messages. *Technical Report Internal Memo., M.I.T., Cambridge,MA*, 1979.

[118] Noé Gaumont, Clémence Magnien, and Matthieu Latapy. Finding remarkably dense sequences of contacts in link streams. *Social Network Analysis and Mining*, 6(1):87, 2016.

[119] Noé Gaumont, Tiphaine Viard, Raphaël Fournier-S'Niehotta, Qinna Wang, and Matthieu Latapy. Analysis of the temporal and structural features of threads in a mailing-list. In *Complex Networks VII*, pages 107–118. Springer, 2016.

[120] S. Gilbert and C. Newport. The computational power of beeps. In *Proc. of 29th International Symposium on Distributed Computing (DISC)*, 2015.

[121] Emmanuel Godard. *Distributed Computability in Communication Networks*. Habilitation à diriger des recherches en mathématiques et en informatique, Université Aix-Marseille, 2015.

[122] Emmanuel Godard and Dorian Mazauric. Computing the dynamic diameter of non-deterministic dynamic networks is hard. In *Algorithms for Sensor Systems - 10th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2014, Wroclaw, Poland, September 12, 2014, Revised Selected Papers*, pages 88–102, 2014.

[123] Carlos Gómez-Calzado, Arnaud Casteigts, Mikel Larrea, and Alberto Lafuente. A connectivity model for agreement in dynamic systems. In *21st Int. Conference on Parallel Processing (EUROPAR)*, volume 9233 of *LNCS*, pages 333–345, 2015.

[124] F. Greve, L. Arantes, and P. Sens. What model and what conditions to implement unreliable failure detectors in dynamic networks? In *Proceedings of the 3rd International Workshop on Theoretical Aspects of Dynamic Distributed Systems*, pages 13–17, Rome, Italy, 2011. Springer.

[125] F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7):79–88, 1997.

[126] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[127] P. Holme. Network reachability of real-world contact sequences. *Physical Review E*, 71(4):46119, 2005.

[128] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[129] B. Huang and Th. Moscibroda. Conflict resolution and membership problem in beeping channels. In *DISC*, pages 314–328, 2013.

[130] D. Ilcinkas and A. Wade. On the power of waiting when exploring public transportation systems. *Proceedings of the 15th International Conference on Principles of Distributed Systems (OPODIS)*, pages 451–464, 2011.

[131] Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 119–131. ACM, 1990.

[132] P. Jacquet, B. Mans, and G. Rodolakis. Information propagation speed in mobile and delay tolerant networks. *IEEE Transactions on Information Theory*, 56(1):5001–5015, 2010.

[133] Aubin Jarry. *Connexité dans les réseaux de télécommunications. (Connectivity in telecommunication networks)*. PhD thesis, University of Nice Sophia Antipolis, France, 2005.

[134] P. Jeavons, A. Scott, and L. Xu. Feedback from nature: simple randomised distributed algorithms for maximal independent set selection and greedy colouring. *Distributed Computing*, DOI 10.1007/s00446-016-0269-8, 2016.

[135] Ahmed Jedda, Arnaud Casteigts, Guy-Vincent Jourdan, and Hussein Mouftah. Bsf-ued: A novel time-efficient bluetooth scatternet formation algorithm based on unnecessary edges deletion. In *18th IEEE Symposium on Computers and Communications (ISCC)*, 2013.

[136] Ahmed Jedda, Arnaud Casteigts, Guy-Vincent Jourdan, and Hussein Mouftah. Bluetooth scatternet formation from a time-efficiency perspective. *Wireless Networks*, 20(5):1133–1156, 2014.

[137] M. Kellett. *Black hole search in the network and subway models*. PhD thesis, University of Ottawa, 2012.

[138] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of 32nd ACM Symposium on Theory of Computing*, pages 504–513, Portland, USA, 2000. ACM.

[139] A. Keränen and J. Ott. DTN over aerial carriers. In *Proceedings of 4th ACM Workshop on Challenged Networks*, pages 67–76, Beijing, China, 2009. ACM.

[140] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The one simulator for dtn protocol evaluation. In *Proceedings of the 2nd international conference on simulation tools and techniques*, page 55. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[141] G. Kossinets, J. Kleinberg, and D. Watts. The structure of information pathways in a social communication network. In *Proceedings of 14th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 435–443, Las Vegas, USA, 2008. ACM.

[142] V. Kostakos. Temporal graphs. *Physica A*, 388(6):1007–1023, 2009.

[143] K. Kothapalli, M. Onus, C. Scheideler, and C. Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. In *20th Int. Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece*. IEEE, 2006.

[144] Alex Kravchik and Shay Kutten. Time optimal synchronous self stabilizing spanning tree. In Yehuda Afek, editor, *Distributed Computing*, volume 8205 of *Lecture Notes in Computer Science*, pages 91–105. Springer Berlin Heidelberg, 2013.

[145] Gautier Krings, Márton Karsai, Sebastian Bernhardsson, Vincent D Blondel, and Jari Saramäki. Effects of time window size and placement on the structure of an aggregated communication network. *EPJ Data Science*, 1(1):4, 2012.

[146] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM symposium on Theory of computing (STOC)*, pages 513–522. ACM, 2010.

[147] Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *ACM SIGACT News*, 42(1):82–96, 2011.

[148] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, 2015.

[149] Rémi Laplace. *Applications et services DTN pour flotte collaborative de drones*. PhD thesis, Université de Bordeaux, 2012.

[150] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *CoRR*, abs/1710.04073, 2017.

[151] Yannick Léo, Christophe Crespelle, and Eric Fleury. Non-Altering Time Scales for Aggregation of Dynamic Networks into Series of Graphs. In *11th International Conference on emerging Networking EXperiments and Technologies – CoNEXT 2015*, 11th International Conference on emerging Networking EXperiments and Technologies – CoNEXT 2015, Heidelberg, Germany, 2015.

[152] I. Litovsky, Y. Métivier, and E. Sopena. *Graph Relabelling Systems and Distributed Algorithms. H. Ehrig, H.J. Kreowski, U. Montanari and G. Rozenberg (Eds.),* Handbook of Graph Grammars and Computing by Graph Transformation, pages 1–53, 1999.

[153] C. Liu and J. Wu. Scalable routing in cyclic mobile networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(9):1325–1338, 2009.

[154] X. Liu, Arnaud Casteigts, Nishith Goel, Amiya Nayak, and Ivan Stojmenovic. Multiratecast in wireless fault-tolerant sensor and actuator networks. In *Proc. of 2nd IEEE Intl. Conference on Computer Science and its Applications (CSA)*, 2009.

[155] Giuseppe Antonio Di Luna, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Live exploration of dynamic rings. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*, pages 570–579, 2016.

[156] N. A. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.

[157] Subhrangsu Mandal and Arobinda Gupta. Approximation algorithms for permanent dominating set problem on dynamic networks. In *International Conference on Distributed Computing and Internet Technology*, pages 265–279. Springer, 2018.

[158] Bernard Mans and Luke Mathieson. On the treewidth of dynamic graphs. *Theoretical Computer Science*, 554:217–228, 2014.

[159] F. Marchand de Kerchove and F. Guinand. Strengthening topological conditions for relabeling algorithms in evolving graphs. Technical report, Université du Havre, 2012.

[160] Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

[161] Othon Michail and Paul G Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, 2018.

[162] M. Naor and L. Stockmeyer. What can be computed locally? In *Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 184–193. ACM, 1993.

[163] S. Navlakha and Z. Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, 2015.

[164] Yessin M. Neggaz. *Automatic Classification of Dynamic Graphs*. PhD thesis, University of Bordeaux, 2016.

[165] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 104–110, Cologne, Germany, 2005. ACM.

[166] Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

[167] D. Peleg. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.*, 8(1):96–99, 1990.

[168] Yoann Pigné, Arnaud Casteigts, Frédéric Guinand, and Serge Chaumette. Construction et maintien d'une forêt couvrante dans un réseau dynamique. In *12e Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (ALGOTEL)*, 2010.

[169] Yvonne Anne Pignolet, Matthieu Roy, Stefan Schmid, and Gilles Tredan. The many faces of graph dynamics. *Journal of Statistical Mechanics: Theory and Experiment*, 2017(6):063401, 2017.

[170] R. Ramanathan, P. Basu, and R. Krishnan. Towards a formalism for routing in challenged networks. In *Proceedings of 2nd ACM Workshop on Challenged Networks (CHANTS)*, pages 3–10, 2007.

[171] Michel Raynal. Message adversaries. *Encyclopedia of Algorithms*, pages 1–6, 2014.

[172] Michel Raynal, Julien Stainer, Jiannong Cao, and Weigang Wu. A simple broadcast algorithm for recurrent dynamic systems. In *28th IEEE International Conference on Advanced Information Networking and Applications, AINA 2014, Victoria, BC, Canada, May 13-16, 2014*, pages 933–939, 2014.

[173] John H Reif. A topological approach to dynamic graph connectivity. *Information Processing Letters*, 25(1):65–70, 1987.

[174] G. Rossetti and R. Cazabet. Community Discovery in Dynamic Networks: a Survey. *ArXiv e-prints*, July 2017.

[175] Nicola Santoro. *Design and analysis of distributed algorithm*. Wiley, 2007.

[176] Nicola Santoro. Time to change: On distributed computing in dynamic networks (keynote). In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 46. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[177] Nicola Santoro, Walter Quattrociocchi, Paola Flocchini, Arnaud Casteigts, and Frédéric Amblard. Time-varying graphs and social network analysis: Temporal indicators and metrics. In *3rd AISB Social Networks and Multiagent Systems Symposium (SNAMAS)*, pages 32–38, 2011.

[178] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 304–313. Springer, 1989.

[179] J. Schneider and R. Wattenhofer. What is the use of collision detection (in wireless networks)? In *DISC*, pages 133–147, 2010.

[180] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. Crawdad dataset cambridge/haggle (v. 2009-05-29). *CRAWDAD wireless network data archive*, 2009.

[181] Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

[182] A. Tanenbaum and M. van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.

[183] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora. Small-world behavior in time-varying graphs. *Phys. Rev. E*, 81(5), 2010.

[184] Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.

[185] Richard J Trudeau. *Introduction to graph theory*. Courier Corporation, 2013.

[186] A. Vargas. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multi-conference (ESM)*, pages 319–324, 2001.

[187] Tiphaine Viard, Clémence Magnien, and Matthieu Latapy. Enumerating maximal cliques in link streams with durations. *arXiv preprint arXiv:1712.06970*, 2017.

[188] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *nature*, 393(6684):440–442, 1998.

[189] John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In *Proc. of MOBICOM*, pages 377–388. ACM, 2012.

[190] Masafumi Yamashita and Tiko Kameda. Computing on anonymous networks: Part I and II. *IEEE Trans. on Par. and Distributed Systems*, 7(1):69 – 96, 1996.

[191] Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The computational complexity of finding separators in temporal graphs. *arXiv preprint arXiv:1711.00963*, 2017.