



Defence Research and
Development Canada

Recherche et développement
pour la défense Canada



Deterministic algorithms in dynamic networks

Problems, analysis, and algorithmic tools

Arnaud Casteigts and Paola Flocchini

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

Defence R&D Canada – Ottawa

Contract Report
DRDC Ottawa CR 2013-021
April 2013

Canada

Deterministic algorithms in dynamic networks

Problems, analysis, and algorithmic tools

Arnaud Casteigts
Paola Flocchini
University of Ottawa

Prepared by:

University of Ottawa
School of Electrical Engineering and Computer Science
800 King Edward Avenue
Ottawa, Ontario
K1N 6N5

Contract Number: W7714-115111/001/SV
Contract Scientific Authority: Matthew Kellett 613-991-4362

The scientific or technical validity of this Contract Report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

Defence R&D Canada – Ottawa

Contract Report
DRDC Ottawa CR 2013-021
April 2013

Approved by

Original signed by Jean-François Rivest

Jean-François Rivest
Cyber Operations and Signals Warfare Section

Approved for release by

Original signed by Chris McMillan

Chris McMillan
Chief Scientist, DRDC Ottawa

© Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 2013

© Sa Majesté la Reine (en droit du Canada), telle que représentée par le ministre de la Défense nationale, 2013

Abstract

The number of telecommunication networks deployed in a dynamic environment is quickly growing. This is true in our everyday life (e.g., smartphones, vehicles, or satellites) as well as in the military context (e.g., dismounted soldiers or swarms of UAVs). Unfortunately, few theoretical tools enable, to date, the study of dynamic networks in a formal and rigorous way. As a result, it is hard and sometimes impossible to guarantee, mathematically, that a given algorithm will reach its objectives once deployed in real conditions. Having such guarantees would seem to be crucial in a military context. In a previous report we identified a collection of recent theoretical tools whose purpose is to model, describe, and leverage dynamic networks in a formal way. This report focuses on problems, algorithms, and analysis techniques. We review recent efforts towards the design and analysis of distributed algorithms in dynamic networks, with an emphasis on those results that are of a deterministic and analytical nature. The topics include a discussion on how mobility impacts the very definition of problems; a set of generic tools to be used in the design or analysis of dynamic network algorithms; a discussion on the impact of various types of dynamics on the feasibility and complexity of given problems; a classification of dynamic networks based on dynamic graph properties; and finally, a discussion on how real-world mobility contexts relate to some of these classes.

Résumé

Le nombre de réseaux de télécommunications déployés dans un environnement dynamique augmente rapidement. Cette réalité s'applique à la vie quotidienne (ordiphones [téléphones intelligents], véhicules, satellites) et au contexte militaire (soldats à pied, essais d'UAV). Malheureusement, peu d'outils théoriques permettent, jusqu'à maintenant, d'étudier les réseaux dynamiques de manière formelle et rigoureuse. Par conséquent, il est difficile et parfois impossible de garantir, mathématiquement, qu'un algorithme donné atteindra ses objectifs une fois déployé dans des conditions réelles. Avoir ces garanties semble être essentiel dans un contexte militaire. Dans un rapport précédent, on a identifié un éventail d'outils théoriques récents dont le but est de modéliser et décrire des réseaux dynamiques et de les mettre à profit, le tout de manière officielle. Ce rapport est axé sur les problèmes, les algorithmes et les techniques d'analyse. On examine les efforts récents en matière de conception et d'analyse d'algorithmes distribués dans des réseaux dynamiques, en insistant sur les résultats de natures déterministe et analytique. Parmi les sujets abordés, on trouve l'incidence de la mobilité sur la définition des problèmes, un ensemble d'outils génériques qui seront utilisés pendant la conception ou l'analyse des algorithmes de réseau dynamique; un examen de l'incidence des différents types de dynamique sur la faisabilité et la complexité de problèmes données; une classification des réseaux dynamiques en fonction des propriétés du graphique dynamique et un examen des liens à établir entre les contextes de mobilité «réels» et certaines de ces classes.

This page intentionally left blank.

Executive summary

Deterministic algorithms in dynamic networks: Problems, analysis, and algorithmic tools

Arnaud Casteigts, Paola Flocchini; DRDC Ottawa CR 2013-021; Defence R&D Canada – Ottawa; April 2013.

Background: Telecommunication networks are evolving continuously and the number of those networks deployed in a dynamic environment is quickly growing. This is true in our everyday life (e.g., smartphones, vehicles, or satellites) as well as in military contexts (e.g., dismounted soldiers or swarms of UAVs). Unfortunately, few theoretical tools enable, to date, the study of dynamic networks in a formal and rigorous way. The current trend is instead to rely on simulations, which necessarily are not fully representative of real world conditions, in order to assess the behaviour of a particular solution in a given type of network.

Results: In a previous report (Casteigts and Flocchini 2013, CR 2013-020), we identified a collection of recent theoretical tools whose purpose is to model, describe, and leverage dynamic networks in a formal way. This report focuses on problems, algorithms, and analysis techniques, using the former report as a basis. We review recent efforts toward the design and analysis of distributed algorithms in dynamic networks with an emphasis on those results that are of a deterministic and analytical nature. The topics include a discussion on how mobility impacts the very definition of problems; a set of generic tools to be used in the design and analysis of dynamic network algorithms; a discussion of the impact of various types of dynamics on the feasibility and complexity of given problems; a classification of dynamic networks based on dynamic graph properties; and finally, a discussion on how real-world mobility contexts relate to some of these classes.

Significance: Having these kinds of algorithmic tools is instrumental in studying the behaviour of algorithms in dynamic networks. These tools are all the more important when the objective is to show, mathematically, that an algorithm will or will not solve every possible instance of a given problem in a given environment. In a military context, where the failure of an algorithm could have serious consequences, guaranteed performance can be very appealing.

Future plans: This reports completes a line of work whose main object was an inventory of recent models and algorithmic tools for distributed computing in dynamic networks. Research in this domain is clearly in its infancy, and we hope this work will be useful to guide future efforts. In particular, it lays some foundations for exploring further what can be done, or not, in dynamic networks in terms of collaborative tasks and self-organization.

Sommaire

Deterministic algorithms in dynamic networks: Problems, analysis, and algorithmic tools

Arnaud Casteigts, Paola Flocchini ; DRDC Ottawa CR 2013-021 ; R & D pour la défense Canada – Ottawa ; avril 2013.

Renseignements généraux : Les réseaux de télécommunication sont en constante évolution et le nombre de réseaux déployés dans un environnement dynamique augmente rapidement. Cette réalité s'applique à la vie quotidienne (ordiphones [téléphones intelligents], véhicules, satellites) et au contexte militaire (soldats à pied, essais d'UAV). Malheureusement, peu d'outils théoriques permettent, jusqu'à maintenant, d'étudier les réseaux dynamiques de manière officielle et rigoureuse. La tendance actuelle consiste à se fier à des simulations qui, nécessairement, ne représentent pas l'ensemble des conditions «réels» afin d'évaluer le comportement d'une solution particulière dans un type de réseau donné.

Résultats : Dans un rapport précédent (Casteigts and Flocchini 2013, CR 2013-020), on a identifié un éventail d'outils théoriques récents dont le but est de modéliser et de décrire des réseaux dynamiques et de les mettre à profit, le tout de manière officielle. Le présent rapport est axé sur les problèmes, les algorithmes et les analyses techniques, et le rapport précédent lui sert de fondement. On examine les efforts récents en matière de conception et d'analyse d'algorithmes distribués dans des réseaux dynamiques, en insistant sur les résultats de natures déterministe et analytique. Parmi les sujets abordés, on trouve l'incidence de la mobilité sur la définition des problèmes, un ensemble d'outils génériques qui seront utilisés pendant la conception et l'analyse des algorithmes de réseau dynamique ; un examen de l'incidence des différents types de dynamique sur la faisabilité et la complexité de problèmes donnés ; une classification des réseaux dynamiques en fonction des propriétés du graphique dynamique et un examen des liens à établir entre les contextes de mobilité «réels» et certaines de ces classes.

Importance : Posséder ces types d'outils algorithmiques est essentiel à l'étude du comportement des algorithmes dans les réseaux dynamiques. Ces outils sont d'autant plus importants que l'objectif consiste à montrer, mathématiquement, qu'un algorithme résout ou non toutes les occurrences possibles d'un problème dans un environnement donné. Dans un contexte militaire, où la défaillance d'un algorithme peut avoir de graves conséquences, la performance garantie peut être très intéressante.

Plans futurs : Ce rapport mène à bien une série de travaux dont l'objectif principal consistait à dresser un inventaire des modèles récents et des outils algorithmiques d'informatique distribuée dans les réseaux dynamiques. La recherche dans ce domaine en est à ces débuts, et on souhaite que ce travail soit utile pour orienter les travaux à venir. Elle permet notamment de jeter les bases pour explorer plus avant ce qui peut ou non être fait dans les réseaux dynamiques en matière de tâches collaboratives et d'autoorganisation.

Table of contents

Abstract	i
Résumé	i
Executive summary	iii
Sommaire	iv
Table of contents	v
List of figures	viii
List of tables	ix
List of algorithms	ix
A note on references	xi
Acknowledgements	xi
1 Introduction	1
2 Background	3
3 Nature of the problems	5
3.1 New metrics for existing problems	6
3.2 New environments for existing problems	8
3.2.1 Exploration in static graphs	9
3.2.2 Exploration in periodically-varying graphs	9
3.2.3 Exploration in subways	11
3.2.4 Exploration in networks with edge deletions	13
3.3 Definitions of new problems	13
3.3.1 Election and spanning trees	14
3.3.2 Covering problems	17

4	Algorithmic tools and techniques	22
4.1	Temporal-lags vector clocks (T-Clocks)	22
4.1.1	Measuring temporal lags	22
4.1.2	T-Clocks as a network abstraction	24
4.1.3	Example: foremost broadcast trees	24
4.1.4	Example: fastest broadcast trees	26
4.2	Raising the level of abstraction	28
4.2.1	Graph Relabelling Systems	28
4.2.2	Population protocols	29
4.2.3	Relevance in a dynamic environment	30
4.3	Using invariants	30
4.4	Using progression hypotheses	30
4.5	Topological events as a trigger for computation	31
5	Topological conditions	34
5.1	Example of graph conditions in static networks	34
5.2	Characterizing graph conditions in dynamic networks	35
5.2.1	Combination of graph relabellings and evolving graphs	36
5.2.2	Necessary and sufficient conditions	37
5.2.3	Example analyses	38
5.2.3.1	Propagation algorithm	38
5.2.3.2	Centralized counting algorithm	39
5.2.3.3	Decentralized counting algorithm	39
5.2.4	Tightness of conditions	41
5.2.5	A note on maintenance algorithms	42

6	Computational relationship between classes of dynamic graphs	43
6.1	Summary of the results	44
6.2	Basic results and limitations	46
6.3	TDB[<i>foremost</i>]	47
6.3.1	TDB[<i>foremost</i>] in \mathcal{R}	48
6.3.2	TDB[<i>foremost</i>] in \mathcal{B}	49
6.4	TDB[<i>shortest</i>]	51
6.5	Computational relationship	53
7	Classification of networks and algorithms	55
7.1	From conditions to classes of dynamic graphs	55
7.2	Relations between classes	56
7.3	Comparison of algorithms based on their requirements	57
8	Classes of dynamic graphs vs. mobility contexts	59
9	Conclusion	60
	References	63
	List of acronyms/abbreviations	71

List of figures

Figure 1:	Summary of key concepts	3
Figure 2:	Example of a dynamic graph.	4
Figure 3:	Example of six classical distributed problems (part 1).	5
Figure 4:	Example of six classical distributed problems (part 2).	6
Figure 5:	Example of tree-based broadcast from a source s	7
Figure 6:	Different meanings for length and distance based on journeys.	7
Figure 7:	Example of foremost broadcast trees.	8
Figure 8:	Example of cautious walk on a line.	12
Figure 9:	A typical MANET scenario.	14
Figure 10:	Two versions of the election problem.	15
Figure 11:	A spanning tree on top of a random UDG.	16
Figure 12:	Example of a spanning forest over a MANET.	16
Figure 13:	Why some problems cannot accept a “permanent” variation.	18
Figure 14:	Temporal view that c has of a	23
Figure 15:	T-CLOCKS as an abstraction to track temporal views.	24
Figure 16:	A simple TVG where fastest broadcast does not match fastest journeys .	26
Figure 17:	Temporal distance from a to c , as a function of emission date.	27
Figure 18:	Aggregation of distances	28
Figure 19:	Spanning tree construction over a static graph	29
Figure 20:	Example spanning forest algorithm execution sequence	33
Figure 21:	Basic conditions for election related to graph coverings.	34
Figure 22:	A basic scenario, where a node (b) moves during the execution.	35
Figure 23:	Combination of graph relabellings and evolving graphs.	37

Figure 24: Logical implications of tight conditions.	42
Figure 25: Node waiting for affiliation messages.	51
Figure 26: Propagation of rounds of duration Δ	53
Figure 27: A first hierarchy of dynamic networks.	56
Figure 28: An additional hierarchy of dynamic networks.	57
Figure 29: Relations between mobility contexts and classes of dynamic graphs.	59

List of tables

Table 1: Set of children relative to emitter a	25
Table 2: Set of parents relative to emitter a	25
Table 3: Feasibility and reusability of TDB.	45
Table 4: Complexity of TDB.	45

List of algorithms

1 Spanning forest algorithm.	32
--------------------------------------	----

This page intentionally left blank.

A note on references

This report uses the “author-year” style of citation. As a result, when a paper involves three or more authors, only the first one appears in the citation, e.g., First *et. al.* We invite the reader to keep in mind that the usage in theoretical computer science is to order the authors of a paper alphabetically, and as such, the first author is not necessarily the “main” contributor of the cited paper. The complete list of authors of a paper can be found in the References section.

Acknowledgements

We are deeply thankful to Matthew Kellett for useful guidance and feedback on this report.

This page intentionally left blank.

1 Introduction

This report is the sequel of a previous report (Casteigts and Flocchini 2013, CR 2013-020) in which we present a set of models and formalisms devoted to the study of dynamic networks. While the first report puts the stress on definitions and concepts, this report focuses on problems, algorithms, and analysis techniques, using the former as a basis. To the highest extent possible, we make this report as self-contained as possible by recalling informally the main definitions wherever needed; nevertheless, if some definitional aspects remain unclear to the reader, we invite her to browse the first report in order to find appropriate details and answers.

Telecommunication networks are in the middle of a significant mutation. They are becoming ubiquitous and the number and diversity of those being deployed in a dynamic environment is growing quickly. This state of facts holds in our everyday life (e.g., smartphones, vehicles, or satellites) as well as in a military context (e.g., dismounted soldiers or swarms of UAVs). Few theoretical tools enable, to date, the study of such dynamic networks in a formal and rigorous way. On the contrary, the current trend is to rely on simulations in order to assess the behaviour of a given solution in a given type of network, with ins and outs that are, by definition, not fully understood.

Building on top of dedicated models and formalisms, we review recent efforts toward the design and analysis of distributed algorithms in dynamic networks with an emphasis on those results that are of a deterministic and analytical nature. A natural emphasis is placed on results in which the authors have been involved. The topics discussed include how mobility impacts the very definition of problems; a set of generic tools to be used in the design or analysis of dynamic network algorithms; a discussion on the impact of various types of dynamics on the feasibility and complexity of algorithms; a classification of networks and algorithms based on their (requirement for) dynamics; and finally how real-world mobility contexts relate to some of these classes.

The fact that the network is dynamic has a significant impact on the kind of tasks that can be performed. In general, the impact of dynamicity ranges from suggesting new optimality metrics (e.g., foremost, shortest, and fastest broadcast), to making a problem harder (e.g., exploration of dynamic networks), to suggesting radically different definitions of a given problem whose original formulation would make no sense or become ambiguous in a dynamic context. For example, what is a *spanning tree* in a partitioned dynamic network? Is it a tree whose logical structure is realized over time and that contains all the nodes, or is it a set of disjoint trees (a *forest*) such that each tree at a given time covers some connected subset of the nodes (ideally, one such tree per connected component). As we will see, a whole class of combinatorial problems and their distributed analogues can actually be reframed into (at least) three different ways in a dynamic network, which we refer to as the “temporal”, the “evolving”, and the “permanent” variants. These topics are the main subject of Section 3.

Unsurprisingly, the formulation of new problems or variants of problems call for appropriate solutions and design techniques. We review in Section 4 a list of conceptual tools that can be used in the design of distributed algorithms for dynamic networks. The list is certainly

not comprehensive and research in the area is still at an early stage. We focus on those techniques or constructs that are quite generic and susceptible to being applied in a variety of situations. They range from generic constructs that can be used as building blocks (e.g., temporal-lag vector clocks), to abstract computational models, to the use of progression hypotheses or invariants for correctness.

Besides the complexity in time or in number of messages, a common approach for analyzing distributed algorithms is to look at the assumptions they require on the underlying network. We discuss in detail in Section 5 how this approach can be extended to dynamic networks; in particular, how a given property of the evolution of the network can be rigorously shown to be necessary or sufficient to the success of an algorithm. Several examples are provided for basic distributed problems such as broadcast, counting or election. These examples result in a number of *classes* of dynamic graphs, each of which captures the set of dynamic networks in which a given solution is feasible or not.

In Section 6, we consider three additional classes of dynamic graphs, whose edges are respectively recurrent, recurrent within a bounded time, or periodic. We review a recent work that studied the *computational relationship* between these classes, considering different types of knowledge such as the number n of nodes in the network, a bound Δ on the recurrence time, and a period p on the edges schedule. The question is in particular about the relations between $\mathcal{P}(\mathcal{R}_n)$, $\mathcal{P}(\mathcal{B}_\Delta)$, and $\mathcal{P}(\mathcal{P}_p)$, where $\mathcal{P}(\mathcal{C}_k)$ is the set of problems \mathcal{P} one can solve in class \mathcal{C} with knowledge k . These contexts are shown to form a strict computational hierarchy.

Section 7 reviews a number of classes of dynamic networks that are based on logical properties on dynamic graphs. Some of these classes are those resulting from the analyses in Section 5 and Section 6; a handful of others are from the recent literature on mobile ad hoc networks and delay tolerant networks. We show how these classes can be related to each other through inclusion relations, and how this particular approach allows us to compare algorithms in turn on the fair basis of their topological requirements.

Finally, we share some thoughts in Section 8 regarding the relation between real-world mobility contexts and a number of classes of dynamic graphs, with the prospect of understanding better each context and enabling systematic transfer of results among them. We conclude in Section 9 with some remarks and open problems.

2 Background

This report is the sequel of a previous report (Casteigts and Flocchini 2013) in which we present a set of models and formalisms devoted to the study of dynamic networks. These models and formalisms allow us to formulate a number of concepts that have appeared in recent literature. That report focuses, in particular, on those concepts of a temporal nature, which follow from relaxing the connectivity assumption in a dynamic network. The main examples include temporal variants of the concepts of *connectivity*, *distances*, and *paths*.

We review in the table below some of these concepts, the most central ones, to which the reader is invited to refer when a doubt arises in the reading.

Concept	Meaning	Notation
Presence function	Indicates whether an edge e is present at time t .	$\rho(e, t)$.
Latency function	Gives the crossing delay (latency) of edge e at time t .	$\zeta(e, t)$.
Journey	Path over time from a node u to a node v .	$\mathcal{J}_{(u,v)}$
Topological length	of a journey \mathcal{J} : its number of hops.	$ \mathcal{J} _h$
Temporal length	of a journey \mathcal{J} : its overall duration.	$ \mathcal{J} _t$
Departure	of a journey \mathcal{J} : date when first edge starts being crossed.	$departure(\mathcal{J})$
Arrival	of a journey \mathcal{J} : date when last edge ends being crossed.	$arrival(\mathcal{J})$
Temporal view	that a node v has of another node u at time t : latest time an event at u could influence v by time t .	$\phi_{v,t}(u)$
Temporal distance	Min time to reach node v from node u at time t .	$\hat{d}_{u,t}(v)$

Figure 1: Summary of key concepts

A *direct* journey is a journey that uses no waiting time at intermediate nodes. Contrasting with this notion, *indirect* journeys are those journeys where at least one hop requires some pause at the underlying node before being executed (typically, waiting for the next edge to appear). The set of all possible journeys from a node u to a node v is denoted as $\mathcal{J}_{(u,v)}^*$, and we allow the shorthand $u \rightsquigarrow v$ to mean that a journey possibly exists from node u to node v , that is, $\mathcal{J}_{(u,v)}^* \neq \emptyset$. It is equivalent to say that node u can causally influence node v . By convention, $u \rightsquigarrow u$ for any node u . Finally, we define the *horizon* of a node u as the set $\{v \in V : u \rightsquigarrow v\}$, that is, the set of nodes it can reach (or influence).

In this report, dynamic graphs are sometimes called time-varying graph (TVG) and sometimes evolving graphs. If the graph is studied from a global perspective, as a sequence of static snapshots $\mathcal{G} = \{G_1, G_2, \dots\}$, whether in discrete-time or in continuous-time with a time index $\{t_1, t_2, \dots\}$, we call it an evolving graph. If we consider it from a local perspective, through the quadruplet $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ where V and E are as usual, \mathcal{T} is the time span of the network (*a.k.a.* lifetime), and ρ is a function that indicates whether a given edge is available at a given time (presence function), then we call it a time-varying graph. Sometimes, we consider additional information—such as a latency function ζ that indicates the time it takes to cross a given edge at a given time—which yields a more elaborate definition, *i.e.*,

$\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$. Similarly, two functions of the presence and latency of nodes could also be considered in addition to edges; however, none of the examples in this report require such a definition.

Note that evolving graphs and time-varying graphs are equivalent formalisms up to some very theoretical limits—the presence function ρ might generate sequences that are uncountable in the TVG case, whereas the fact of considering a *sequence* of snapshots in the case of evolving graph limits it to countable transitions. However, these concerns are outside the scope of this report, and whether we use one or the other here only depends on how convenient they are in the underlying context.

These graphs are usually depicted using their underlying structure (the union of nodes and edges, also called *underlying graph*), on top of which the presence of edges is depicted with time intervals. In the example of Figure 2, edge (a, b) is present from 20 (inclusive) to 30

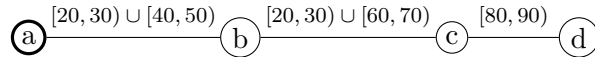


Figure 2: Example of a dynamic graph.

(exclusive), and then again from 40 (inclusive) to 50 (exclusive), and so is the meaning for (b, c) and (c, d) .

We invite the reader to have a look at our first report to explore deeper the modelling aspects of dynamic graphs, as well as the new concepts and metrics these graphs induce.

3 Nature of the problems

The fact that a network is dynamic has a significant impact on the kind of tasks one can perform in it. In general, the impact of dynamicity ranges from making a problem harder (e.g., *black hole search* by mobile agents in dynamic networks, discussed in Section 3.2) to suggesting new metrics (e.g., in the case of *broadcast* or *routing*), to inducing radically different definitions of a given problem, whose original formulation would make no sense or become ambiguous in a dynamic context. This holds in particular for networks in which connectivity takes place over time (highly mobile ad hoc networks or *delay-tolerant* networks). In such a context, many classical distributed problems must be redefined and there is several ways to do so. For example, what is a *spanning tree* in a partitioned dynamic network? Is it a tree whose logical structure is realized over time and that contains all the nodes, or is it a set of changing trees (i.e., a dynamic *forest*) such that each tree at a given time covers a subset of nodes that are connected at that time. The same question holds for *election* or *counting*. Deciding which definition to adopt depends on the target application and technological environment. As we will see, a whole class of combinatorial problems and their distributed analogues could be reframed in (at least) three different ways in dynamic networks. The present section explores these definitional variations through a handful of classical problems in distributed computing, namely, *broadcast* and *routing*, *election*, *spanning trees*, *dominating sets* and other *covering* problems. Figures 3 and 4 illustrate some of the main problems, in their classical (i.e., static) version. We provide for each an informal description of the objectives, and the initial and final states.

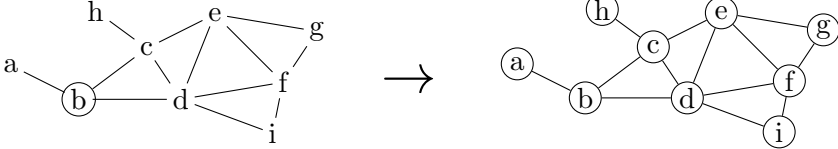
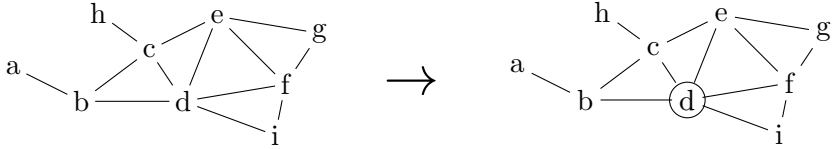
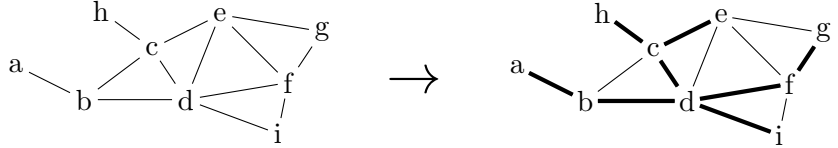
Problem	Informal definition and illustration
Broadcast	<p data-bbox="539 1075 1344 1108">Propagating a piece of information from one node to all others.</p> 
Election	<p data-bbox="623 1285 1260 1318">Distinguishing exactly one node among all others.</p> 
Spanning tree	<p data-bbox="487 1495 1396 1528">Selecting a cycle-free set of edges that interconnects the whole network.</p> 

Figure 3: Example of six classical distributed problems (part 1).

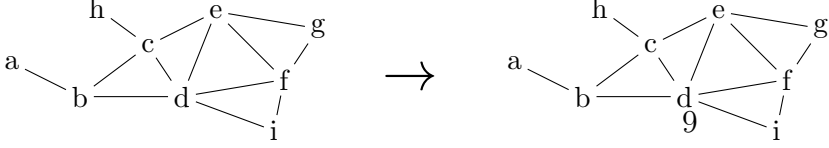
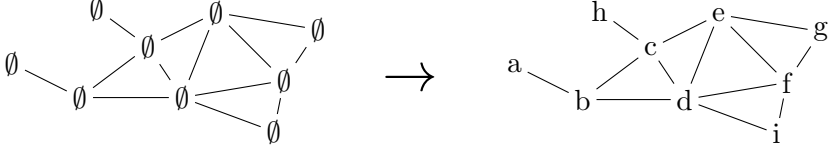
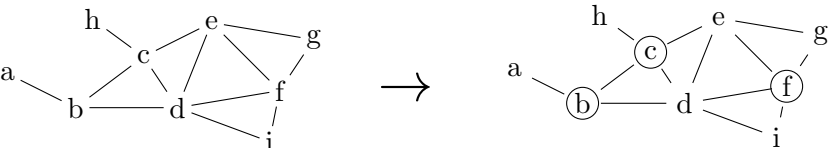
Problem	Informal definition and illustration
Counting	<p data-bbox="609 243 1312 275">Getting one node to know the overall number of nodes.</p> 
Naming	<p data-bbox="678 453 1242 485">Assigning unique identifiers to all the nodes.</p> 
Dominating set	<p data-bbox="508 663 1409 695">Finding a set of nodes such that all other nodes have a neighbour in it.</p> 

Figure 4: Example of six classical distributed problems (part 2).

3.1 New metrics for existing problems

Dealing with dynamic networks and in particular with temporal connectivity—the fact that a path requires waiting at intermediate nodes to be completed over time and space—has a significant impact on the kind of metrics an algorithm can optimize. Let us illustrate these aspects through the problems of *broadcast* and *routing*. The *broadcast* problem refers to propagating a message from one initial node (referred to as the source or emitter) to every other node in the network (See Figure 3). A trivial way to broadcast in a connected static network is to flood the network, that is, to have every node repeating the message to each of its neighbour whenever it receives it for the first time. However, an additional objective is usually to minimize some quantity related to the resources taken by the process, such as time, energy, or the number of messages. Propagating the message along a tree structure has good properties in this respect, and is therefore common-practice; see Figure 5 for an example.

Broadcasting in a *dynamic* network is different in that the links of the topology are time-dependent. Furthermore, the network is not guaranteed to be connected at any instant, and thus, nodes should be able to buffer the message for some time before propagating it further, i.e., propagation is made by means of *journeys* rather than paths; see detailed definitions in (Casteigts and Flocchini 2013) or Section 2 for informal definitions. One consequence is the emergence of new optimality criteria (or *metrics*) to be optimized. Indeed, the optimality of a journey can be defined both in topological or temporal terms. Precisely, a journey could be shortest, fastest, or foremost depending on which of its attribute is considered, as illustrated on Figure 6. Given a dynamic graph whose edge schedule is known

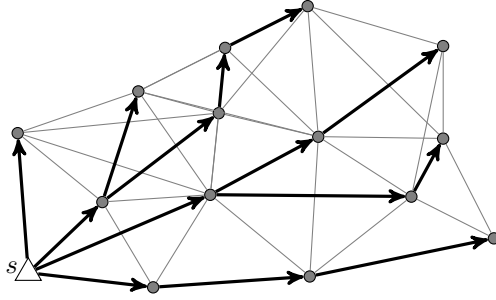
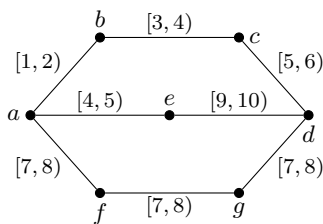


Figure 5: Example of tree-based broadcast from a source s .

or unknown, or unknown with some restrictions (e.g., periodicity of the edges), the problems of broadcasting in a foremost, shortest, or fastest ways are very different in essence. These three variants were first introduced by Bui-Xuan et al. (2003) in the case that the network schedule is known beforehand and available to a central algorithm. The distributed version of these problems, without knowledge of the schedule (but with a variety of assumption regarding regularities and knowledge) was later explored by Casteigts et al. (2010a, 2012b). A treatment of the solutions and techniques related to this problem is proposed in Section 4. An essential quality of broadcasting in dynamic networks is that the optimality of a solution is *time-dependent*: a solution might be optimal for some emission date, but not for others, which adds complexity to the structure of a solution—typically, a set of time-dependent journey-based trees instead of a single path-based tree, as illustrated by Figure 7. These trees do not indicate, strictly speaking, what journeys to follow (i.e., paths together with crossing dates), but only what are the underlying paths, which is enough (every edge being used as early as possible). For example, the foremost tree corresponding to initiation date 50 goes through b , so a forwards the message to b at date 50, then b knows it must forward to c , which occurs at 70 when the corresponding edge appears.

The problem of routing is closely related to that of broadcasting. The essential difference in routing is that the message is intended for a single destination node that is already identified before propagation, and the objective is to find the best route to deliver the message to this destination. Variants exist where the destination is a group of node (multicast) or even a location (geographic routing). As with broadcast, the routing problem must be reformulated in terms of journeys, and thus in terms of the foremost, shortest, or fastest qualities. Note



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

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

Figure 6: Different meanings for length and distance based on journeys. The intervals of time on edges indicates when they are present.

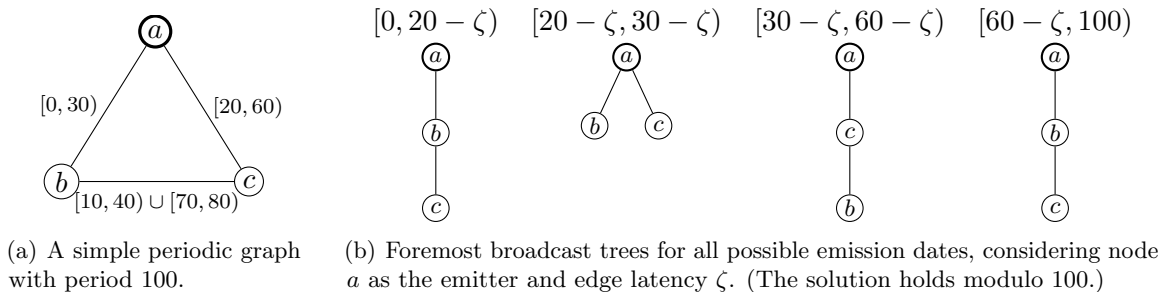


Figure 7: Example of foremost broadcast trees, which are time-dependent entities. Picture from (Casteigts et al. 2012b).

that, in general, having a complete solution to the broadcast problem (e.g., the set of all time-dependent optimal trees) allows one to solve routing trivially. However this is more than is needed in many cases and thus not the preferred approach. Various techniques have been developed in delay-tolerant networks based for example on proactive knowledge on the network schedule (Jain et al. 2004), probabilistic (Lindgren et al. 2003) or encounter-based strategies (Dubois-Ferriere et al. 2003, Grossglauser and Vetterli 2003, Jones et al. 2007). A taxonomy of routing approaches is proposed by Zhang (2006).

3.2 New environments for existing problems

Besides the emergence of new metrics, dealing with dynamic environments also suggests new contexts for classical problems. Taking *exploration* as an example, we review recent works that have addressed it in various types of dynamic networks. The problem also gives us an opportunity to discuss the mobile agent paradigm, frequently used in this context. Agents are computational entities that have the ability to migrate and change their execution support (underlying node). As such, they can perform some computation at a node, move to another node, then perform some new computation again, and so on. Several variants exist depending on the abilities of the agents, e.g., to carry data with them, to modify the underlying node, to communicate with other agents; or depending on the initial conditions—in particular, whether they are initially scattered or co-located on a same node. Mobile agents were proven computationally equivalent to message passing algorithms for several of these variants (Barrière et al. 2003, Chalopin et al. 2006, Das et al. 2007a). However, they suggest a programming paradigm that is much different in essence and facilitate the natural expression (and thinking) of some distributed problems like exploration.

The exploration problem consists in having the agent(s) visit all nodes of a network, either alone or collectively. The map construction problem is exploration where the goal is to create a map of the underlying network by exploring it. Variants include the case where some nodes are malicious and try preventing the agents from a successful exploration (e.g. making them disappear, like in the case of *black holes* nodes). We review below some recent work involving mobile agents to solve such problems in dynamic networks. The interested reader is referred to Kellett (2012) for a thorough treatment of the subject.

3.2.1 Exploration in static graphs

Concretely, the problem of exploration is to have one or several computational entities (called agents) cross every edge in a network—or every node, for some variants. Early work on related problems consisted in solving maze or exploring planar graphs. Then came the exploration of various types of topologies such as (un)directed graphs, trees, or rings. Some work aims to solve these problems from a centralized standpoint; however, these fall outside the scope of this report. On the distributed side, solutions exploit information that is available from within the network.

The objectives of exploration can be varied: to solve the problem repeatedly (*perpetual exploration*); to recognize when the problem has been solved (*exploration with stop*); to return to the starting point once the exploration is finished (*exploration with return*); or to terminate when a map is available (*map construction*). The performance of a solution is usually assessed in terms of the number of edges traversed or the number of moves performed (whether as an absolute or as a competitive ratio over the offline optimal solution).

Next is a list of key references in the domain. This list corresponds to the literature covered in Kellett (2012), which comprises a good deal of the existing material. The type of entity doing the exploration varies among papers. The first paper related to exploration is actually about a physical maze-solving machine built by Shannon (1951). The rest of the papers refer either to the TSP problem: Averbakh and Berman (1996); to finite automata: Blum and Kozen (1978), Fraigniaud et al. (2005); to robots or mobile agents: Ambühl et al. (2011), Albers and Henzinger (2000), Ambühl et al. (2011), Awerbuch et al. (1999), Bender et al. (1998), Bender and Slonim (1994), Das et al. (2007b, 2005), Deng and Papadimitriou (1990, 1999), Dessmark and Pelc (2004), Dudek et al. (1991), Dynia et al. (2007), Flocchini et al. (2007, 2008, 2010b, 2009a), Fraigniaud et al. (2004, 2006a, 2005, 2006b, 2004, 2006a), Gasieniec et al. (2007), Panaite and Pelc (1999). In these investigations, the entities have a variety of features and their computational power depends on their characteristics. In all cases the entities move autonomously from node to neighbouring node following a deterministic protocol, and have some way of harnessing information, whether from the topology or from other agents. The entities typically have some limited memory to store information, but they could also be memoryless, taking actions without remembering their past. Communication between entities can also take various forms, from direct communication (occurring when present at the same node) to acting on the underlying network (e.g. using pebbles to be dropped on the nodes and subsequently picked up, or whiteboards present at the nodes on which information can be written).

3.2.2 Exploration in periodically-varying graphs

Flocchini et al. (2009b, 2010a, 2012a,b,c,d) started to look at exploration in a class of dynamic networks where every node follows a periodic route. The resulting class of graph was called periodically varying graphs (PV graphs). The type of PV graph studied in these works is well exemplified by a bus system where “carriers” move synchronously between stations (called *sites*). Hence, there are three types of entities here: the carriers, the sites, and the exploring agent(s). To explore the PV graph, an agent can jump from carrier to

carrier when and only when both are located at a same site. The model is presented in more details below. This model was subsequently used in several works around PV graphs, including (Flocchini et al. 2010a, 2012a,b,c,d, Ilcinkas and Wade 2011, Brejová et al. 2011).

The carrier model: Consider a set C of n_C carriers that move among a set S of n_S sites. A carrier $c \in C$ follows a route $R(c)$ between all the sites in its domain $S(c) = \{s_0, s_1, \dots, s_{n_S(c)-1}\} \subseteq S$, where $n_S(c) = |S(c)|$. A carrier's route $R(c) = \langle r_0, r_1, \dots, r_{l(c)-1} \rangle$ is a cyclic sequence of *stops*: after stopping at site $r_i \in S(c)$, the carrier moves to $r_{i+1} \in S(c)$. All operations on the indices are taken modulo $l(c) = |R(c)|$, called the *length* of the route. It is assumed that carriers move *asynchronously*, taking a finite but unpredictable amount of time to move between stops. A route is called *simple* if $n_S(c) = l(c)$. A *transfer* site is any site that is in the domain of two or more carriers.

A carrier's route $R(c) = \langle r_0, r_1, \dots, r_{l(c)-1} \rangle$ defines an edge-labelled directed multigraph $\vec{G}(c) = (S(c), \vec{E}(c), \lambda(c))$, called a *carrier graph*, where $S(c)$ is the set of nodes, $\vec{E}(c)$ is the set of edges, and $\lambda(c)$ is the set of labels, and where there is an edge labelled $(c, i+1)$ from r_i to r_{i+1} (again, modulo $l(c)$). The entire network is then represented by the edge-labelled directed multigraph $\vec{G} = (R, \vec{E}, \lambda)$, called the *PV graph*, where $R = \cup_{c \in C} R(c)$, $\vec{E} = \cup_{c \in C} \vec{E}(c)$, and $\lambda = \{\lambda(c) : c \in C\}$. Associated to this graph is the *transfer graph* of \vec{G} , which is defined as the edge-labelled undirected multigraph $H(\vec{G}) = (C, E_T)$ where the nodes are the carriers and, $\forall c, c' \in C, c \neq c', s \in S$, there is an edge between c and c' labelled s iff $s \in S(c) \cap S(c')$, i.e., s is a transfer site between c and c' .

Note that in this original model, agents cannot stop at sites; however, subsequent works, such as those described in Section 3.2.3, assume that such a stop is possible and that agents can communicate through whiteboards at the sites.

PVG exploration: The exploration problem can be extended in the natural way to PV graphs, that is, as the process of a visiting all the nodes (sites) and exiting the system within finite time. Flocchini et al. (2009b) study the computability and the complexity of this problem. The paper first investigate the computability of PVG exploration and establishes necessary conditions for the problem to be solvable. It is proven that in anonymous systems (i.e., where nodes have no identifiers) exploration is unsolvable if the agent has no knowledge of (an upper bound on) the size of the largest route; if the nodes have distinct ids, then either n or an upper-bound on the system period must be known for the problem to be solvable. These necessary conditions for anonymous systems hold even if the routes are homogeneous and the agent has unlimited memory and knows k .

The study also considers the complexity of *PVG exploration* and establishes lower bounds on the number of moves. It is shown that in general $\Omega(kp)$ moves are necessary for homogeneous systems and $\Omega(kp^2)$ for heterogeneous ones, where p is the length of the longest route. This lower bound holds even if the agent knows n, k, p , and has unlimited memory. Notice that the parameter p in the above lower bounds can be arbitrarily large since the same site can appear in a route arbitrarily many times. A natural question is whether the lower bounds change when restrictions are imposed on the “redundancy” of routes.

To investigate the impact of the routes structure on the complexity of the problem, Flocchini et al. (2009b) consider PV graphs where all the routes are *simple*, that is, do not contain self-loops nor multiple edges. It is shown that the same type of lower bound holds also for this class, through establishing a $\Omega(kn^2)$ lower bound for homogeneous and $\Omega(kn^4)$ lower bound for heterogeneous systems with simple route. Then each route is further restricted to be *circular*, that is, an edge appears in a route at most once. Even in this case, the general lower bound holds; in fact, it takes $\Omega(kn)$ moves for homogeneous and $\Omega(kn^2)$ for heterogeneous systems with circular routes. Interestingly, these lower bounds hold even if the agent has full knowledge of the entire PV graph and has unlimited memory.

These limitations on computability and complexity are shown to be tight. This bound is proven constructively by presenting two worst case optimal solutions: one for anonymous systems and one for those with ids. The algorithm for anonymous systems solves the problem without requiring any knowledge of n or k ; it only uses the necessary knowledge of an upper bound $B \geq p$ on the size of the longest route. The number of moves is $O(kB)$ for homogeneous and $O(kB^2)$ for heterogeneous systems. This cost depends on the accuracy of the upperbound B on p , and the algorithm is optimal so long as B proportional to p . In the case with identifiers, the algorithm solves the problem without requiring any knowledge of p or k ; in fact it only uses the necessary knowledge of n . The number of moves in this case is $O(kp)$ and $O(kp^2)$, which matches the lower bound.

3.2.3 Exploration in subways

In a recent series of work, Flocchini et al. (2010a, 2012c,b), Kellett (2012) extend the PV graph result by asking the question, what happens if an agent can step down from the carrier and wait at sites? Such a context led the authors to the definition of the “subway” model, also described in detail in (Kellett 2012, Chapter 2).

The subway model: This model considers k agents that start at unpredictable times. As previously, the agents move opportunistically around the network using the carriers as medium. Here, however, an agent can move from a carrier to a site (*disembark*) or from a site to a carrier (*board*), but it cannot move directly from one carrier to another. During its journey on a carrier, the agent can count the number of stops (sites) that the carrier has passed, and decide to disembark accordingly.

The agents in this model can communicate with each other using *whiteboards* that are located at each site (Flocchini et al. 2010a, 2012c), or on the carriers themselves Flocchini et al. (2012b). All agents execute the same protocol in an *asynchronous* fashion (each one takes a finite but unpredictable amount of time to perform computations). Several options were considered regarding the starting time and site for each agents: either they start at varying times co-located on the same site (case with whiteboard at sites), or they start at varying times scattered on possibly different sites (case with whiteboard at carriers). The site on which an agent starts is called its *homebase*.

Black hole search in general: Until this line of work, most of the existing work on black hole search had focused on the problem of a team of co-located agents finding a single black

hole in the network, as for example in (Dobrev et al. 2002, 2006) where the authors prove a tight lower bound of $\Omega(n^2)$ moves when the agents are completely ignorant of the underlying network. These results were extended in several ways. First, Flocchini et al. (2009a) presents a solution using agents that start scattered in the network. Instead of a single black hole, any number of black holes and black links is allowed, with some limitations. (This context is referred to as the exploration of *dangerous graphs*.) The solution solves the problem in $O(nm)$ moves, where n is the number of nodes, and m is the number of links; see Flocchini et al. (2009a) and (Kellett 2012, Chapter 4) for details.

Black hole search in the subway model: Getting back to the subway model, in the case that agents start co-located on a same site and communicate using whiteboards on the stations, Flocchini et al. (2010a, 2012c) prove some basic limitations and propose a solution that requires $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves. The case where agents start scattered in the network and communicate using whiteboards *on the carriers* is studied in Flocchini et al. (2012b). The presented solution is based on the cooperative independent exploration of the network by the agents and has a complexity of $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves.

The two solutions mentioned above are again optimal, based on a lower bound, by the same authors, on the complexity of any solution to the black hole search problem. Such a solution requires $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where γ is the number of stops that all carriers make on black hole sites. Note that carrier moves is a complexity measure that is specific to the subway model (while being similar to agent moves in the carrier model). Both solutions also work with an optimal number of agents, $k = \gamma + 1$. See (Kellett 2012, Chapter 6) for details on the lower bounds.

A note on the walking technique: A basic technique in the presence of black holes is the use of *cautious walk*. A cautious walk is a walk whereby an agent steps back after every progression; that is, it moves two steps forward, then one step backward, then two steps forwards, and so on and so forth. The agent also leaves some mark so that if it fails to make the backward step because it encounters some dangerous link or node, the other agents can avoid making the same mistake. This simple technique is appealing because it does not induce a rise in complexity (the total number of moves is only multiplied by 3—a constant factor). It is illustrated on a line by Figure 8.

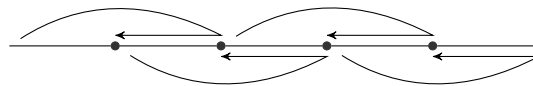


Figure 8: Example of cautious walk on a line.

All the explorations without black holes rely on some forms of traversals, so traversal by the agents is the basic tool there, performed in different ways depending on the knowledge available.

3.2.4 Exploration in networks with edge deletions

Even more recently, Flocchini et al. (2012a) started to address the exploration problem in a fault-tolerant context, where an arbitrary number of links can be deleted in an adversarial way (with some restrictions on the adversary). The model considered here is not the subway model nor the carrier model, but a more classical network model. The problem is to visit every (accessible) link in the network, marking locally the frontier links leading to black holes and the (accessible) black links. The problem is considered solved if at least one agent survives (and all agents either die or terminate within finite time). The solution proposed in the paper uses an optimal number of agents and $O(nm^2)$ moves, where n is the number of nodes and m is the number of edges.

The network model: Let $G = (V, E)$ be a simple connected undirected graph with n nodes and $m = |E|$ links. A team of k agents is considered. Agents start scattered in the network at varying times, follow the same algorithm, and move asynchronously. Each node in the network is equipped with a whiteboard on which the agents can read/write in fair mutual exclusion. A subset V_B of the nodes are black holes and a subset E_B of the links are black links; both being characterized by the fact that they make agents disappear. Let F_B be the set of frontier links, which are links that connect safe nodes to black holes. Let E_I be the set of inaccessible links, which are those links in-between two black holes. The safe portion of the network $G_S = (V_S, E_S)$ is assumed to be connected and the adversary cannot violate this property. Neither can it delete a link while an agent is crossing it. Finally, it is assumed that the agents know the number of safe nodes, $n_S = |V_S|$ (otherwise termination is impossible).

An algorithm that solves this problem is presented in Flocchini et al. (2012a). This algorithm, called *ExploreDG-LD*, requires $k \geq f + 1$ agents, where $f = |F_B| + 2|E_B \setminus (E_I \cup F_B)|$. It proceeds by having each exploring agent build a tree out from its homebase. The cautious walk technique (see Figure 8) is used during exploration to ensure that only one agent is eliminated per frontier link and at most two agents are eliminated per black link (one from each endpoint). Contact between two trees are detected when a non-tree link is incident to a node in each tree. Then both trees are merged on that link. An agent terminates the algorithm when the underlying tree contains n_S nodes and there is no verification or exploration work left. Clearly, link deletions complicate the process of building and merging trees, in particular the deletion of a tree link can have a significant effect. Hence, the adversary can remove access of one or several agents to the coordinating information for the tree in which they are working. These deletions being undetectable, maps are used at the nodes and carried by the agents to detect such deletions. More details can be found in (Flocchini et al. 2012a), as well as in (Kellett 2012, Chapter 5).

3.3 Definitions of new problems

While the definition of problems like broadcast or routing remain intuitive in dynamic networks, many other problems become irrelevant or definitionally ambiguous. We first discuss two examples of classical distributed problems that can be reframed in two versions—a “temporal” one and an “instant-based” (also called “evolving”) one—then extend the

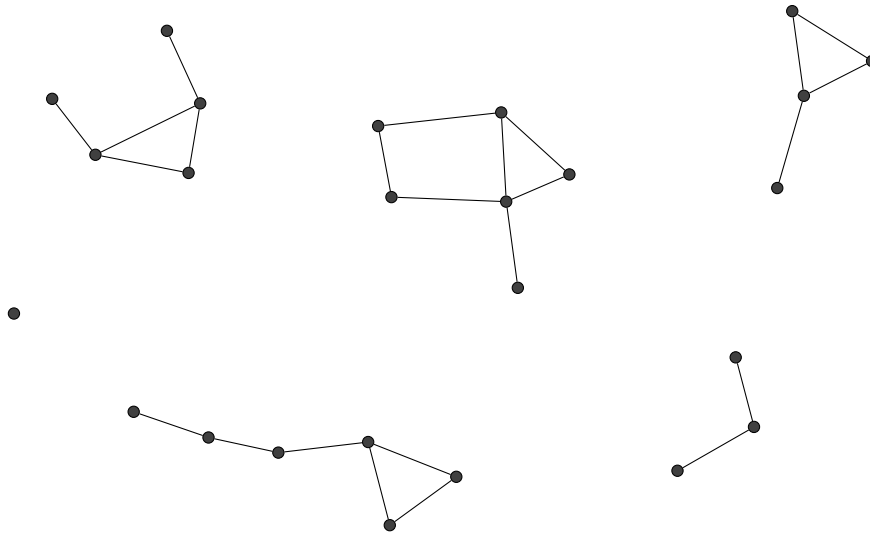


Figure 9: A typical MANET scenario.

same thinking to other classes of problems in which at least three distinct definitions make sense.

3.3.1 Election and spanning trees

Consider the example of *election* or *spanning tree* construction. In static networks, election is the problem of distinguishing one node, a *leader*, among all others (see Figure 3) in order to have it play a specific role afterwards. In a sense, the purpose of election is to create *centralization* in an otherwise *decentralized* setting, so as to facilitate subsequent coordination tasks (many distributed problems become simpler, or even trivial, once a unique leader has been identified, e.g., *naming* or *counting* in static network). What should the concept of a leader be in a dynamic network? Several options are available to redefine election in such a setting.

Before discussing them, consider typical instances of mobile ad hoc networks (MANETs), such as a set of mobile robots exploring a disaster site, or dismounted soldiers inspecting a territory (see Figure 9). In such a scenario, the network can be partitioned anytime into several connected components, each representing a given cluster of nodes that evolve semi-independently. Assume the temporal connectivity of the whole network can still be established over time, based on meetings among groups or group recomposition.

How to define the concept of election (and that of a leader) in this scenario? The most straightforward adaptation of the static version is certainly to distinguish a single node among the whole network, as illustrated on Figure 10(a). Such a node could be called a *temporal leader* because its role is realized over time and justified by the fact that the network is temporal connected (otherwise the leader could influence only a small subset of the network). However, if the groups remain united for a substantial amount of time and do not interconnect quickly, the above definition is inappropriate for the kind of coordination

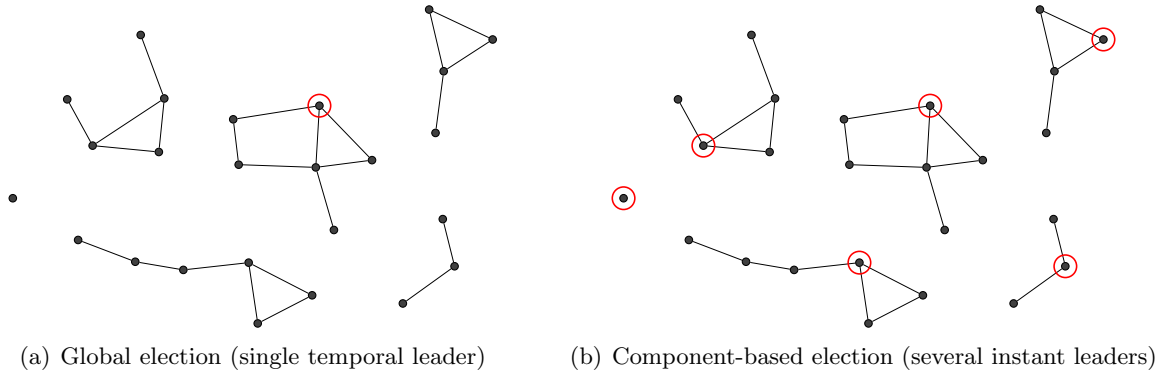


Figure 10: Two versions of the election problem.

action we expect from a leader. Having exactly one leader per connected component seems far more appropriate in this case (see Figure 10(b)). From an instantaneous perspective, this point of view is equivalent to considering each connected component as a distinct network where election must be performed. However, since the groups are subject to occasional splits and merges, the problem is one of *maintenance*: the objective is to maintain one and exactly one leader per component as the groups compositions evolve over time (e.g., by regenerating a leader when a group splits, or deleting one when two groups merge).

The same remarks hold for the problem of *spanning tree* construction. In static graphs, a spanning tree is a *connected* subset of the edges such that every node is adjacent to at least one edge in this set and these edges form no cycle (see Figure 3). Equivalently, there is one and exactly one possible path between any pair of node. This problem is closely related to the election problem because once a leader is identified, it is often easy to build a tree from that node (as the root of the tree) to the others, progressively. Several optimization metrics are usually considered, such as the average length of the edges or the diameter of the tree (maximal number of hops between any two nodes). Figure 11 shows an example of spanning tree in which the length of the longest edge is minimized. This type of solution is useful in wireless networks that assume that all the nodes must have the same communication range.

Just as for election, the spanning tree problem can be transposed into several dynamic versions. Besides variants where the dynamics are seen as transient and occasional faults (which is outside the scope of this report), we distinguish two main variants of the problem. The first one, a temporal version, has the objective of building a tree whose function is established over time and which, as a result, has logical links that do not evolve over time. The second version of the problem is to *maintain* a forest of spanning trees, each covering a distinct connected component of the network, as illustrated on Figure 12.

The list of problems accepting the same variations will likely grow further. It includes, at least, counting and naming (see Figure 4). In counting, we might either want to compute, over time, the total number of nodes in the whole network, or maintain the best approximation of the local number of nodes in each connected component. The uniqueness of identifiers in the naming problem could be similarly defined in terms of the whole network or just the

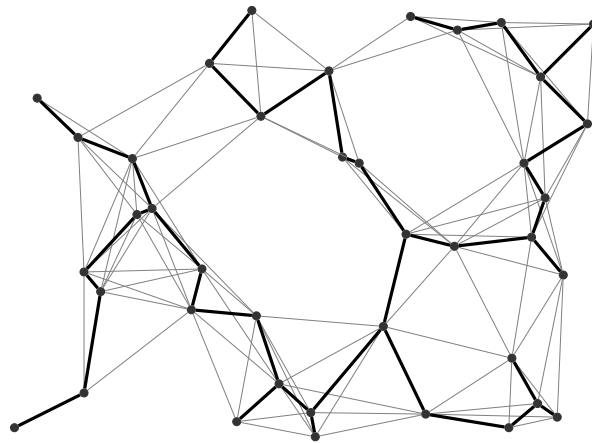


Figure 11: A spanning tree on top of a random *UDG*. Picture from (Casteigts et al. 2010b)

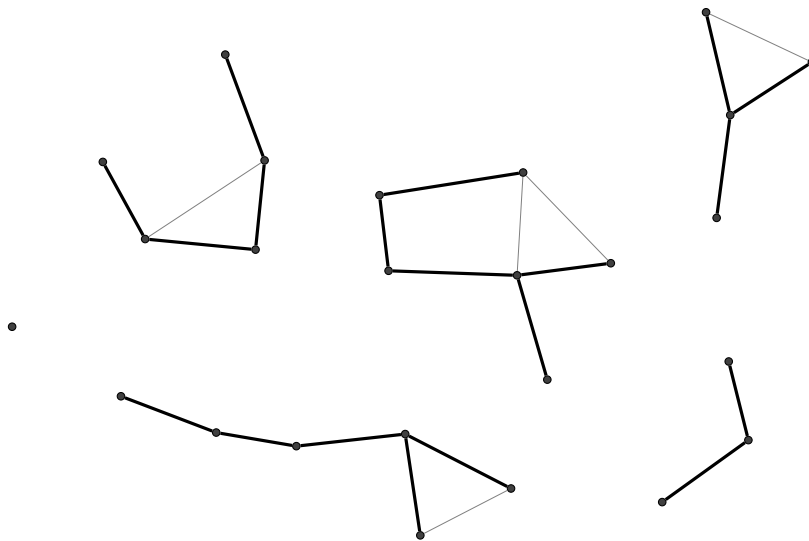


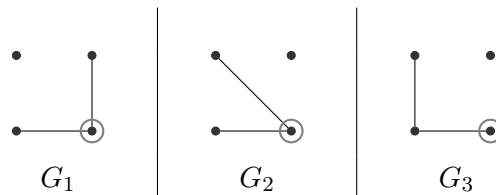
Figure 12: Example of a spanning forest over a MANET.

connected components (the latter could be relevant, for example, to minimizing the size of the identifiers). Algorithmic approaches related to election and spanning trees in both variants are discussed in Section 4.

3.3.2 Covering problems

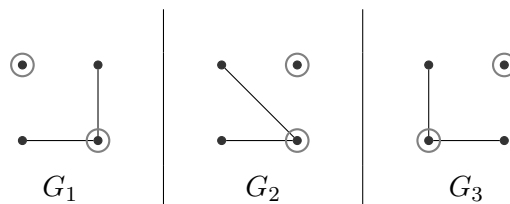
In Section 3.3.1, we discussed two basic ways to redefine a problem, and we illustrated it in the case of election and spanning trees. Some problems actually accept further variations in a way that still makes sense (Casteigts et al. 2011). Consider the example of dominating sets. In a static context, a *dominating set* is a subset of nodes DS such that each node in the network either is in DS or has a neighbour in DS . The goal is usually to minimize the size of DS . Given a dynamic network represented as a (discrete) *evolving graph* $\mathcal{G} = \{G_i\}$, we can reframe the problem in the following ways:

- *Temporal dominating set*: The solution is defined with respect to the sequence as a whole, over time. That is, the dominating set is such that each node is covered in *at least one* G_i . Here is an example of such a dominating set, for a given evolving graph $\mathcal{G} = \{ \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix}, \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix}, \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix} \}$:



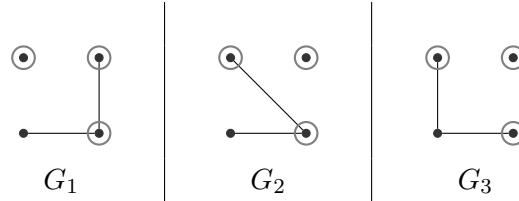
Here, the solution consists of a single node (the bottom right node), which is sufficient because this node will, over time, share an edge with every other node. Temporal dominating sets can be used, for example, to minimize the number of nodes that propagate a message while guaranteeing that all the nodes get it eventually.

- *Evolving dominating set*: In this version, every snapshot of the evolving graph sequence is seen as a separate graph for which an independent solution can be computed. As such, the dominating set evolves over the execution to contain only what is needed in each step. Here a possible solution for the same example is



This definition of the problem is also of practical relevance, for example, if the rate of changes in the network is low enough to consider each graph of the sequence as representative of a static period during which the network is stable. One important question here is whether the solution to one G_i could be used to compute *more quickly* the solution to G_{i+1} given some restrictions on the number of change between G_i and G_{i+1} . If this is the case, the algorithm becomes one of *maintenance* in a similar way as we discussed above for election and spanning trees.

- *Permanent* dominating set: Similar to the *temporal* dominating set, this solution does not change over time. In this case, however, it must remain valid for *every* graph of the sequence *taken individually*. We thus have to find a non-changing subset of nodes DS such that DS is a valid solution in any given G_i . A possible solution for the same example is



Observe that DS must contain at least 3 nodes here. Indeed, the top left node is isolated in G_1 and thus must belong to DS at least for this reason. The same goes for the top right node in G_3 . Then any of the two bottom nodes must be added to make up a valid solution in any of the three graphs.

At this point, the reader might wonder why the problems of spanning tree construction and election were not similarly translated into a *permanent* version. The reason is that the very concepts of a permanent leader or a permanent spanning tree would not make sense in some configurations. Consider the two evolving graphs in Figure 13. In the left scenario

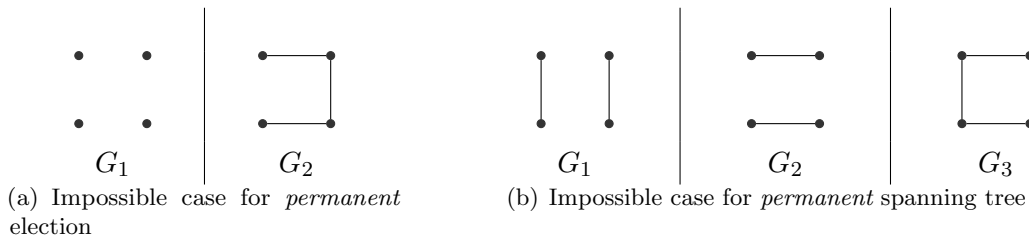


Figure 13: Why some problems cannot accept a “permanent” variation.

(Figure 13(a)), every node is initially isolated, which forces the selection of every node as leader of its component. Since the solution has to be “permanent”, the set of leaders cannot be updated for the next graph G_2 , which thus happens to have 4 leaders despite the fact that it is connected, thereby violating any reasonable definition of what a leader should be. By a similar argument, the right scenario (Figure 13(b)) forces the selection of all available edges through G_1 and G_2 , which violates in G_3 the cycle-free property a spanning tree should have. If the dominating set problem was considered with the additional constraint of being minimal (or minimum) in every G_i , its permanent variant would similarly be made inconsistent.

The dominating set problem (without additional constraints) is not the only problem that admits all three variants. Others include the *vertex cover* problem (the problem of finding a set of nodes such that every edge has at least one of its endpoints in the covering set) and the *edge cover* problem (finding a set of edges such that every node has at least one of its adjacent edges in covering set). Problems that do *not* accept the third variant include the

graph colouring problem (assigning a distinct colour to every neighbouring node while trying to minimize the total number of colors) for reasons similar to election and spanning tree construction.

Relation between the three variants

It can be observed that for problems like *dominating sets* or *vertex cover*, all three variants are strongly related (Casteigts et al. 2011). Given such a problem Π , one can easily check that a solution to *permanent* Π is a (possibly far from optimal) solution to *evolving* Π , and a solution to any G_i in the *evolving* Π is also a valid solution for *temporal* Π . The connection is even stronger: the *intersection* of solutions to *evolving* Π is a valid solution for *temporal* Π , and their *union* is a valid solution to *permanent* Π . From this perspective, the *evolving* variant appears quite central, and the *permanent* or *temporal* variants actually form upper and lower bounds for the size of the solution for the *evolving* variant.

From a distributed standpoint, the permanent and the temporal variants may appear a bit unnatural in that they require some knowledge of the future of the graph evolution. However, this knowledge is relevant in a few, yet important, practical scenarios where knowledge of this type can be inferred from combining regularity assumptions and past information. Examples include networks whose edges are *periodic* (Flocchini et al. 2009b, 2012c, Kellett 2012) or *recurrent* (Casteigts et al. 2010a). The latter means that if an edge existed once then it is guaranteed to re-appear at some future time (possibly known or unknown, or bounded or unbounded, depending on the cases).

A closer look at the evolving variant

A crucial question about the evolving variant is whether we can leverage the solution to graph G_i to compute the solution to G_{i+1} more quickly (or better within a given time). This question raises the general question: Is maintenance easier than construction from scratch? This question has several possible answers, depending on what exactly is meant by “easier”. Computational complexity tends to identify a *feasible* task with one that can be performed in *polynomial time*. In a nutshell, this means the execution time can be expressed as a polynomial whose indeterminate is the size of the problem (typically, for a graph problem, the number of nodes). For example, if the execution time is proportional to $4n^8 + 5n^4 + n^2$ where n is the number of nodes, then we say the corresponding solution is a polynomial time solution, and the corresponding problem is “in \mathbf{P} ”. Problems not in \mathbf{P} are viewed as being practically infeasible. Examples include *exponential-time* problems, that is, problems requiring an execution time exponential in n , e.g., $2^n + 3n^2$. A good deal of research in computational complexity consists in finding the boundary between \mathbf{P} and non- \mathbf{P} problems, through the study of a battery of basic problems.

Getting back to the matter at hand, the most straightforward interpretation of the “maintenance” question is whether a non- \mathbf{P} problem for some G_i could fall into \mathbf{P} if we knew already the solution to G_{i-1} , assuming G_i and G_{i-1} differ only by some elementary operation such as the single addition or deletion of an edge (let us call such graphs *neighbour*

graphs). Unfortunately, the answer is no, and there is a well-known argument to establish this.

By contradiction. Given a problem Π , assume there is no algorithm in \mathbf{P} that solves Π , but there exists a maintenance algorithm \mathcal{A} that does so. That is, given two neighbour graphs G_i and G_{i+1} and a solution to G_i , \mathcal{A} finds a solution to G_{i+1} in polynomial time. Now consider an arbitrary graph $G = (V, E)$. We can decompose G into an artificial sequence G_1, G_2, \dots, G_k such that $G_0 = (V, \emptyset)$ and $G_k = G$, and every pair G_i, G_{i+1} are neighbour graphs (in other words, we add a new edge in every G_i until we have recreated G). We can then solve G by applying algorithm \mathcal{A} k times starting from G_0 (whose solution is trivial). Hence, if maintaining Π is in \mathbf{P} then solving Π is at most $k = |E| \leq |V|^2$ times slower, which is still in \mathbf{P} , leading to a contradiction. A generalization of this argument to parametrized complexity, showing that the argument holds up to $W[1]$ -hard problems, is given by Casteigts et al. (2011). \square

This result sounds like a bad news for maintenance algorithms in general. Informally, it tells us the set of problems we can *maintain* is not larger than the set of problems we can *solve* from scratch. But remember this holds if we identify “feasible” with “polynomial time”. What if we strengthen the definition of a feasible task and require that it takes less than polynomial time? Considering the point of view of *distributed* maintenance algorithms, this seems a reasonable approach. Indeed, contrary to centralized algorithms that are in general executed *offline* (i.e., the execution is made at a different time than the network lifetime), distributed algorithms are supposed to be executed concurrently with the network evolution, which implies that the amount of time in-between two consecutive G_i is limited. It makes no sense to assume this time is polynomial, or even linear in the number of nodes. To be realistic, the time in between two G_i should be considered as a *constant* (i.e, independent from n), or, even worse, inversely proportional to n (i.e., the more nodes, the more frequent the changes). Under these assumptions, the question of whether maintenance is easier than construction from scratch becomes open again, and the answer might turn out to be yes for several cases. In essence, the matter gets to the line of research initiated by Naor and Stockmeyer twenty years ago asking what can be computed locally (Naor and Stockmeyer 1993, Suomela 2011). This avenue of research is exciting and we believe it is conceptually fertile.

The evolving variant in the literature

Ideas similar to the evolving variant have been explored from a non-distributed perspective under various names: *reoptimization*, *dynamic complexity theory*, and *incremental computation*. Typical previous work has focused on polynomial or logarithmic time problems, and did not include the consideration of locality. However, many interesting results and ideas could be applicable in a distributed context. Patnaik and Immerman (1997) consider dynamic complexity from a descriptive complexity theory perspective, defining DYNFO, a class of dynamic problems that are expressible in first order logic. Weber and Schwentick (2007) build upon this, again concentrating on a descriptive complexity approach. Holm et al. (2001) give a series of results that can readily be interpreted as maintenance algorithms

in our context. Their results for problems like *connectivity* and *2-Edge* or *biconnectivity* rely on the maintenance under edge deletion and addition of a solution for a *minimum spanning forest*, giving polylogarithmic running times for all problems, but with no bound on locality. [Miltersen et al. \(1994\)](#) present another, similar approach where the dynamism is achieved at a lower level by perturbing individual bits in the input. They also focus on problems of polynomial complexity showing, in our context, that problems such as the *circuit value problem* and *propositional horn satisfiability* have no polylogarithmic maintenance algorithms but that interestingly there exist other **P**-complete problems that do. [Ausiello et al. \(2011\)](#) discuss a different model, where there is only interest in using an existing optimal solution to solve to some degree of approximation a perturbed instance.

4 Algorithmic tools and techniques

We review a list of conceptual tools that can be used in the design of distributed algorithms for dynamic networks. The list is certainly not comprehensive and research in the area is still at an early stage. We focus on those techniques or constructs that are quite generic and capable of being applied in a variety of situations, with a natural emphasis on those results the authors have been involved in. The topics range from generic constructs that can be used as building blocks (e.g., temporal-lags vector clocks) to abstract computational models to the use of progression hypothesis or invariants for correctness.

4.1 Temporal-lags vector clocks (T-Clocks)

Highly-dynamic networks, and in particular delay-tolerant networks (DTNs), are characterized by a possible absence of contemporaneous end-to-end communication routes (routes in which each hop follows directly in time after the hop before, also called *direct journeys*). In most cases, however, communication can still be achieved over time and space through disconnected routes (*indirect journeys*) using store-carry-forward-like mechanisms. The duration of a given route in this context rarely depends solely on the number of hops. One question that immediately arises is *how far apart can nodes be, temporally?* Clearly, this quantity is time-dependent: it varies depending on when it is considered. *Could this quantity be measured precisely for every node, at each point in time?* As we will see, the knowledge of such temporal distances is instrumental in solving more complex problems, such as performing foremost or fastest broadcast (or routing) in some classes of dynamic graphs, or electing a node that is *temporally* central.

4.1.1 Measuring temporal lags

The general problem of measuring temporal lags in a dynamic network was addressed in a number of works from the field of social network analysis (Holme 2005, Kostakos 2009, Kossinets et al. 2008). Indeed, social networks are in essence similar to DTNs, but unlike them, social networks often produce large datasets of connection history. Kossinets et al. (2008) ask how out-of-date each node can be with respect to other nodes, and provide a *centralized* algorithm that processes a known sequence of contacts to measure these lags based on an adaptation of vector clocks. Besides looking at the question from a centralized point of view, these studies assume that contacts between nodes are punctual in time, i.e. they have no duration and can be given as triplets (u, v, t) where u and v are two nodes and t is the date of contact between them. This assumption, likely due to the nature of the data (e.g. email exchanges), is punctual in essence.

The specificity of delay-tolerant networks: The situation in DTNs is typically different because contacts between nodes can have arbitrary durations and overlap in time with each other (we will refer to such contacts as *non-punctual*). This aspect renders computation of temporal distances much more complex because it implies the possible co-existence of indirect routes on the one hand (routes with waiting pauses at intermediate nodes), and *continuums* of direct routes on the other hand (when the entirety of the route can be realized

in a row, as in usual static networks). Typical DTNs exhibit a mixture of both routes in various proportions. In a recent paper, [Casteigts et al. \(2012b\)](#) look at the question of measuring temporal lags in this more general setting and from a *distributed* point of view as well. Specifically, they ask whether it is possible for each node to know exactly—and in real time—how out of date it is with respect to all others? The question is answered positively in a continuous-time setting (assuming only a fixed and uniform latency for message propagation over single edges). Feasibility is demonstrated constructively. The algorithm generalizes the one from [Kossinets et al. \(2008\)](#) to non-punctual contacts and makes it distributed.

Temporal views: One of the most central concepts in this work is that of *temporal view*. The temporal view a node v has of a node u at time t , denoted $\phi_{v,t}(u)$, is the latest (i.e., largest) time or *date* $t' \leq t$ at which a message received by time t at v could have been emitted at u ; that is, in our formalism

$$\phi_{v,t}(u) = \text{Max}\{\text{departure}(\mathcal{J}) : \mathcal{J} \in \mathcal{J}_{(u,v)}^* \wedge \text{arrival}(\mathcal{J}) \leq t\}.$$

The fact that contacts have arbitrary durations makes it possible for adjacent contacts to overlap in time, thereby producing complex patterns of time lag between nodes. As an example, consider the plots in [Figure 14](#), showing the evolution of the temporal view

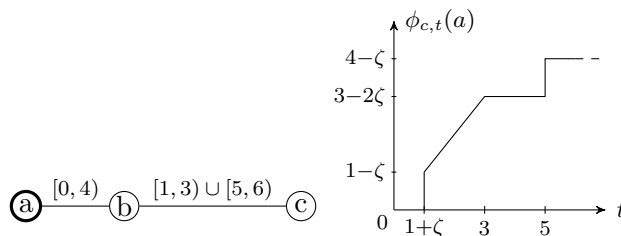


Figure 14: Temporal view that c has of a , as a function of time (with latency $\zeta \ll 1$). Example from ([Casteigts et al. 2012b](#)).

that node c has of node a in a basic TVG. Unlike the case with punctual contacts—where evolution occurs in discrete steps only—here there is a mixture of discrete and continuous evolution. (The reader is encouraged to spend a few minutes on this example as these concepts are essential in what follows.)

Direct journeys are often faster than indirect ones, however this is not necessarily the case (consider a very long direct journey versus a short indirect one whose edges traversals follow after a very short wait). As a result, the temporal view ϕ at any time could result from either type of journey. The views resulting from the best direct and indirect journeys at any given time are referred to as the *direct view* and the *indirect view*, respectively, in ([Casteigts et al. 2012b](#)). At any time, ϕ is in fact the maximum of the two.

The algorithm in ([Casteigts et al. 2012b](#)) tracks the evolution of both direct views and indirect views independently. While indirect views are stored as simple values and updated

punctually, when topological events occur, direct views are inferred, on demand, from the knowledge of topological distance from the considered remote node to the local node (i.e., the *level* of the local node with respect to the remote node). Thus, the algorithm consists in maintaining up-to-date information about two kinds of variables for every remote node: the *level* (for direct views), and the largest *date* at which a message carried to the local node through a indirect journey could have been emitted at the remote node (for indirect views). This algorithm is referred to as T-CLOCKS.

4.1.2 T-Clocks as a network abstraction

One way of using T-CLOCKS is to consider them as an *abstraction* that provides high-level information about the temporal views—namely, information to track both direct and indirect views. Technically, the abstraction consists of an intermediate layer between the network and some higher algorithm (see Figure 15), which it informs by means of generating two events: `levelChanged()`, reflecting the evolution of a direct view, and `dateImproved()`, reflecting the evolution of an indirect view.

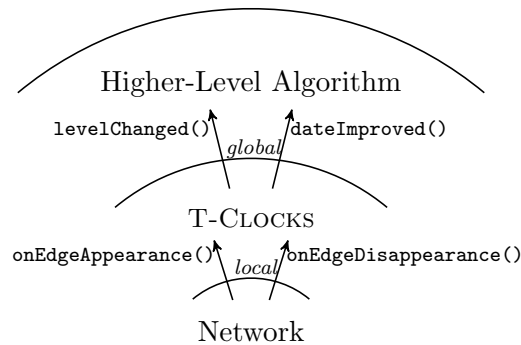


Figure 15: T-CLOCKS as an abstraction to track temporal views. Picture from (Casteigts et al. 2012b).

Casteigts et al. (2012b) illustrate how this abstraction can be used as a building block to solve more concrete problems. In particular, they provide algorithms to learn *foremost* broadcast trees: a set of broadcast trees that vary depending on the emitter and the emission date (modulo the period), guaranteeing the *earliest* possible delivery time at all nodes; and *fastest* broadcast trees: a set of broadcast trees that vary with the emitter and guarantee that the time spent between first message emission and last message reception is minimized, even if this means delaying the first emission.

4.1.3 Example: foremost broadcast trees

It is natural to ask what set of journeys a message emitted at a given source (or *emitter*) should follow to reach all nodes at the earliest possible date. As discussed above in Paragraph 4.1, this choice depends on which date the broadcast is initiated (the *initiation* date), and even then, several options may exist. A nice property of the foremost metric is that among all

the possible foremost journeys, there is (at least) one whose *prefixes* are themselves foremost journeys, i.e., every intermediary node is reached in a foremost fashion. This property allows one to consider, for a given emitter and initiation date, a *tree* of foremost journeys that we refer to as a *foremost broadcast tree* (foremost BT, for short) for that particular date. As an example, the foremost BTs corresponding to the graph of Figure 7(a) on page 8 for emitter *a* are shown in Figure 7(b) on page 8 as a function of the initiation date.

Periodically varying graphs: The assumption of periodicity is not strictly needed to perform a foremost broadcast (flooding the network would also do); however, it allows for the trees to be learnt and reused at a lower cost since the optimality of journeys clearly holds modulo *p*. Considering again the same example, the first tree is optimal for any initiation date in $[0, 19)$, or $[100, 119)$, or $[900, 919)$, *etc.* It is thus sufficient to build all foremost BTs relative to one single period, in order to solve the problem completely.

Locality of knowledge: An important aspect of computing foremost broadcast trees is that a node does not need to know the whole tree in order to make a local choice. The node needs only know which neighbours it should forward a message to, which is based on the source and initiation date. The corresponding information relative to source node *a* in Figure 7(a) is shown in Table 1.

Table 1: Set of children relative to emitter *a*.

on <i>a</i>	Initiation date	$[0, 19)$	$[19, 29)$	$[29, 59)$	$[59, 100)$
	Children	$\{b\}$	$\{b, c\}$	$\{c\}$	$\{b\}$
on <i>b</i>	Initiation date	$[0, 19)$	$[19, 59)$	$[59, 100)$	
	Children	$\{c\}$	\emptyset	$\{c\}$	
on <i>c</i>	Initiation date	$[0, 29)$	$[29, 59)$	$[59, 100)$	
	Children	\emptyset	$\{b\}$	\emptyset	

Learning children tables for all potential emitters is the purpose of the algorithm, whose informal strategy is as follows. The algorithm actually starts the other way around, with nodes determining their set of optimal *parents* in the trees, relative to one complete period of initiation dates and all sources. This is done based on information provided by the T-CLOCKS abstraction. The resulting information is stored in a structure equivalent to Table 2. Once

Table 2: Set of parents relative to emitter *a*.

on <i>b</i>	Initiation date	$[0, 29)$	$[29, 59)$	$[59, 100)$
	Parent	a	c	a
on <i>c</i>	Initiation date	$[0, 19)$	$[19, 59)$	$[59, 100)$
	Parent	b	a	b

a node has determined its set of parents with respect to a complete period of initiation dates, it notifies each parent by sending the corresponding intervals. On the parent side, the intervals are processed upon receipt to fill in their children table.

4.1.4 Example: fastest broadcast trees

While “foremost” refers to minimizing the arrival date, “fastest” refers to minimizing the overall time spent once the broadcast is initiated. As such, one might be willing to delay the effective starting date of a broadcast for the purpose of making it faster, which makes sense, for example, in communication networks whose medium is shared exclusively, or to minimize a trip duration in the context of transportation networks.

What a fastest broadcast “tree” should be: A fundamental difference between fastest and foremost journeys is that finding fastest journeys whose prefixes are themselves fastest may not always be possible. Consider the example in Figure 16, assuming node a is willing to broadcast at time 0. Reaching d in a fastest way requires a to send the first message at $49 - \epsilon$ and then propagate the message from b to c anywhere between 60 and 69, thereby implying a duration of at least 12 from a to c . However, faster journeys of duration 2 could exist earlier from a to c . This observation is crucial to formulate the problem of fastest broadcast. Attempting to reach each node using fastest journeys may be relevant in the case of point-to-point communication (and this objective was the one considered in Bui-Xuan et al. (2003)); but it is clearly less relevant in a context of *broadcast* for the aforementioned reason, since this would imply a same message is sent several times over a same edge (e.g., over (bc) , once as part of the fastest journey to c , another time as part of the fastest journey to d).

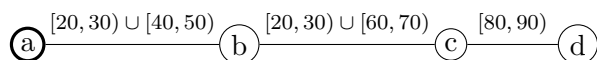


Figure 16: A simple TVG where fastest broadcast does not match fastest journeys (edge latency $\zeta = 1$). Example from (Casteigts et al. 2012b).

Casteigts et al. (2012b) address the problem of minimizing the *overall duration* of the broadcast, that is, the time elapsed between the first message emission and the last message reception in the whole network. As such, a *fastest broadcast tree* (or fastest BT) corresponds to a union of journeys which may or may not be individually fastest.

A link with temporal eccentricity: The main subproblem becomes determining *when* the emitter has the potential to reach all nodes the fastest, i.e., when the “longest” foremost journey from the emitter to any other node (also called its *temporal eccentricity*) is minimum. Note that learning minimum temporal eccentricities in distributed networks is an interesting problem in its own right, and may be used in other tasks than broadcasting (e.g., electing leader nodes based on their ability to reach other nodes quickly).

Once the emitter knows its own time of minimum eccentricity, the fastest broadcast reduces to performing a *foremost* broadcast at this particular date (by definition, this is optimal). This foremost broadcast can be done using the output of the foremost BT algorithm in previous section, or by building a single foremost BT for the selected date (for a single date, foremost BTs can be built by means of flooding whereby all the nodes record which of their neighbour gave them the message first, followed by local acknowledgments of these relations).

In both cases, it is not the broadcast itself that causes us difficulty, we thus describe the mechanisms by which a node can learn its temporal eccentricity in a periodic TVG using T-CLOCKS.

The temporal eccentricity (or simply eccentricity below) of a node u at date t is formally defined as

$$ecc_u(t) = \max\{\hat{d}_{u,t}(v) : v \in V\}, \quad (1)$$

that is, the maximum among all temporal distances from u to any node at time t (see Section 2 for definitions).

Temporal distance vs. temporal views: There is a strong connection between temporal distances and temporal views. Both actually refer to the same quantity seen from different perspectives: the temporal distance is a duration defined locally to an emitter at an emission date, while the temporal view is a date defined locally to a receiver at a reception date. In fact, we have

$$\hat{d}_{u,t_e}(v) = t_r - \phi_{v,t_r}(u) \quad (2)$$

where t_e is an emission date, and t_r is the corresponding earliest reception date.

Principle of the algorithm: The algorithm in (Casteigts et al. 2012b) consists in inferring (and recording) temporal distances at every node relative to a given emitter based on the evolution of temporal views monitored through T-CLOCKS. The inference is made as per the equivalence relation of Equation 2. Precisely, for a given emitter u , every node $v \neq u$ infers $\hat{d}_u(v)$ from $\phi_v(u)$ over one period and records it in a *distance table*. Since we deal with continuous-time and possibly overlapping contacts, this information is recorded as a set of *intervals* that correspond to the different phases of evolution of the distance (discrete or continuous) as illustrated in Figure 17. The aggregation operation is illustrated in Figure 18.

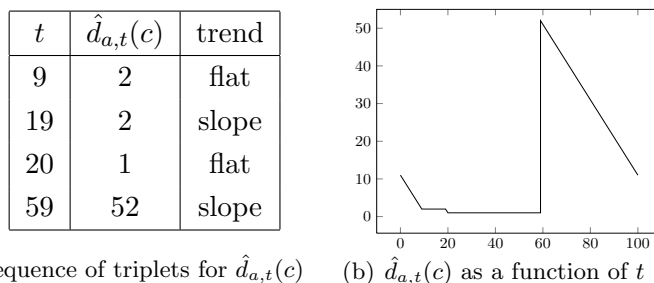


Figure 17: Temporal distance from a to c , as a function of emission date. Example from (Casteigts et al. 2012b).

After the distance tables have been computed with respect to some emitter u , they can be opportunistically aggregated along a tree rooted in u (the aggregation tree is arbitrary). Aggregation of a children table consists of a segment-wise *maximum* against the local distance table (segments are artificially split, if needed, to be aligned with each other).

Finally, once the emitter has aggregated the table of its last child, the final result corresponds to its eccentricity over time (by Equation 1). Any of the minimum values can thus be

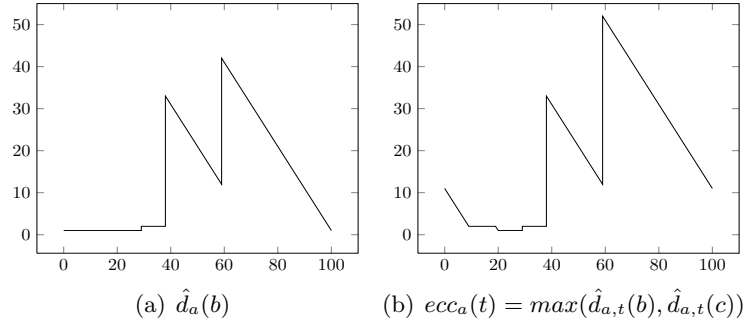


Figure 18: Aggregation of distances. The left curve (distance from a to b) is combined to the curve of Figure 17(b) (distance from a to c) in order to yield the eccentricity of node a over one period. Example from (Casteigts et al. 2012b).

selected and used as initiation date for the broadcast.

4.2 Raising the level of abstraction

Distributed algorithms can be expressed using a variety of communication models (e.g., message passing, mailboxes, shared memory). Although a vast majority of algorithms are designed using one of these models—the message passing model—the very fact that one of them is chosen implies that the obtained results (e.g., positive or negative characterizations and associated proofs) are somewhat limited by the scope of that model. This problem of diversity among models, already pointed out twenty years ago in Lynch (1989), led researchers to consider higher abstraction levels when studying fundamental properties of distributed systems. In a previous report (Casteigts and Flocchini 2013), we presented two related tools: *graph relabelling systems* and *population protocols*. We recall here the main definitions, some of which are used in Section 5.

4.2.1 Graph Relabelling Systems

Graph relabelling systems were proposed by Litovsky et al. (1999) as a mean to represent distributed algorithms at a abstract level and local scale. This formalism allows one to represent an algorithm as a set of local interaction rules that modify atomically the state of small connected subsets of nodes—typically a pair or a star of neighbouring nodes. Each rule is encoded as a couple (*preconditions*, *actions*), where *preconditions* describes the pattern of state that triggers the rule, and *actions* indicates to new states the corresponding nodes take on as a result.

Figure 19 shows a basic example of algorithm made up of a single rule, $\overset{I}{\bullet} \overset{0}{\bullet} N \longrightarrow \overset{I}{\bullet} \overset{1}{\bullet} I$, which builds a spanning tree over an arbitrary graph. Concretely, the effect of the rule is to extend the existing tree by including new connected nodes (labelled N) and the corresponding edges. Initially, all the nodes are labelled N except one distinguished node, the root of the tree, labelled I . All the edges are labelled 0 (they do not belong to the tree). Then, as the

rule is applied repeatedly, more nodes and edges are included, leading eventually to a valid spanning tree.

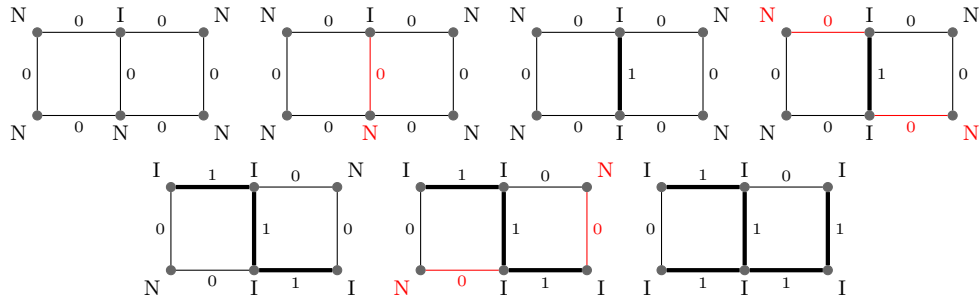


Figure 19: Spanning tree construction over a static graph, using graph relabellings. Example adapted from (Litovsky et al. 1999).

Several relabelling steps can occur simultaneously, as long as they involve disjoint sets of nodes. The order in which the interactions take place is not specified by the algorithm; it can be regarded as an implementation choice or even an external component, for example, controlled by an adversary. The various models of computation (e.g., pairwise, starwise) are however sometimes associated with an underlying synchronization procedure that can be implemented in the message passing model.

4.2.2 Population protocols

Another approach that shares the objective of abstracting the communication model is that of *population protocols* (Angluin et al. 2006). Population protocols can be seen as a particular instance of graph relabelling systems, which involves pairwise interaction and assumes particular properties on the underlying synchronization (also called an interaction scheduler). In particular, the original paper on population protocols assumes that every pair of nodes interact infinitely often—the graph of interaction is complete. The brilliant idea behind population protocols is to consider the interaction scheduler as representative of the connectivity induced by nodes’ movements in a mobile network, and thus make it a model for interaction in dynamic networks (however realistic the infinitely frequent interactions may be).

Research around population protocols has mainly focused on examining variations of the initial model and characterizing, for each variant, the type of predicates (formulas of logic related to the nodes states) that could be possibly computed. For instance, the initial model is shown to compute the class of *semilinear* predicates, that is, all predicates definable by first order Presburger arithmetics (Ginsburg and Spanier 1966). The addition of various features allows one to increase the power of such a distributed machine. For instance, adding a constant number of bits of memory on the edges makes it equivalent to a non-deterministic Turing Machine with $O(n^2)$ space (Chatzigiannakis et al. 2010) Variations can also concern properties of the underlying scheduler, for example, related to its fairness or self-evolution

(Chatzigiannakis et al. 2009); however, the fact that nodes interact infinitely often seems not have been questioned in these works.

4.2.3 Relevance in a dynamic environment

Whether in the case of graph relabellings or population protocols, the abstraction offered by these tools allows for general results to be characterized. In particular, negative results in these models imply impossibility in all concrete models, such as shared memory, mailboxes, or message passing (and as a result of equivalence, the mobile agent paradigm as well). By distancing the analysis from low-level concerns, it also allows one to focus on more fundamental aspects like the connectivity or the symmetries of a graph and their impact on the feasibility of problems.

The need for this type of abstraction is all the more relevant in dynamic networks, where the evolution of the topology adds yet another level of complexity to the analysis. Of particular interest is the question of how a given property on the dynamics impacts the feasibility or complexity of problems, regardless of concrete communications. Section 5 discusses in detail this question, using graph relabellings as the basic tool.

4.3 Using invariants

In the chaos of dynamic networks, it is always possible to design algorithms in such a way that they guarantee some global invariants over the whole execution. The counting algorithm proposed by Angluin et al. (2006) in the context of population protocols is one such example. In this algorithm, every node is initially a counter that has counted itself. Then, as the nodes interact, the counters opportunistically merge by pairs. As the counters keep merging, their tally increases and their number decreases, leading eventually to a single counter that has the right tally. A possible application of this counting principle is given, consisting of monitoring a flock of birds for fever, with the role of *counters* being played by sensors in each bird and the bird assume to interact pairwise infinitely often.

The correctness of the algorithm is based on the following invariant. Let \mathcal{C} (resp. \mathcal{F}) be the set of counter nodes (resp. counted nodes). Then $|\mathcal{C}| + |\mathcal{F}| = |V|$ holds at any time of the computation. It follows that if $|\mathcal{C}| = 1$ then the corresponding node has its counter value equal to $|V|$. Under the assumption of a fixed number of nodes, this algorithm thus reaches the desired state when exactly one counter remains.

4.4 Using progression hypotheses

Informally speaking, a progression hypothesis is one that guarantees the execution of a algorithm keeps making some progress (whenever possible) towards reaching its goal. Progression hypotheses can be defined at several levels, e.g., on the topology, on the synchronization between nodes, or on the ordering of interactions.

An example of a progression hypothesis based on topological properties is given by O'Dell and Wattenhofer (2005) in the case of broadcast. The basic idea is to assume that, however

dynamic the network might be, it always remains connected in a static sense. In terms of evolving graphs that means that every G_i of the sequence is connected in the usual way. Given a process of broadcast, this guarantees that in each step, at least one informed node must have a non-informed node as neighbour, and thus some progress can be made on the broadcast. As a result, the duration of the overall process can be bounded by the number of nodes; at least, if we assume the nodes cannot fail to communicate with their neighbour.

In general, no topological property can, alone, guarantee that the nodes will effectively communicate and collaborate. This property is particularly true in finite-time dynamic graphs, where every edge appears only a limited number of times. In this case, guaranteeing the success of an algorithm requires additional assumptions about the existence of interactions. [Casteigts et al. \(2009a\)](#) propose a generic progression hypothesis for the application of pairwise interactions in evolving graphs in the context of graph relabelling systems. The hypothesis requires that, given an evolving graph $\mathcal{G} = \{G_1, G_2, \dots\}$ where every G_i covers some period $[t_i, t_{i+1})$, it is possible for each node to apply at least one relabelling rule with each of its neighbours in every period provided that the rule preconditions are *already* satisfied at time t_i and still satisfied at the time the rule is applied. As discussed in Section 5, hypotheses of this kind are required to characterize *sufficient* conditions in terms of dynamic graph properties.

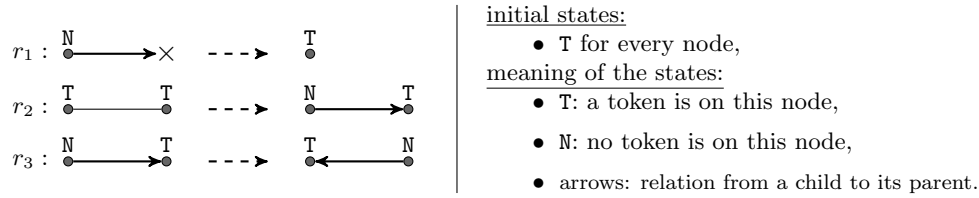
Things are a bit different when the edges are assumed to appear infinitely often. In this case, it make sense to consider progression hypotheses at a more abstract level. Several such hypotheses have been considered in the area of population protocols, based on various fairness assumptions on the interaction scheduler ([Angluin et al. 2006](#), [Chatzigiannakis et al. 2009](#)). Given the global state of the network, called a *configuration*, the basic fairness assumption states that if a configuration C_2 can be reached from another configuration C_1 , and C_1 is reached infinitely often, then C_2 must also be reached infinitely often. In other words, all recurrently reachable configurations must be reached eventually. Several variants of fairness have been proposed, including local fairness, global fairness, or k -bounded fairness. It is this type of assumption that guarantees, for instance, that the execution of the counting principle from Section 4.3 will eventually reach a single-counter configuration.

4.5 Topological events as a trigger for computation

In some cases, it may be relevant to execute dedicated operations when a topological event occurs. This technique is fundamental in dynamic networks whenever such changes should impact a solution being computed. The idea is very general and applies in a variety of models. For example, message passing algorithms can usually be specified in an event-based fashion using the three following types of events ([Santoro 2007](#)):

- spontaneous impulse (e.g., at initialization time)
- arrival of a message
- firing of a local trigger (e.g., timer or alarm)

Algorithm 1 Spanning forest algorithm based on coalescing and regenerating trees.



In the same vein, topological events can also be considered as another type of event. The T-CLOCKS algorithm we discussed in Section 4.1, which is a message-passing algorithm, is mostly made up of routines that execute when the appearance or the disappearance of an edge is detected, that is, its operations are encapsulated inside `onEdgeAppearance()` and `onEdgeDisappearance()` methods that are supposedly called by the system when these events occur.

Casteigts and Chaumette (2005) suggest applying the same principle in the case of graph relabelling systems by introducing special rules that a node can apply locally whenever adjacent edges appear or disappear. An example of an algorithm using this feature is shown in Algorithm 1. This algorithm was proposed in (Casteigts 2006, Casteigts et al. 2009b) to maintain a forest of spanning trees in a delay-tolerant network, which is the same problem we discussed in Section 3 (illustrated in Figure 12 on page 16). The main quality of this algorithm is that it does not assume the existence of stable periods in-between consecutive events; it can actually tolerate any number of simultaneous events without entering an inconsistent state. The solution is based on a perpetual alternation of *topology-induced* splits and *computation-induced* mergers of a forest of spanning trees based on the circulation of tokens that are none other than the roots themselves. Hence, each tree in the forest hosts exactly one token whose circulation is strictly confined to the edges of its own tree. When two tokens arrive at the endpoints of the same edge (rule r_2) both trees are merged on that edge and one of the two tokens is destroyed. If an edge disappears and it was part of a tree, the corresponding tree is broken into two pieces, one of which becomes token-free. The corresponding endpoint, because it lost the edge to its parent, knows instantly that it has become the highest node in its remaining tree, and it can thus generate a new token locally (rule r_1). As a result, both *mergers* and *splits* are purely localized, atomic phenomena. The rest of the time, the token circulates (rule r_3) in the hope of finding new merging opportunities. Consistency—here defined as the absence of cycles and presence of exactly one token per tree—is always guaranteed. Convergence towards a single tree per connected component is generally not expected due to the frequency of topological changes; however, it can be reached eventually if the graph remains static sufficiently long. How fast it converges remains an open question. A detailed example of execution is shown on Figure 20.

Note that, in general, the circulation of tokens is a powerful technique that is often used for voting, mutual exclusion, and leader election (Israeli and Jalfon 1990, Cooper et al. 2012).

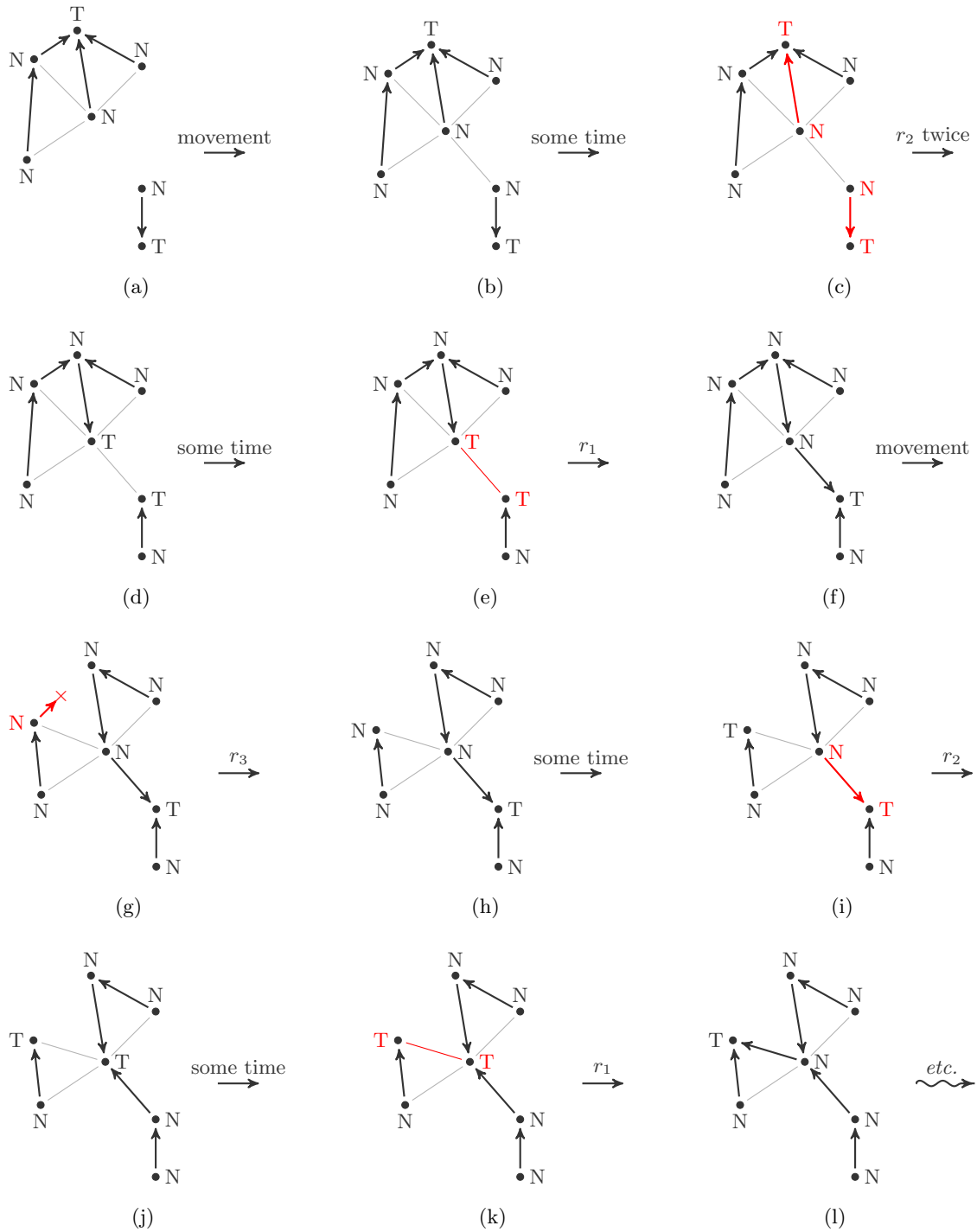


Figure 20: Example of a (possible) execution sequence for the spanning forest algorithm.

5 Topological conditions

Besides the complexity in time or in number of messages, a common approach for analyzing distributed algorithms is to look at the assumptions they require on the underlying network. We review here recent efforts to extend this type of approach to dynamic networks. In particular, we ask how a given property of the evolution of a network can be shown to be fundamentally necessary or sufficient to the success of an algorithm.

5.1 Example of graph conditions in static networks

Let us start with providing a few examples of such conditions in static networks, to give an intuition of what is a necessary graph condition. One of the most recurrent obstacles to the solvability of distributed problems is the presence of symmetries in a network. The simplest example occurs with the problem of election in a two-node network (see Section 3 for definition of the election problem). Consider the two-node graph on the left of Figure 21(a). There exists no *deterministic* distributed algorithm that can guarantee a successful election in this graph. Informally, both nodes could keep making the same decision simultaneously, and forever so. The reason is that this graph possesses symmetry: it is a sort of “multiple” of another simpler graph, the one-node loop graph (depicted on the right of the same figure). Thus, from a local point of view, a node has no mean to decide whether it is alone or has a neighbour that acts in the same way as it does (i.e., sends and receives the same messages). It is therefore impossible for it to decide whether to be a leader or not, since any such decision could be duplicated and lead to an inconsistency.

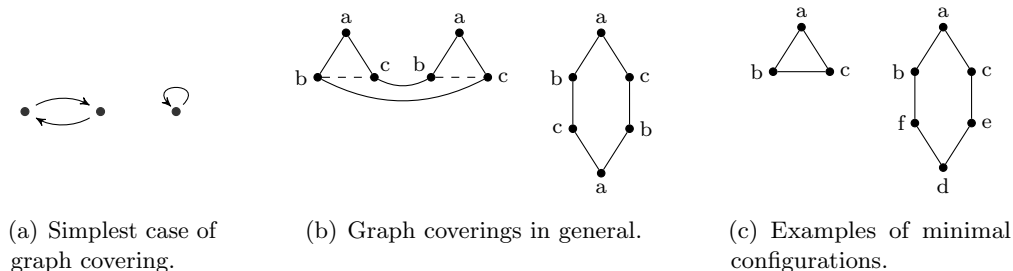


Figure 21: Example of basic conditions for the election problem in static networks: minimality in terms of graph coverings.

The symmetry problem is pointed out in several early works in distributed computing, and is examined closely by Angluin (1980). The graph-theoretic concept at stake here is that of *covering*. We say that the two-node graph on the left is a *2-covering* of the one-node loop graph on the right. It is today a well-known fact that election is only feasible in those graphs that are *minimal* in terms of covering (Yamashita and Kameda 1996, Mazurkiewicz 1997). This concept generalizes well to labelled graphs. Figure 21(b) gives an example of a generic construction rule to build a covering: take two (or more) identical graphs; remove the same edge from each; and re-connect them by switching the edges endpoints (circularly, if more than two). Finally, Figure 21(c) shows two examples of configurations in which

election is feasible. Note that in the case of election, minimality in terms of graph covering is both a *necessary* and a *sufficient* condition.

This example illustrates what might be a condition, in terms of a graph property, for a given distributed problem. There are many other examples in static graphs, e.g. *graph planarity* for the problem of face routing, or *bounded growth* for maximum independent set. The rest of this section is concerned with the transposition of this type of approach, and results, into the realm of dynamic networks.

5.2 Characterizing graph conditions in dynamic networks

As a trivial example, consider the broadcasting of a piece of information in the network depicted in Figure 22. The possibility of completing the broadcast in this scenario clearly depends on which node is the initial emitter: a and b may succeed, while c cannot. Why? How can we express this intuitive property that the topological evolution must have with respect to the emitter? Flattening the time dimension without keeping information about the ordering of events would obviously be removing important specifics, such as the fact that nodes a and c are in a non-symmetrical configuration.

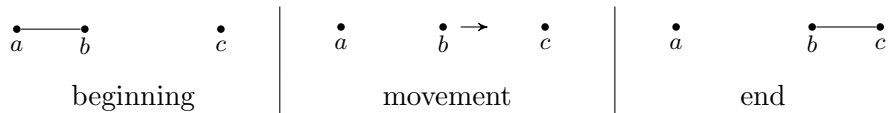


Figure 22: A basic scenario, where a node (b) moves during the execution.

In recent work, Casteigts et al. (2009a, 2012a) suggest combining the formalism of graph relabelling systems with that of evolving graphs in order to model computation in a dynamic network at a high level of abstraction. This combination makes it possible to express the fine-grained properties of the network’s dynamics (based on evolving graphs) and then examine what impact these properties have on the possible sequence of execution of an algorithm. The main advantage of graph relabellings is their high level of abstraction. Distributed algorithms are generally expressed using a particular communication models (e.g., message passing, mailboxes, or shared memory). But the very fact of choosing one implies that the obtained results (e.g., positive or negative characterizations and associated proofs) are somewhat limited to the scope of the model. This problem of diversity among formalisms and results, already pointed out twenty years ago by Lynch (1989), led researchers to consider higher abstractions when studying fundamental properties of distributed systems, among which graph relabelling systems were proposed by Litovsky et al. (1999).

We review here some important definitions, while referring the reader to our first report Casteigts and Flocchini (2013) for a more formal treatment. In a nutshell, an evolving graph is a sequence of static graphs $\mathcal{S}_G = \{G_1, G_2, \dots\}$ whose evolution represents that of a dynamic network. The scenario on Figure 22 corresponds to $\mathcal{S}_G = \{G_1 = \bullet \text{---} \bullet, G_2 = \bullet \dots, G_3 = \bullet \text{---} \bullet\}$. Evolving graphs can be associated with additional time information, such as a time index $\mathcal{S}_T = \{t_1, t_2, \dots\}$ that indicates the intervals of time corresponding to

each graph in the sequence (i.e., G_1 corresponds to the period $[t_1, t_2)$). Evolving graphs are usually given as the couple $\mathcal{G} = (G, \mathcal{S}_{\mathcal{G}})$ or triplet $\mathcal{G} = (G, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{T}})$, where G is the union of all graphs in $\mathcal{S}_{\mathcal{G}}$, also called the *underlying graph*. In our example, $G = \bullet \leftrightarrow \bullet$. In the following we consider evolving graphs in their triplet form $\mathcal{G} = (G, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{T}})$, which is more general since it can simulate the simpler version using $\mathcal{S}_{\mathcal{T}} = \{1, 2, \dots\}$.

Graph relabelling systems are discussed in Section 4 of this document and also with a bit more detail in our first report. They were introduced as a means to describe local computations in distributed networks by Litovsky et al. (1999). Using them, a distributed algorithm is represented as a set of local interaction rules that are independent from the effective communications. Within the formalism of graph relabellings, the network is represented by a labelled graph whose labels indicate the algorithmic state of the corresponding nodes and edges. An interaction rule is thus defined as a local transition pattern (*preconditions, actions*), where *preconditions* and *actions* relate to the values of these labels. In the broadcasting example, the algorithm could be made up of a single propagation rule, namely $I \bullet \xrightarrow{N} \bullet \xrightarrow{I} I$, with the following meaning: if a node that is informed (labelled I) interacts with a node that is not (labelled N), then it transmits to this node the information (i.e., the N -node is relabelled into I , while the I -node remains unchanged). Initially all the nodes but the emitter are labelled N . The opportunistic repetition of such a rule within the sequence of graph $\mathcal{S}_{\mathcal{G}}$ eventually completes the propagation if this sequence of graph satisfies some conditions. Which ones?

5.2.1 Combination of graph relabellings and evolving graphs

Given an evolving graph $\mathcal{G} = (G, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{T}})$ and a date $t_i \in \mathcal{S}_{\mathcal{T}}$, \mathbf{G}_i is the *labelled* graph representing the state of the network *just after* the topological event of date t_i occurred, and by $\mathbf{G}_{i[}$ the labelled graph representing the network state *just before* that event occurred. This concept can be written as

$$Event_{t_i}(\mathbf{G}_{i[}) = \mathbf{G}_i$$

A number of distributed operations may occur between two consecutive events. Hence, for a given algorithm \mathcal{A} and two consecutive dates $t_i, t_{i+1} \in \mathcal{S}_{\mathcal{T}}$, we denote by $\mathcal{R}_{\mathcal{A}_{[t_i, t_{i+1})}}$ one of the possible relabelling sequences induced by \mathcal{A} on the graph G_i during the period $[t_i, t_{i+1})$. (Note that there might be various such sequences depending on the type of synchronization considered between nodes. The level of abstraction here is more general.) This concept can be written as

$$\mathcal{R}_{\mathcal{A}_{[t_i, t_{i+1})}}(\mathbf{G}_i) = \mathbf{G}_{i+1[}$$

A complete execution sequence from t_0 to t_k is then represented by an alternated sequence of relabelling steps and topological events, which can be written as

$$X = \mathcal{R}_{\mathcal{A}_{[t_{k-1}, t_k)}} \circ Event_{t_{k-1}} \circ \dots \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1}, t_i)}} \circ \dots \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0, t_1)}}(\mathbf{G}_0)$$

This combination is illustrated on Figure 23. As mentioned above, the deterministic execution of a relabelling algorithm depends on the way nodes select each other to interact. Hence, we denote by $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$ the set of all possible execution sequences of an algorithm \mathcal{A} over an evolving graph \mathcal{G} .

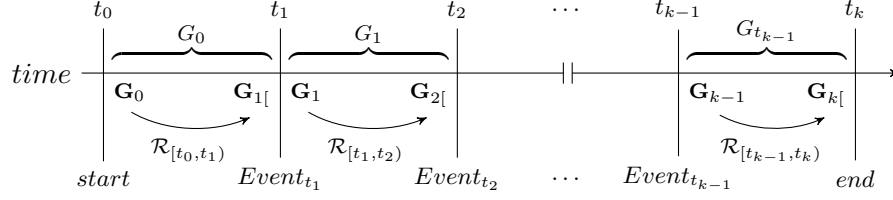


Figure 23: Combination of graph relabellings and evolving graphs. Picture from (Casteigts et al. 2009a).

This formulation of an execution allows us to define precisely the concepts of *topology-related* necessary or sufficient conditions. These concepts rely on a proper formulation of the objectives of an algorithm and what it means for an execution to be successful or not.

5.2.2 Necessary and sufficient conditions

Given an algorithm \mathcal{A} and a labelled graph \mathbf{G} , the state one wishes to reach by the end of the execution can be given by a logic formula \mathcal{P} on the labels of the nodes (or edges, if appropriate). For the example above (the single-rule propagation algorithm, let us call it \mathcal{A}_1), one such terminal state would be that all nodes are informed, i.e.,

$$\mathcal{P}_1(\mathbf{G}) = (\forall v \in V, \lambda(v) = I),$$

where λ is the labelling function. The objective $\mathcal{O}_{\mathcal{A}}$ is then defined as the fact of satisfying the desired property by the end of the execution, that is, on the final labelled graph \mathbf{G}_k . Here, we have $\mathcal{O}_{\mathcal{A}_1} = \mathcal{P}_1(\mathbf{G}_k)$.

Given an algorithm \mathcal{A} , its objective $\mathcal{O}_{\mathcal{A}}$ and an evolving graph property $\mathcal{C}_{\mathcal{N}}$, the property $\mathcal{C}_{\mathcal{N}}$ is a (*topology-related*) *necessary* condition for $\mathcal{O}_{\mathcal{A}}$ if and only if

$$\forall \mathcal{G}, \neg \mathcal{C}_{\mathcal{N}}(\mathcal{G}) \implies \neg \mathcal{O}_{\mathcal{A}}$$

Proving this result comes from proving that $\forall \mathcal{G}, \neg \mathcal{C}_{\mathcal{N}}(\mathcal{G}) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid \mathcal{P}(\mathbf{G}_k)$. (The desired state is not reachable by the end of the execution unless the condition is satisfied.) Symmetrically, an evolving graph property $\mathcal{C}_{\mathcal{S}}$ is a (*topology-related*) *sufficient* condition for \mathcal{A} if and only if

$$\forall \mathcal{G}, \mathcal{C}_{\mathcal{S}}(\mathcal{G}) \implies \mathcal{O}_{\mathcal{A}}$$

Proving this result comes from proving that $\forall \mathcal{G}, \mathcal{C}_{\mathcal{S}}(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{P}(\mathbf{G}_k)$.

Because the abstraction level of these computations is not concerned with the underlying synchronization, no topological property can guarantee, alone, that the nodes will effectively communicate and collaborate to reach the desired objective. Therefore, the characterization of *sufficient* conditions requires additional assumptions on the synchronization. Below is a generic progression hypothesis that enables it. This assumption may or may not be considered as realistic depending on the expected rate of topological changes.

Progression Hypothesis 1. (PH_1). *In every time interval $[t_i, t_{i+1})$, each node is able to apply at least one relabelling rule with each of its neighbours, provided the rule preconditions are already satisfied at time t_i (and still satisfied at the time the rule is applied).*

5.2.3 Example analyses

This analytical framework was illustrated in (Casteigts et al. 2009a, 2012a) by the analysis of three basic algorithms: the above propagation algorithm and two counting algorithms (one with a distinguished counter, the other one with coalescing counters). The main advantage of this framework is the ability to formalize the proofs to a large extent (and thereby open a door towards partial mechanization); however, for the sake of intuition, we review here their arguments in a natural language style, referring the reader to the above papers for complete proofs. Most of these arguments are very simple.

5.2.3.1 Propagation algorithm

Let us analyze the single-rule propagation algorithm \mathcal{A}_1 from above. The existence of a journey (*resp.* strict journey under PH_1 , see above) between the emitter and every other node is a necessary (*resp.* sufficient) condition to achieve $\mathcal{O}_{\mathcal{A}_1}$.

Condition 1. $\forall v \in V, \text{emitter} \rightsquigarrow v$

(There exists a journey between the emitter and every other node).

Lemma 1. *If a non-emitter node has the information at some point, this implies the existence of an incoming journey from a node that had the information before.*

Proof. If a non-emitter node has the information at some point, then it has necessarily applied the rule with another node. Thus, an edge existed at a previous date between this node and a node labelled I . By transitivity, this implies that a journey existed between a node labelled I and this node. \square

Theorem 1. *Condition 1 (\mathcal{C}_1) is a necessary condition on \mathcal{G} to allow algorithm \mathcal{A}_1 to reach its objective $\mathcal{O}_{\mathcal{A}_1}$.*

The proof follows from Lemma 1 and the initial states (I for the emitter, N for all other nodes), we have $\mathcal{O}_{\mathcal{A}_1} \implies \mathcal{C}_1$, and thus $\neg\mathcal{C}_1 \implies \neg\mathcal{O}_{\mathcal{A}_1}$.

The characterization of a sufficient condition below uses the concept of *strict journeys*. Given a graph $\mathcal{G} = (G, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{T}})$ and its sequence $\mathcal{S}_{\mathcal{G}} = \{G_1, G_2, \dots\}$, a strict journey is one such that at most one edge is crossed in each G_i . In other words, at most one edge is crossed in-between consecutive topological events. The existence of such a journey from a node u to a node v is noted $u \overset{st}{\rightsquigarrow} v$.

Condition 2. $\forall v \in V, \text{emitter} \overset{st}{\rightsquigarrow} v$

Theorem 2. *Under Progression Hypothesis 1 (PH_1 , defined in the previous section), Condition 2 (\mathcal{C}_2) is sufficient on \mathcal{G} to guarantee that algorithm \mathcal{A}_1 will reach $\mathcal{O}_{\mathcal{A}_1}$.*

Proof. By PH_1 , we have that for any graph $G_i = (V, E_i) \in \mathcal{S}_{\mathcal{G}}$, if a node u is labelled I at t_i then all its neighbours must be labelled I by t_{i+1} . By iteration of this statement over the sequence $\mathcal{S}_{\mathcal{G}}$, if a node u is initially labelled I , then all the nodes v such that $u \xrightarrow{st} v$ in \mathcal{G} will eventually be labelled I . Knowing that the initial label of the emitter is I , we can conclude that $\mathcal{C}_2(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{P}_1(\mathbf{G}_k)$ \square

5.2.3.2 Centralized counting algorithm

We next look at a centralized counting algorithm, \mathcal{A}_2 . The algorithm studied here is centralized in the sense that one node (the *counter* in charge of counting all the others) is distinguished from the other nodes at the beginning. The propagation algorithm seen above also uses a distinguished node, the emitter. The counter node has two labels (C, n) , meaning that it is the counter (C), and that it has already counted n nodes (initially 1, itself). The other nodes are labelled either F or N , depending on whether they have already been counted or not. The counting rule is then given as $\overset{C, n}{\bullet} \xrightarrow{N} \overset{C, n+1}{\bullet} \xrightarrow{F}$.

Under the assumption of a fixed number of nodes, the algorithm reaches a terminal state when all nodes are counted, which corresponds to the fact that no more nodes are labelled N :

$$\mathcal{P}_2 = (\forall v \in V, \lambda(v) \neq N)$$

The objective is thus to satisfy this property by the end of the execution ($\mathcal{O}_{\mathcal{A}_2} = \mathcal{P}_2(\mathbf{G}_k)$). There is a simple condition that is both necessary and sufficient: the existence of an edge at some point of the execution between the *counter* node and every other node. Proving this result is straightforward, so we omit the proofs.

Condition 3. $\forall v \in V \setminus \{\text{counter}\}, (counter, v) \in E$, where E is the set of edges of the underlying graph.

Theorem 3. Condition 3 (\mathcal{C}_3) is a necessary condition to allow algorithm \mathcal{A}_2 to reach its objective $\mathcal{O}_{\mathcal{A}_2}$.

Theorem 4. Under PH_1 , \mathcal{C}_3 is also a sufficient condition to guarantee that algorithm \mathcal{A}_2 will reach its objective $\mathcal{O}_{\mathcal{A}_2}$.

5.2.3.3 Decentralized counting algorithm

In contrast to the previous algorithm, the next algorithm, \mathcal{A}_3 does not require a distinguished initial state for a node. Indeed, all nodes are initialized with the same labels $(C, 1)$, meaning that they are all initially counters that have already included themselves into the count. Then, depending on the topological evolutions, the counters opportunistically merge by pairs, that is, $\overset{C, i}{\bullet} \xrightarrow{C, j} \overset{C, i+j}{\bullet} \xrightarrow{F}$. The same counting principle is discussed in Section 4.3 and proposed by [Angluin et al. \(2006\)](#). Note that this algorithm can also serve as an election algorithm based on the same coalescing principle. In the optimistic scenario, at the end of the execution, only one node remains labelled C and its second label gives the total number

of nodes in the graph. We are interested here in the topological conditions that enable (resp. guarantee) a successful execution of this algorithm.

The correctness of this algorithm is based on the following invariant (also mentioned in Section 4.3). Let \mathcal{C} (resp. \mathcal{F}) be the set of nodes in state C (resp. F). Then $|\mathcal{C}| + |\mathcal{F}| = |V|$ holds at any time of the computation. It follows that if $|\mathcal{C}| = 1$ then the corresponding node has its counter value equal to $|V|$. Under the assumption of a fixed number of nodes, this algorithm thus reaches the desired state when exactly one node remains labelled C :

$$\mathcal{P}_3 = (|\mathcal{C}| = 1).$$

As with the two previous algorithms, the objective here is to satisfy this property by the end of the execution: $\mathcal{O}_{\mathcal{A}_3} = \mathcal{P}_3(\mathbf{G}_k)$. The characterization below proves that the existence of a node belonging to the *horizon* of every other node is a necessary condition for this algorithm.

Condition 4. $\exists v \in V : \forall u \in V, u \rightsquigarrow v$.

(At least one node is reachable from all the others by a journey.)

Lemma 2. $\forall u \in V, \exists u' \in V : u \rightsquigarrow u' \wedge \lambda_{t_k}(u') = C$

(Counters cannot disappear from their own horizon.)

Proof. By contradiction. The only operation that can suppress C labels is the application of the counting rule. Since all nodes are initially labelled C , assuming that Lemma 2 is false (i.e., that there is no C -labelled node in the horizon of a node) comes to assume that a relabelling sequence took place transitively from node u to a node u' that is outside the horizon of u , which is by definition impossible. \square

Theorem 5. Condition 4 (\mathcal{C}_4) is necessary for algorithm \mathcal{A}_3 to reach its objective $\mathcal{O}_{\mathcal{A}_3}$.

Proof. $\neg\mathcal{C}_4(\mathcal{G})$ implies that none of the nodes can be reached by all others. Thus, given any potential final counter, at least one other node could not have reached it through a journey. By Lemma 2, this implies that at least two final counters remain at the end of the execution ($|\mathcal{C}| \geq 2$ in \mathbf{G}_k). \square

The characterization of a sufficient condition for algorithm \mathcal{A}_3 is left open by Casteigts et al. (2009a). De facto, the original assumption of population protocols, namely that every pair of node interacts infinitely often, would form a sufficient condition for this algorithm to converge in finite time; however, this time is *unbounded* and thus does not offer a guarantee of success in any finite instance of evolving graph. Searching for properties that could be satisfied on a given finite-time evolving graph is particularly appealing in terms of applications, since it would allow to check if that property holds on a given network trace (and thus, guarantee that the algorithm would have worked with certainty in the corresponding scenario).

Such a condition was characterized by Marchand de Kerchove and Guinand (2012). The condition is that every pair of nodes shares an edge *at least once* during the execution. In other words, given an evolving graph $\mathcal{G} = (G, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{T}})$, its underlying graph G is complete

(be careful not to mistake the concepts of underlying graph of an evolving graph, with that of graph of interaction in population protocols; the edges of the latter are required to appear infinitely often, while those of the former might exist once once).

Condition 5. $\forall u, v \in V, (u, v) \in E$.

(Every pair of node shares an edge at least once during the execution. Or equivalently, the underlying graph G of the evolving graph \mathcal{G} is complete.)

Here is a sketch of the arguments from Marchand de Kerchove and Guinand (2012).

Lemma 3. *C-nodes remain C-nodes until they become F-nodes.*

Lemma 4. *F-nodes remain forever F-nodes.*

Lemma 5. *If two C-nodes share an edge in a given G_i , only one remains by t_{i+1} .*

Theorem 6. *Condition 5 (\mathcal{C}_5) is sufficient under PH_1 for algorithm \mathcal{A}_3 to reach its objective $\mathcal{O}_{\mathcal{A}_3}$.*

Proof. By contradiction. Assume the algorithm fails, i.e., there is at least two C -nodes at the end of the execution. By \mathcal{C}_5 , they must have shared an edge in some G_i . By Lemma 3 their labels were already C , and thus (by Lemma 5) one of them must have disappeared. But Lemma 4 tells us this is impossible. \square

5.2.4 Tightness of conditions

Given a necessary or a sufficient condition \mathcal{C} relative to an algorithm \mathcal{A} , one important question is whether \mathcal{C} is optimal for \mathcal{A} or a better condition could exist. Marchand de Kerchove and Guinand (2012) defines a criterion of *tightness* for necessary and sufficient conditions. The motivation is to avoid basic conditions that tell us nothing or little about the problem at stake. Examples of trivial necessary conditions include $E \neq \emptyset$; examples of excessive sufficient conditions include, for many algorithms, the fact that every pair of nodes interacts infinitely often—this condition would have been sufficient for all algorithms we discussed so far.

Keep in mind that given an algorithm \mathcal{A} , a necessary condition \mathcal{C}_N is one such that $\neg \mathcal{C}_N(\mathcal{G}) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid \mathcal{O}_{\mathcal{A}}$, and a sufficient condition \mathcal{C}_S is one such that $\mathcal{C}_S(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{O}_{\mathcal{A}}$. Now, a necessary condition \mathcal{C}_N is said to be *tight* if and only if

$$\mathcal{C}_N(\mathcal{G}) \implies \exists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid \mathcal{O}_{\mathcal{A}}$$

Symmetrically, a *tight* sufficient condition \mathcal{C}_S is a sufficient condition such that

$$\neg \mathcal{C}_S(\mathcal{G}) \implies \exists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \neg \mathcal{O}_{\mathcal{A}}$$

What does it tell us? In technical terms, it tells us that the set of dynamic graphs that satisfies a tight condition is *maximal* for the considered algorithm. For necessary conditions,

having a dynamic graph *not* in this set implies failure, while having a graph in this set implies a *possibility* of success (depending on how interaction takes place between the nodes). For sufficient conditions, having a graph in the set implies success, while having it outside implies a *possibility* of failure (depending on how interaction takes place between the nodes). These observations are summarized by a diagram on Figure 24.

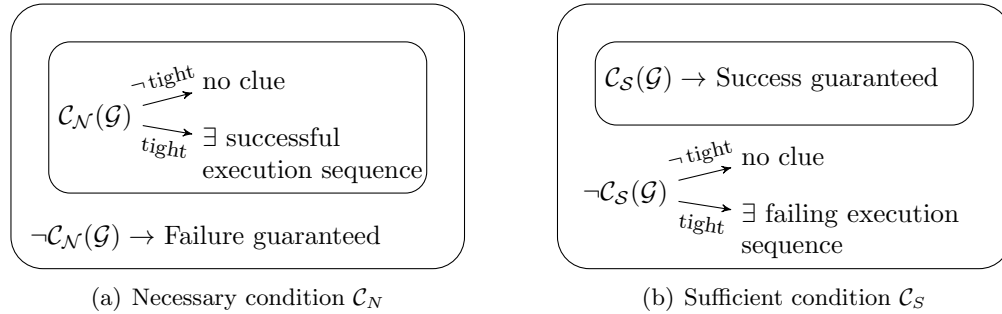


Figure 24: Logical implications of tight conditions.

All of the conditions characterized in this section happen to be tight, including the sufficient condition by Marchand de Kerchove and Guinand (2012) for decentralized counting.

5.2.5 A note on maintenance algorithms

We discuss in Section 3 the various ways a problem could be reframed in a dynamic context. In particular, we can distinguish between two fundamental types of objectives. In the examples above, we consider algorithms whose objective is to satisfy a given property *by the end* of the execution. Another way is to consider the *maintenance* of a desired property despite the evolution of the network (i.e., what we referred to as the *evolving* variant of a problem in Section 3). In this case, the objective should not be expressed in terms of a terminal state, but rather in terms of a recurrent state one wants to see satisfied, say, in-between every two consecutive topological events, i.e., $\mathcal{O}_A = \forall G_i \in S_G, \mathcal{P}(G_{i+1})$.

6 Computational relationship between classes of dynamic graphs

We discussed in Section 5 how the intimate relationship between dynamic properties and the feasibility of problems can be studied in the most general terms. This section explores another facet of the same question: What is the computational relationship between a set of dynamic graph classes? Unlike the conditions studied in Section 5 whose characterizing property was of a finite essence (e.g. existence of at least one journey from a node to all others), the classes we consider here are defined by recurrence properties on an infinite-time network schedule. (Note that all of these properties/classes, whether of a finite or infinite essence, will be connected within the same hierarchy in Section 7). We study the categorization of dynamic graphs into classes using problems already discussed in Sections 3 and 4.1, namely foremost, shortest, and fastest broadcast. As a result, the work presented here offers in a sense a good summary of the analytical approach we advocated throughout the report. Most of these results can be found in Casteigts et al. (2010a, 2012b,c).

The three classes: These are three subclasses of delay-tolerant networks (DTNs)—those networks in which instant connectivity is never guaranteed, but connectivity can still be achieved over time and space. Precisely, the classes are:

- The class \mathcal{R} of all graphs whose edges re-appear infinitely often (*recurrent edges*). That is, if an edge exists at some time then it cannot disappear forever and must eventually re-appear at some unknown (but finite) date in the future, repeatedly. The *underlying graph* is not required to be complete (i.e., not all possible pairs of nodes must interact), but in order to guarantee temporal connectivity it must be *connected*.
- The class \mathcal{B} (for *bounded-recurrent edges*) which consists of those graphs with recurrent edges in which the recurrence time cannot exceed a given duration Δ , and, again, the underlying graph is connected.
- The class \mathcal{P} (for *periodic edges*) which consists of those graphs in which all topological events (appearance or disappearance) repeat identically modulo some period p , and, again, the underlying graph is connected.

As far as *inclusion* is concerned, it clearly holds that $\mathcal{P} \subset \mathcal{B} \subset \mathcal{R}$, but what about the *computational relationship* between these classes? Considering different types of knowledge, namely the number n of nodes in the network, a bound Δ on the recurrence time, and (any multiple of) the period p , the authors looked at the relationship between $\mathcal{P}(\mathcal{R}_n)$, $\mathcal{P}(\mathcal{B}_\Delta)$, and $\mathcal{P}(\mathcal{P}_p)$, where $\mathcal{P}(\mathcal{C}_k)$ is the set of problems one can solve in class \mathcal{C} with knowledge k .

The three problems: The investigation is carried out by studying three variants of broadcast with termination detection at the emitter (TDB) in highly dynamic networks. These variants are those discussed in Section 3: TDB[*foremost*], in which the date of delivery is minimized at every node; TDB[*shortest*], where the number of hops used by the broadcast is minimized relative to every node; and TDB[*fastest*], where the overall duration of the

broadcast is minimized (however late the departure is). As already mentioned, these three metrics were introduced in Bui-Xuan et al. (2003) as part of a centralized, offline problem.

The main questions at stake are how the feasibility and reusability (and to some extent, complexity) of these problems vary in \mathcal{R} , \mathcal{B} , or \mathcal{P} with knowledge \emptyset , n , Δ , or p .

6.1 Summary of the results

We present a short summary of these results, which are then described in detail in Sections 6.2 to 6.5.

Feasibility: It is first shown that none of the three problems are solvable, in any of the classes, unless additional knowledge is considered. A constructive proof then shows that knowing n makes it possible to solve TDB[*foremost*] in \mathcal{R} ; however, it is not sufficient to solve TDB[*shortest*] nor TDB[*fastest*], even in \mathcal{B} . TDB[*shortest*] becomes in turn feasible in \mathcal{B} if Δ is known, but this context is not sufficient to solve TDB[*fastest*]; this later problem being solvable in \mathcal{P} knowing p . Put together, these results allow to show that

$$\mathcal{P}(\mathcal{R}_n) \subsetneq \mathcal{P}(\mathcal{B}_\Delta) \subsetneq \mathcal{P}(\mathcal{P}_p) \quad (3)$$

that is, the computational relationships between these three contexts form a *strict* hierarchy. This hierarchy implies in turn that a partial order \preceq_f exists on the *feasibility* of the three problems, such that

$$\text{TDB[foremost]} \preceq_f \text{TDB[shortest]} \preceq_f \text{TDB[fastest]} \quad (4)$$

Reusability: Regarding the possibility of reusing a solution, that is, the same broadcast tree over several broadcasts, the authors find that reusability in TDB[*shortest*] is easier than that of TDB[*foremost*]. Precisely, when TDB[*shortest*] becomes feasible in \mathcal{B} , it also enables reusability of the broadcast trees, whereas TDB[*foremost*], although it is already feasible in \mathcal{R} , does not enable reusability until in \mathcal{P} . This result is somehow surprising, as it suggests a different order \preceq_r on the *reusability* of these problems, such that

$$\text{TDB[shortest]} \preceq_r \text{TDB[foremost]} \quad (5)$$

Whether reusability is more or less difficult in TDB[*fastest*] than in TDB[*foremost*] is an open question, both of them being impossible in \mathcal{B}_Δ but possible in \mathcal{P}_p .

These results on feasibility and reusability are summarized in Table 3.

Complexity: Although complexity was not the main focus in Casteigts et al. (2012c), they characterized the time complexity and message complexity of their algorithms and observe some noticeable facts. For instance, the message complexity of the algorithm for TDB[*foremost*] is lower knowing Δ than knowing n , and even lower if both are known. Complexity results are summarized in Table 4. Note that TDB involves two processes: the actual dissemination of *information messages*, and the exchange of typically smaller *control messages* (e.g., for termination detection), both of which are separately analyzed.

Metric	Class	Knowledge	Feasibility	Reusability	Result from
Foremost	\mathcal{R}	\emptyset	no	–	} Casteigts et al. (2012c) Casteigts et al. (2012b)
	\mathcal{R}	n	yes	no	
	\mathcal{B}	Δ	yes	no	
	\mathcal{P}	p	yes	yes	
Shortest	\mathcal{R}	\emptyset	no	–	} Casteigts et al. (2012c)
	\mathcal{R}	n	no	–	
	\mathcal{B}	Δ	yes	yes	
	\mathcal{P}	p	yes	yes	
Fastest	\mathcal{R}	\emptyset	no	–	} Casteigts et al. (2012c) Casteigts et al. (2012b)
	\mathcal{R}	n	no	–	
	\mathcal{B}	Δ	yes	no	
	\mathcal{P}	p	yes	yes	

Table 3: 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{R}	n	unbounded	$O(m)$	$O(n^2)$	$O(m)$	$O(n)$
	\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 either of {	\mathcal{B}	Δ	$O(n\Delta)$	$O(m)$	$O(n) : 2n - 2$	$O(n)$	0
		$n\&\Delta$	$O(n\Delta)$	$O(m)$	$O(n) : n - 1$	$O(n)$	0
		$n\&\Delta$	$O(n\Delta)$	$O(m)$	0	$O(m)$	0

Table 4: Complexity of TDB for different classes of dynamic networks and associated knowledge.

6.2 Basic results and limitations

Observe first a general property of the computational relationship between the main three contexts of interest, namely knowing n in \mathcal{R} (noted \mathcal{R}_n), knowing Δ in \mathcal{B} (noted \mathcal{B}_Δ), and knowing p in \mathcal{P} (noted \mathcal{P}_p). These inclusions are shown to be strict in Section 6.5, based on the results we mention over the next subsections.

Theorem 7. $\mathcal{P}(\mathcal{R}_n) \subseteq \mathcal{P}(\mathcal{B}_\Delta) \subseteq \mathcal{P}(\mathcal{P}_p)$

Proof. The right inclusion is straight from the fact that $\mathcal{B} \subseteq \mathcal{P}$ and p is a valid bound Δ on the recurrence time. The left inclusion follows from the facts that $\mathcal{R} \subseteq \mathcal{B}$ and n can be inferred in \mathcal{B} if Δ is already known. This calculation can be done by performing, from any node (say u), a depth-first token circulation that will explore the *underlying graph* G over time. Having a bounded recurrence time indeed allows every node to learn the list of its neighbours in G within Δ time (all incident edges must appear within this duration). As the token is circulated to unvisited nodes, these nodes are marked as visited by u 's token and the token is incremented. Upon returning to u , the token value is n . \square

Here is now a negative result that justifies the need for additional knowledge in order to solve TDB in any of the considered contexts. Thus, we have:

Theorem 8. TDB *cannot be solved in \mathcal{P} without additional knowledge.*

Proof. By contradiction, let \mathcal{A} be an algorithm that solves TDB in \mathcal{P} . Consider an arbitrary $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{P}$ and $x \in V$. Execute \mathcal{A} in \mathcal{G} starting at time t_0 with x as the source. Let t_f be the time when the source terminates (and thus all nodes have received the information). Let $\mathcal{G}' = (V', E', \mathcal{T}', \rho') \in \mathcal{P}$ such that $V' = V \cup \{v\}$, $E' = E \cup \{(u, v) \text{ for some } u \in V\}$, for all $t_0 \leq t < t_f$, $\rho'(e, t) = \rho(e, t)$ for all $e \in E$ and $\rho'((u, v), t) = 0$. Now, consider $\rho'((u, v), t) = 1$ for some $t > t_f$, and the period of \mathcal{G}' is some $p' > t - t_0$. Consider the execution of \mathcal{A} in \mathcal{G}' starting at time t_0 with x as the source. Since (u, v) does not appear from t_0 to t_f , the execution of \mathcal{A} at every node in \mathcal{G}' is exactly the same as at each corresponding node in \mathcal{G} . In particular, node x enters a terminal state at time t_f with node v not having received the information, contradicting the correctness of \mathcal{A} . \square

Inclusion of \mathcal{P} in \mathcal{B} (and thus \mathcal{R}) yields the following corollary.

Corollary 9. TDB *cannot be solved in \mathcal{B} nor \mathcal{R} without any additional knowledge.*

Hence, it is proven by Casteigts et al. (2012c) that additional knowledge of some kind is required to solve TDB in these classes. They subsequently consider three types of knowledge, namely, the number of nodes $n = |V|$, an upper bound Δ on the recurrence time (when in \mathcal{B}), or the period p (in \mathcal{P}).

Impossibility of fastest broadcast: A general impossibility result can be established for TDB[*fastest*] in \mathcal{B} (and *a fortiori* in \mathcal{R}), this problem being unsolvable even if both n and Δ are known.

Theorem 10. $\text{TDB}[\textit{fastest}]$ is not solvable in \mathcal{B} (and a fortiori in \mathcal{R}), if only n and Δ are known.

Proof. The argument relates to the very existence of fastest journeys in an unstructured infinite setting. Consider for example the graph $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{B}$ such that $V = \{v_1, v_2, v_3\}$, $E = \{e_1 = (v_1, v_2), e_2 = (v_2, v_3)\}$ and ρ is such that:

- $\forall t \in \mathcal{T}, \forall i \in \mathbb{N}, \rho(e_1, t) = 1 \iff i\Delta \leq t < i\Delta + \zeta$
- $\forall t \in \mathcal{T}, \forall i \in \mathbb{N}, \rho(e_2, t) = 1 \iff i\Delta + (i+1)^{-1} \leq t < i\Delta + (i+1)^{-1} + \zeta$

for any $\Delta \geq 2\zeta + 1$. In such a setting, every period $[i\Delta, (i+1)\Delta)$ enables a journey \mathcal{J}_i from v_1 to v_3 such that $|J_i|_t = 2\zeta + (i+1)^{-1}$. As a result, there is an infinite sequence of journeys $\mathcal{J}_1, \mathcal{J}_2, ..$ such that $|J_{i+1}|_t < |J_i|_t$ for all $i \in \mathbb{N}$. \square

Theorem 11. $\text{TDB}[\textit{fastest}]$ is feasible in \mathcal{P} with a known period p , and the solution can be reused for subsequent broadcasts.

The algorithm for this result is described in Section 4 of this report. It relies on learning at what time or times in the period the *temporal eccentricity* of the emitter is minimum, then building a foremost broadcast tree for such a date or dates. Note that the broadcast tree so-built remains necessarily optimal in the future, since in \mathcal{P} the whole network schedule repeats forever. It can thus be *memorized* for subsequent broadcasts, i.e., the solution is *reusable*.

The next two sections (6.3 and 6.4) focus on $\text{TDB}[\textit{foremost}]$ and $\text{TDB}[\textit{shortest}]$ in \mathcal{R} and \mathcal{B} , knowing n and/or Δ . Section 6.5 then draws some conclusions on the variation of difficulty among these three problems, as well as on the computational relationship between $\mathcal{P}(\mathcal{R}_n)$, $\mathcal{P}(\mathcal{B}_\Delta)$, and $\mathcal{P}(\mathcal{P}_p)$.

6.3 $\text{TDB}[\textit{foremost}]$

$\text{TDB}[\textit{foremost}]$ in \mathcal{R} or \mathcal{B} clearly requires some sort of flooding because the very fact of probing a neighbour to determine if it already has the information compromises the possibility of sending it in a foremost fashion (in addition to risking the disappearance of the edge in-between the probing and the actual sending). As a consequence of Theorem 8, we know that this problem cannot be solved without additional knowledge. It becomes possible in \mathcal{R} if the number of nodes $n = |V|$ is known, as shown constructively by an algorithm whose termination is, however, not bounded in time. Being in \mathcal{B} with the same knowledge allows its termination to be bounded. Knowing Δ instead of n in \mathcal{B} then makes it possible to propose another solution that has a lower message complexity. This complexity can be further improved if both Δ and n are known, due to the possibility of terminating *implicitly*. Regarding *reusability* for the broadcast of the information, none of the broadcast trees built in \mathcal{R} or even \mathcal{B} turn out to be reusable as such, due to the inherent lack of structure of these classes.

Theorem 12. *Foremost broadcast trees are not reusable as such in \mathcal{B}_Δ (and a fortiori in \mathcal{R}_n).*

Proof. By contradiction, let a tree T be a reusable foremost tree with respect to emission date t . Since the *order* in which edges re-appear in \mathcal{B} is arbitrary (as long as they all occur within a Δ time window), an adversary can act on the schedule in such a way that the edges appear in a different order than that of the hierarchy of T , contradicting the fact that the tree is foremost. \square

Note that the proof argument does *not* relate to the non-existence of trees whose optimality repeat in the future; in fact, there must be at least one tree whose optimality holds infinitely often since there are finitely many possible trees and infinitely many time spans of duration Δ . The argument actually relates to the *non-decidability* of using a given tree. Nonetheless, the knowledge acquired can be helpful to lower the complexity of the termination detection.

6.3.1 TDB[foremost] in \mathcal{R}

This section only discusses the knowledge of n since Δ is not defined for \mathcal{R} . It can be shown constructively that the problem is solvable when n is known. The complete algorithm is in (Casteigts et al. 2012c); its informal description is as follows. Every time a *new* edge appears locally to an informed node, this node sends the information on the edge, and remembers the edge. The first time a node receives the information, it records the sender as parent, transmits the information on its available edges, and sends back a notification message to the parent. Note that these notifications create a parent-relation and thus a *converge-cast tree*. The notification messages are sent using a special primitive *send_retry* to ensure that the parent eventually receives it even if the edge disappears during the first attempt (message loss can be detected as a result of knowing the latency and detecting the disappearance of edges instantly). Each notification is individually propagated along the converge-cast tree using the *send_retry* primitive, and eventually collected at the emitter. Once the emitter has received $n - 1$ notifications, it knows all nodes are informed.

Theorem 13. *When n is known, TDB[foremost] can be solved in \mathcal{R} in unbounded time by exchanging $O(m)$ information messages and $O(n^2)$ control messages, where m is the number of edges.*

Proof. Since a node sends the information to each newly appearing edge, it is easy to see, by connectivity of the *underlying* graph, that all nodes will eventually receive the information. The dissemination itself is necessarily foremost because the information is either directly relayed on edges that are present, or sent as soon as a new edge appears. As for termination detection: every node identifies a unique parent and a converge-cast spanning tree directed towards the source is implicitly constructed; since every node notifies the source (through the tree) and the source knows the total number of nodes, termination is guaranteed. Since information messages might traverse every edge in both directions, and an edge cannot be traversed twice in the same direction, we have that the number of *information* messages is in the worst case $2m$. Since every node but the emitter induces a notification that is forwarded up the converge-cast tree to the emitter. The number of *notification* messages is the sum of distances in converge-cast tree between all nodes and the emitter, $\sum_{v \in V \setminus \{emitter\}} d_{h_tree}(v, emitter)$. The worst case is when the graph is a line where

we have $\frac{n^2-n}{2}$ control messages. Regarding time complexity, the termination of the algorithm is unbounded due to the fact that the recurrence of the edges is itself unbounded. \square

Reusability for subsequent broadcasts: As stated in Theorem 12, a foremost broadcast tree cannot be reused *as such* in subsequent broadcasts. It can however be reused as a converge-cast tree for the notification process where, instead of sending a notification as soon as a node is informed, each node notifies its parent in the converge-cast tree if and only if it is itself informed and has received a notification from each of its children. This would allow to reduce the number of control messages from $O(n^2)$ to $O(n)$, having only one notification per edge of the converge-cast tree.

6.3.2 TDB[foremost] in \mathcal{B}

If the recurrence time is bounded then either the knowledge of n or an upper bound Δ on the recurrence time can be used to solve the problem (with various message complexities).

Knowledge of n .

Since $\mathcal{B} \subseteq \mathcal{R}$, one can obviously solve TDB[foremost] in \mathcal{B} using the same algorithm as in \mathcal{R} (and the same observations apply regarding reusability of the converge-cast tree). Here, however, the termination time becomes bounded due to the fact that the recurrence of edges is itself bounded.

Theorem 14. *When n is known, TDB[foremost] can be solved in \mathcal{B} in $O(n\Delta)$ time by exchanging $O(m)$ information messages and $O(n^2)$ control messages.*

Proof. Since all edges in E are recurrent within any Δ time window, the delivery of the information to the last node must occur within $(n-1)\Delta$ global time. The same property holds for the latest notification, bounding the overall process to a duration of $\Delta(2n-2)$. The rest follows from Theorem 13. \square

Knowledge of Δ .

The information dissemination is performed as with the knowledge of n in \mathcal{R} , but termination detection can be achieved differently using Δ . Again, the full algorithm can be found in (Casteigts et al. 2012c); its informal description is as follows. Due to the time-bounded recurrence, no node can discover a new neighbour after a duration of Δ . Knowing Δ can thus be used by any node to determine whether it is a leaf in the broadcast tree (i.e., if it has not informed any other node within Δ time following its own reception time). This allows the leaves to terminate spontaneously while notifying their parents, which recursively terminate as they receive notifications from all their children.

Specifically, every time a *new* edge appears locally to an informed node, the node sends the information on that edge, and records it. The first time a node receives the information, it chooses the sender as its parent, memorizes the current time (in a variable *firstRD*),

transmits the information on its available edges, and returns an *affiliation* message to its parent using the *send_retry* primitive (starting to build the converge-cast tree). This affiliation message is not relayed upward in the tree, but is only intended to inform the direct parent about the existence of a new child (so this parent knows it must wait for a future notification by this node). If an informed node has not received any affiliation message after a duration of $\Delta + \zeta$, it sends a *notification* message to its parent using the *send_retry* primitive. The wait is bounded by $\Delta + \zeta$ for the following reasons (see also Figure 25). First, the *information* messages cannot be lost when they are sent on an appearing edge, neither can their potential *affiliation* answer (this work assumed that edge appear for a minimum of $2 \cdot \zeta$ time). Thus, the loss of information messages can only occur when the information is directly relayed by a node that has received it (say, as in Figure 25, node *a* relaying at time *firstRD* the information to node *b*). If the information message is lost then it simply means that this edge at that time did not have to be used. On the other hand, if the *affiliation* message is lost, it must be sent again. However, in the worst case, the common edge disappears just before the affiliation message is delivered, and reappears only $\Delta - 2 \cdot \zeta$ later. Affiliation messages can thus be received until *firstRD* + $\Delta + \zeta$.

If a node has one or more children, it waits until it receives a notification message from each of them and then notifies its parent in the converge-cast tree (using *send_retry* again). Once the emitter has received a notification from each of its children, it knows that all nodes are informed.

Theorem 15. *When Δ is known, TDB[foremost] can be solved in \mathcal{B} in $O(n\Delta)$ time by exchanging $O(m)$ information messages and $O(n)$ control messages.*

Proof. Correctness follows the same lines of the proof of Theorem 13. However the correct construction of a converge-cast spanning tree is guaranteed by the knowledge of Δ (i.e., the nodes of the tree that are leaves detect their status because no new edges appear within Δ time) and the notification starts from the leaves and is aggregated before reaching the source. The number of information messages is $O(m)$ as the exchange of information messages is the same as in \mathcal{R} knowing n , but the number of notification and affiliation messages drops to $2(n - 1)$. Every node but the emitter sends a single affiliation message; as for the notification messages, instead of sending a notification as soon as it is informed, each node notifies its parent in the converge-cast tree if and only if it has received a notification from each of its children resulting in one notification message per edge of the tree. The time complexity of the dissemination itself is the same as for the foremost broadcast when n is known. The time required for the emitter to subsequently detect termination is an additional $\Delta + \zeta + \Delta(n - 1)$ (the value $\Delta + \zeta$ corresponds to the time needed by the last informed node to detect that it is a leaf, and $\Delta(n - 1)$ corresponds to the worst case of the notification process, a line graph with the emitter on one end and the last informed node on the other). \square

Reusability for subsequent broadcasts: Clearly, the number of nodes n , which is not *a priori* known here, can be obtained through the notification process of the first broadcast (by having nodes reporting their number of descendants in the tree, while notifying hierarchically). All subsequent broadcasts can thus behave as if both n and Δ were known. Next we show this technique allows one to solve the problem without any control messages.

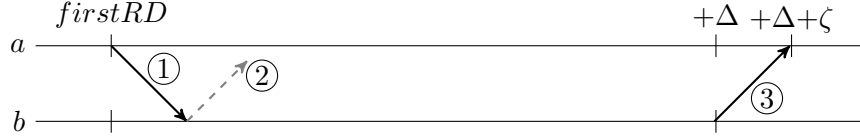


Figure 25: Case when a node waits $\Delta + \zeta$ for receiving potential affiliation messages. Picture from (Casteigts et al. 2012c).

Knowledge of both n and Δ

In this case, the emitter knows an upper bound on the broadcast termination date; in fact, the broadcast cannot last longer than $n\Delta$ (this worst case is when the foremost tree is a line graph). Termination detection can thus become implicit after this amount of time, which removes the need for any control message (whether of affiliation or of notification).

Theorem 16. *When Δ and n are known, $\text{TDB}[\text{foremost}]$ can be solved in \mathcal{B} in $O(n\Delta)$ time by exchanging $O(m)$ information messages and no control messages.*

6.4 $\text{TDB}[\text{shortest}]$

Let us recall that the objective of $\text{TDB}[\text{shortest}]$ is to deliver the information to each node within a minimal *number of hops* from the emitter, and to have the emitter detect termination within finite time. In contrast to the foremost case, knowing n is insufficient to perform a shortest broadcast in \mathcal{R} or even in \mathcal{B} . However, it becomes feasible in \mathcal{B} when Δ is also known. Moreover, any shortest tree built at some time t will remain optimal in \mathcal{B} relative to any future emission date $t' > t$. This feature allows for the possible reuse of the solution to $\text{TDB}[\text{shortest}]$ in subsequent broadcasts.

It is shown below that knowing n is not sufficient to solve $\text{TDB}[\text{shortest}]$ in \mathcal{B} (and thus in \mathcal{R}). We also describe how to solve the problem when Δ is known, and finally when both n and Δ are known.

$\text{TDB}[\text{shortest}]$ in \mathcal{B} with knowledge of n

The following theorem establishes that knowing n is not sufficient to solve $\text{TDB}[\text{shortest}]$ in \mathcal{B} (and thus in \mathcal{R}).

Theorem 17. *$\text{TDB}[\text{shortest}]$ is not feasible in \mathcal{B} (nor a fortiori in \mathcal{R}) knowing only n .*

Proof. By contradiction, let \mathcal{A} be an algorithm that solves $\text{TDB}[\text{shortest}]$ in \mathcal{B} with the knowledge of n only. Consider an arbitrary $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{B}$ and $x \in V$. Execute \mathcal{A} in \mathcal{G} starting at time t_0 with x as the source. Let t_f be the time when the source terminates and T the shortest broadcast tree along which broadcast was performed. Let $\mathcal{G}' = (V', E', \mathcal{T}', \rho') \in \mathcal{B}$ such that $V' = V$, $E' = E \cup \{(x, v) \text{ for some } v \in V : (x, v) \notin E\}$, $\rho'(e, t) = \rho(e, t)$ for all $e \in E, 0 \leq t \leq t_f$, $\rho'((x, v), t) = 0$ for all $t_0 \leq t < t_f$, and $\rho'((x, v), t) = 1$ for some $t > t_f$ (we can take Δ as large as needed here). Consider

the execution of \mathcal{A} in \mathcal{G}' starting at time t_0 with x as the source. Since (x, v) does not appear between t_0 and t_f , the execution of \mathcal{A} at every node in \mathcal{G}' will be exactly as at the corresponding node in \mathcal{G} and terminate with v having received the information in more than one hop, contradicting the fact that T is a shortest tree, and thus the correctness of \mathcal{A} . \square

TDB[*shortest*] in \mathcal{B} with knowledge of Δ

We present here a solution to TDB[*shortest*] in \mathcal{B} when Δ is known. The idea is to propagate the message along the edges of a *breadth*-first spanning tree of the underlying graph. Here is an informal description of the algorithm in (Casteigts et al. 2012c). Assuming that the message is created at some date t , the mechanism consists of authorizing nodes at level i in the tree to inform new nodes only between time $t + i\Delta$ and $t + (i + 1)\Delta$ (doing it earlier would lead to a non-shortest tree, while doing it later is pointless because all the edges have necessarily appeared within one Δ). So the broadcast is confined into rounds of duration Δ as follows: whenever a node sends the information to another, it sends a time value that indicates the remaining duration of its round (that is, the starting date of its own round plus Δ minus the current time minus the crossing delay), so the receiving node knows when to start informing new nodes in turn (if it did not have the information yet). An example of this timing is shown in Figure 26. When the node a attempts to become b 's parent, node a transmits its own starting date plus Δ minus the current date minus ζ . This duration corresponds to the exact amount of time the child would have to wait, if the relation is established, before integrating other nodes in turn. If a node has not informed any other node during its round, it notifies its parent. When a node has been notified by all its children, it notifies its parent. Note that this requires parents to keep track of the number of children they have, and thus children need to send *affiliation* messages when they select a parent (with the same constraints as already discussed in Figure 25). Finally, when the emitter has been notified by all its children, it knows the broadcast is terminated.

Theorem 18. *When Δ is known, TDB[shortest] can be solved in \mathcal{B} in $O(n\Delta)$ time by exchanging $O(m)$ information messages and $O(n)$ control messages.*

Proof. The fact that the algorithm constructs a breadth-first (and thus shortest) delay-tolerant spanning tree follows from the connectivity over time of the underlying graph and from the knowledge of the duration Δ . The bound on recurrence is used to enable a rounded process whereby the correct distance of each node to the emitter is detected. The number of *information* messages is $2m$ as the dissemination process exchanges at most two messages per edge. The number of *affiliation* and *notification* messages are each of $n - 1$ (one per edge of the tree). The time complexity for the construction of the tree is at most $(n - 1)\Delta$ to reach the last node, plus $\Delta + \zeta$ at this node, plus at most $(n - 1)\Delta$ to aggregate this node's notification. (The additional ζ caused by waiting affiliation messages matters only for the last round, since the construction continues in parallel otherwise.) The total is thus at most $(2n - 1)\Delta + \zeta$. \square

Reusability for subsequent broadcasts: Thanks to the fact that shortest trees remain shortest regardless of the emission date, all subsequent broadcasts can be performed within

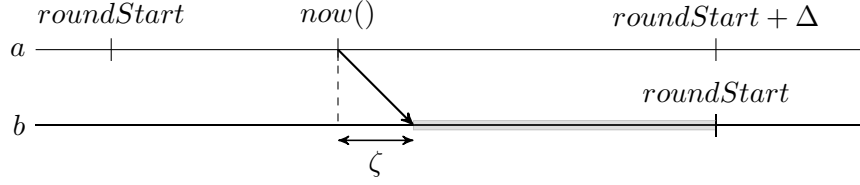


Figure 26: Propagation of rounds of duration Δ . Picture from (Casteigts et al. 2012c).

the same, already known tree, which reduces the number of information message from $O(m)$ to $O(n)$. Moreover, if the depth d of the tree is detected through the first notification process then all subsequent broadcasts can enjoy an implicit termination detection that is itself optimal in time (after $d\Delta$ time). No control message is needed.

TDB[shortest] in \mathcal{B} with knowledge of n and Δ

When both n and Δ are known, one can apply the same dissemination procedure as above (knowledge of Δ) combined with an implicit termination detection deduced from the very first broadcast (after $n\Delta$ time) and thus avoid using control messages.

Theorem 19. *When n and Δ are known, TDB[shortest] can be solved in \mathcal{B} in $O(n\Delta)$ time by exchanging $O(m)$ information messages and no control messages.*

However, avoiding control messages prevents the emitter from learning the depth d of the shortest tree, and thus prevents lowering the termination bound to $d\Delta$ time. An alternative solution would be to achieve explicit termination for the first broadcast in order to build a reusable broadcast tree (and learn its depth d in the process). In this case, dissemination is achieved with $O(m)$ information messages, termination detection is achieved as above with $O(n)$ control messages (where, however, affiliation messages are not necessary, and the number of control messages would decrease to $n - 1$). In this way, we would have an increase in control messages, but the subsequent broadcasts could reuse the broadcast tree for dissemination with $O(n)$ information messages, and termination detection could be implicit with no exchange of control message at all after $d\Delta$ time. The choice of either solution may depend on the size of the information message and on the expected number of broadcasts planned.

6.5 Computational relationship

On the basis of these results, Casteigts et al. (2012c) conclude regarding the computational relationship between $\mathcal{P}(\mathcal{R}_n)$, $\mathcal{P}(\mathcal{B}_\Delta)$, and $\mathcal{P}(\mathcal{P}_p)$.

Theorem 20. $\mathcal{P}(\mathcal{R}_n) \subsetneq \mathcal{P}(\mathcal{B}_\Delta) \subsetneq \mathcal{P}(\mathcal{P}_p)$

Proof. The fact that $\mathcal{P}(\mathcal{R}_n) \subseteq \mathcal{P}(\mathcal{B}_\Delta) \subseteq \mathcal{P}(\mathcal{P}_p)$ was observed in Theorem 7. To make the left inclusion strict, one has to exhibit a problem Π such that $\Pi \in \mathcal{P}(\mathcal{B}_\Delta)$ and $\Pi \notin \mathcal{P}(\mathcal{R}_n)$. By Theorem 17 and Theorem 18, TDB[shortest] is one such example. The right inclusion is

similarly proven strict, based on the fact that $\text{TDB}[fastest]$ is in $\mathcal{P}(\mathcal{P}_d)$ (Theorem 11) but it is not in $\mathcal{P}(\mathcal{B}_\Delta)$ (Theorem 10). \square

Now, considering the fact that $\text{TDB}[foremost] \in \mathcal{P}(\mathcal{R}_n)$ while $\text{TDB}[shortest] \notin \mathcal{P}(\mathcal{R}_n)$, and the fact that $\text{TDB}[shortest] \in \mathcal{P}(\mathcal{B}_\Delta)$ while $\text{TDB}[fastest] \notin \mathcal{P}(\mathcal{B}_\Delta)$, together with the inclusions of Theorem 20, we have

$$\text{TDB}[foremost] \preceq_f \text{TDB}[shortest] \preceq_f \text{TDB}[fastest]$$

where \preceq_f is a partial order on these problems topological requirements (relative to *feasibility*). The order is “only” partial here because the variations of feasibility of these problems may be different in another set of assumptions. Following a similar reasoning (that is the fact that the solutions to $\text{TDB}[shortest]$ are reusable in \mathcal{B}_Δ whereas those to $\text{TDB}[foremost]$ are not) leads to

$$\text{TDB}[shortest] \preceq_r \text{TDB}[foremost]$$

where \preceq_r is a partial order on these problems topological requirements (relative to *reusability*). This result is somehow surprising, as it suggests a problem could be both easier or more difficult than another, depending on which aspect is looked at (feasibility *vs.* reusability). In other words, the difficulty of these problems seems to be multi-dimensional. Whether reusability is easier for $\text{TDB}[fastest]$ or $\text{TDB}[foremost]$ is an open question, since both of these problems are not solvable in \mathcal{B}_Δ but are solvable in \mathcal{P}_p (Casteigts et al. 2012b).

7 Classification of networks and algorithms

In this section, we show how the conditions (\mathcal{C}_1 to \mathcal{C}_4) characterized in Section 5 (and repeated below) can be used to define dynamic graph classes, some of which are included in others. These conditions are related to the analyses of a number of algorithms: \mathcal{A}_1 for propagation, \mathcal{A}_2 for centralized counting, and \mathcal{A}_3 for distributed counting. The dynamic graph classes are in turn connected to those studied in Section 6 and to more classes from recent literature. Overall, these inclusion relations yield a hierarchy of 17 dynamic graph classes, each of which corresponds to a feasible or unfeasible problem. Such a hierarchy can also be used, in turn, to compare algorithms or problems on the fair basis of their topological requirements in a dynamic context.

7.1 From conditions to classes of dynamic graphs

From $\mathcal{C}_1 = (\forall v \in V, emitter \rightsquigarrow v)$, we derive two classes of dynamic graphs. \mathcal{F}_1 is the class in which at least one vertex can reach all the others by a journey. If a dynamic graph does not belong to this class then there is no chance for algorithm \mathcal{A}_1 to succeed whatever the initial emitter. \mathcal{F}_2 is the class where every vertex can reach all the others by a journey. If a dynamic graph does not belong to this class then at least one vertex, if chosen as an initial emitter, will fail to inform all the others using algorithm \mathcal{A}_1 .

From $\mathcal{C}_2 = (\forall v \in V, emitter \overset{st}{\rightsquigarrow} v)$, we derive two classes of dynamic graphs. \mathcal{F}_3 is the class in which at least one vertex can reach all the others by a *strict* journey. If a dynamic graph belongs to this class then there is at least one vertex that could, for certain, inform all the others using algorithm \mathcal{A}_1 (under Progression Hypothesis 1 on page 38). \mathcal{F}_4 is the class of dynamic graphs in which every vertex can reach all the others by a *strict* journey. If a dynamic graph belongs to this class then the success of algorithm \mathcal{A}_1 is guaranteed for any vertex as initial emitter (again, under Progression Hypothesis 1).

From $\mathcal{C}_3 = (\forall v \in V \setminus \{counter\}, (counter, v) \in E)$, we derive two classes of graphs. \mathcal{F}_5 is the class of dynamic graphs in which at least one vertex shares, at some point of the execution, an edge with every other vertex. If a dynamic graph does not belong to this class then there is no chance of success for algorithm \mathcal{A}_2 , whatever the vertex chosen for counter. Here, if we assume Progression Hypothesis 1 then \mathcal{F}_5 is also a class in which the success of the algorithm can be guaranteed for one specific vertex as counter. \mathcal{F}_6 is the class of dynamic graphs in which every vertex shares an edge with every other vertex at some point of the execution. If a dynamic graph does not belong to this class then there exists at least one vertex that cannot count all the others using algorithm \mathcal{A}_2 . Again, if we consider Progression Hypothesis 1 then \mathcal{F}_6 becomes a class in which the success is guaranteed whatever the counter.

Finally, from $\mathcal{C}_4 = (\exists v \in V : \forall u \in V, u \rightsquigarrow v)$, we derive the class \mathcal{F}_7 , which is the class of graphs such that at least one vertex can be reached from all the others by a journey (in other words, the intersection of all nodes *horizons* is non-empty). If a graph does not belong to this class then there is absolutely no chance of success for algorithm \mathcal{A}_3 .

7.2 Relations between classes

The classes defined so far can be connected into a hierarchy by means of inclusion relations (Casteigts 2007, Casteigts et al. 2009a). Since “all” implies “at least one”, we have $\mathcal{F}_2 \subseteq \mathcal{F}_1$, $\mathcal{F}_4 \subseteq \mathcal{F}_3$, and $\mathcal{F}_6 \subseteq \mathcal{F}_5$. Because a strict journey is a journey, we have $\mathcal{F}_3 \subseteq \mathcal{F}_1$, and $\mathcal{F}_4 \subseteq \mathcal{F}_2$. Since an edge is a (strict) journey, we have $\mathcal{F}_5 \subseteq \mathcal{F}_3$, $\mathcal{F}_6 \subseteq \mathcal{F}_4$, and $\mathcal{F}_5 \subseteq \mathcal{F}_7$. Finally, the existence of a journey between all pairs of vertices (\mathcal{F}_2) implies that each vertex can be reached by all the others, which implies in turn that at least one vertex can be reach by all the others (\mathcal{F}_7). We thus have $\mathcal{F}_2 \subseteq \mathcal{F}_7$. Although we have used here a non-strict inclusion (\subseteq), all the inclusions actually turn out to be strict (\subsetneq)—one can easily find for each inclusion a graph that belongs to the parent class but is outside the child class. Figure 27 summarizes these relations.

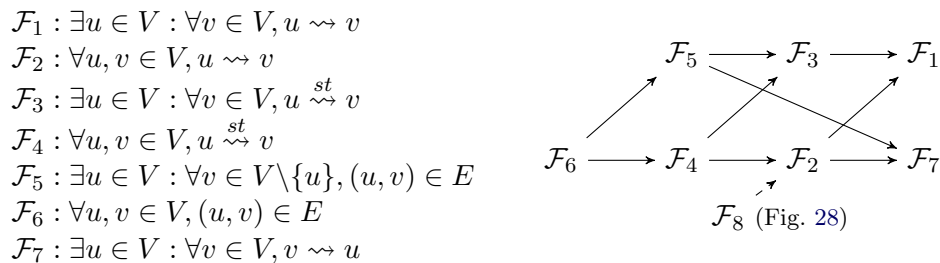


Figure 27: A first hierarchy of dynamic networks based on dynamic graph properties that result from previous analyses. Picture from (Casteigts et al. 2009a).

Further classes were studied in Section 6 and others introduced in the recent literature. They are organized into a hierarchy in (Casteigts et al. 2012d). These classes include \mathcal{F}_8 (*round connectivity*): every node can reach every other node, and be reached back afterwards; \mathcal{F}_9 : (*recurrent connectivity*): every node can reach all the others infinitely often; \mathcal{F}_{10} (*recurrence of edges*): the underlying graph $G = (V, E)$ is connected, and every edge in E re-appears infinitely often; \mathcal{F}_{11} (*time-bounded recurrence of edges*): same as \mathcal{F}_{10} , but the re-appearance is bounded by a given time duration; \mathcal{F}_{12} (*periodicity*): the underlying graph G is connected and every edge in E re-appears at regular intervals; \mathcal{F}_{13} (*eventual instant-routability*): given any pair of nodes and at any time, there always exists a future G_i in which a (static) path exists between them; \mathcal{F}_{14} (*eventual instant-connectivity*): at any time, there always exists a future G_i that is connected in a classic sense (i.e., a static path exists in G_i between any pair of nodes); \mathcal{F}_{15} (*perpetual instant-connectivity*): every G_i is connected in a static sense; \mathcal{F}_{16} (*T-interval-connectivity*): all the graphs in any sub-sequence $G_i, G_{i+1}, \dots, G_{i+T}$ have at least one connected spanning subgraph in common. Finally, \mathcal{F}_{17} is the reference class for population protocols, it corresponds to the subclass of \mathcal{F}_{10} in which the underlying graph G (*graph of interaction*) is a complete graph.

All these classes have been shown to have particular algorithmic significance. For example, \mathcal{F}_{16} allows one to speed up the execution of some algorithms by a factor T (Kuhn et al. 2010). In the context of broadcast, \mathcal{F}_{15} allows one to have at least one new node informed in every G_i , and consequently to bound the broadcast time by (a constant factor of) the network size (O’Dell and Wattenhofer 2005). \mathcal{F}_{13} and \mathcal{F}_{14} were used by Ramanathan et al. (2007) to

characterize the contexts in which *non-delay-tolerant* routing protocols can eventually work if they retry upon failure. Classes \mathcal{F}_{10} , \mathcal{F}_{11} , and \mathcal{F}_{12} have been shown to have an impact on the distributed versions of *foremost*, *shortest*, and *fastest* broadcasts with termination detection. Precisely, foremost broadcast is feasible in \mathcal{F}_{10} , whereas shortest and fastest broadcasts are not; shortest broadcast becomes feasible in \mathcal{F}_{11} (Casteigts et al. 2010a), whereas fastest broadcast is not and becomes feasible in \mathcal{F}_{12} (See also Section 6). Also, even though foremost broadcast is possible in \mathcal{F}_{10} , recording the journeys for subsequent use is not possible in \mathcal{F}_{10} nor \mathcal{F}_{11} ; it is however possible in \mathcal{F}_{12} (Casteigts et al. 2012b). Finally, \mathcal{F}_8 can be regarded as a *sine qua non* for termination detection in many contexts.

Interestingly, this second range of classes—from \mathcal{F}_8 to \mathcal{F}_{17} —can also be entirely connected through inclusions, as illustrated on Figure 28. Both classifications can also be interconnected through \mathcal{F}_8 , a subclass of \mathcal{F}_2 , which brings us to a total of 17 connected classes. A hierarchy of this type can be useful in several respects, including the possibility to transpose results or to compare solutions or problems on a formal basis, which we discuss now.

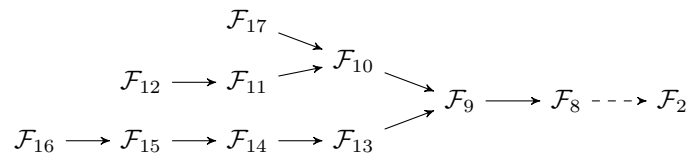


Figure 28: An additional hierarchy of dynamic networks based on further classes found in recent literature. Picture from Casteigts et al. (2012d).

7.3 Comparison of algorithms based on their requirements

Let us consider the two counting algorithms given in Section 5. To have any chance of success, the centralized counting algorithm, \mathcal{A}_2 , requires the dynamic graph to be in \mathcal{F}_5 (with a fortunate choice of counter) or in \mathcal{F}_6 (with any vertex as counter). On the other hand, the distributed counting algorithm, \mathcal{A}_3 , requires the dynamic graph to be in \mathcal{F}_7 . Since both \mathcal{F}_5 (directly) and \mathcal{F}_6 (transitively) are included in \mathcal{F}_7 , there are some topological scenarios (i.e., $\mathcal{G} \in \mathcal{F}_7 \setminus \mathcal{F}_5$) in which \mathcal{A}_2 has no chance of success, while \mathcal{A}_3 has some. Such observations allow us to claim that \mathcal{A}_3 is more general than \mathcal{A}_2 with respect to its topological requirements. This example illustrates how a classification can help compare two solutions on a fair and formal basis. In the case of these two counting algorithms, however, the claim could be balanced by the fact that the sufficient condition for \mathcal{A}_3 is more demanding than that for \mathcal{A}_2 (given a fortunate choice of node as counter; they are equivalent otherwise).

A similar type of reasoning could also teach us something about the problems themselves, as discussed in Section 6. Indeed, the feasibility results on *shortest*, *fastest*, and *foremost* broadcast with termination detection combined with the inclusion relation $\mathcal{F}_{12} \subset \mathcal{F}_{11} \subset \mathcal{F}_{10}$ tells us there exists a partial order of difficulty between these problems in terms of topological requirements (*foremost* \preceq_f *shortest* \preceq_f *fastest*).

We believe classifications of this type have the potential to lead more equivalence results and formal comparisons between problems and algorithms. Now, one must also keep in mind

that these are only *topology-related* conditions, and that other dimensions of properties—e.g., what knowledge is available to the nodes, or whether they have unique identifiers—keep playing the same important role as they do in static networks. As an example, the above partial ordering of difficulty omits the fact that detecting termination in the *foremost* case in \mathcal{F}_{10} requires the emitter to know the number of nodes n in the network, whereas this knowledge is not necessary for shortest broadcast in \mathcal{F}_{11} (the alternative knowledge of knowing a bound on the recurrence time is, however, not weaker).

8 Classes of dynamic graphs vs. mobility contexts

What properties do real mobile networks such as sensors, pedestrians, robots, UAVs, vehicles, or satellites have in terms of dynamics? Each of these networks is of course dynamic, but in its own peculiar way. It seems essential, and extremely relevant, to understand what assumptions (both topological or computational) are representative and/or realistic for each specific context.

For instance, it is generally understood that satellites have periodic movements (class \mathcal{F}_{12}), while sensor networks are, in general, connected at any instant (class \mathcal{F}_{15}). Interaction between smartphones may represent interactions between people in a given context, such as in a small company with bounded recurrence of edges, typically within a week (class \mathcal{F}_{11}), or in a community with unbounded, yet recurrent interactions (class \mathcal{F}_{10}). Vehicular networks exhibit an important range of densities and connectivity patterns, but they still offer recurrent connectivity over time and space (class \mathcal{F}_9). Vehicles also share some traits with pedestrians through having their movements constrained by the environment (roads and pathways).

On the other hand, robots or UAVs (and to some extent, soldiers too) share the convenient feature that they can influence their own mobility. For instance, they may modify their movements for the very purpose of enforcing some topological properties that are required to complete a given task. They may also avoid interrupting an ongoing crucial communication, making it more realistic to consider coarser-grain atomicity in their computational model.

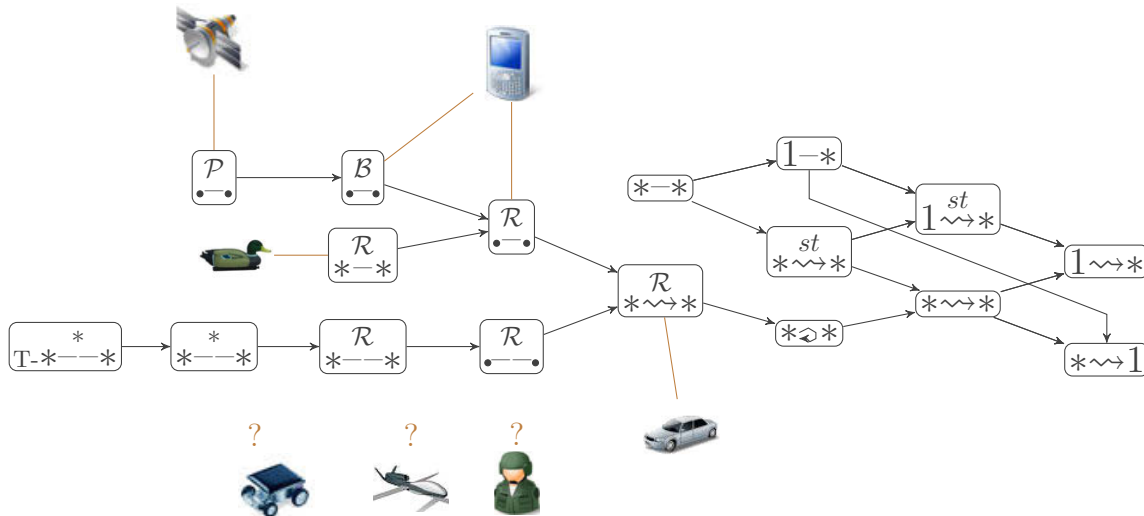


Figure 29: Relations between mobility contexts and classes of dynamic graphs.

Through linking mobility models to analytical 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. Finally, this may also help understand how networks of different natures can interconnect.

9 Conclusion

As the prevalence of dynamic networks and related applications is constantly growing, the need for a better understanding of their nature is increasingly important. Heretofore, the high complexity of dynamic networking environments led most researchers of the domain to study the new problems and solutions by means of simulations. Simulations are beyond doubt an essential component; however, they are by nature incomplete in that they mask the link between a given setting and the corresponding outcome. At the best, one can observe that a given solution works well (or better than another) in a given context, but small changes in the settings may have tremendous implications on the outcome, which makes simulation an unsafe approach for critical applications whose execution context is not perfectly predictable.

This report has provided an overview of recent algorithmic techniques and analysis approaches that aim to consider dynamic networks from a formal perspective. The topics we discussed are certainly not exhaustive; in particular, we did not elaborate on some restricted types of dynamic networks that have received attention lately. These include fault-tolerant networks (where topological events are only occasional) and perpetually connected dynamic networks (where there always exists a path between any pair of nodes). We have focused instead on those highly dynamic networks whose connectivity is never guaranteed at any given time, the network being likely partitioned into a number of separate components (possibly merging or splitting at times). We believe this context reflects dynamic networks better, and to some extent, that making less assumptions first is a good thing when trying to solve a problem.

Taking these networks as our main objective, we discussed a number of related topics. In particular, the significant impact these environments have on the very definition of problems, such as exploring the network or building communication structures. In many cases, typical construction problems become problems of *maintenance* in a dynamic network (e.g., maintaining one leader per component, a spanning forest, or a dominating set) where the solution is required to adapt as the network evolves. We suggested a list of generic approaches or constructs that can be helpful to solve or analyze various problems. These include, among others, temporal-lags vector clocks (or T-CLOCKS) that have the ability to measure delays between nodes in just such a disconnected context, which can be used to solve concrete problems like foremost or fastest broadcast, and abstract models of computation, which can be used to solve more general problems.

The remainder of the document dealt with important aspects of the analysis of distributed algorithms and dynamic graph properties with a strong emphasis on the link between both. In particular, we showed how a given property of the network dynamics could be proven to be necessary or sufficient to the success of a given algorithm, and also at a general level. We described the computational relationship between three important classes of dynamic networks: networks whose edges re-appear recurrently, bounded-recurrently, or periodically. Finally, we showed how all the graph properties studied in the document plus others from the literature can be connected through a hierarchy of dynamic graph classes, itself having many possible applications.

An important question we have not addressed in this document is the relationship between the number of topological events and the cost of an algorithm. Indeed, many algorithmic operations in highly dynamic networks are triggered in reaction to topological events (e.g. the appearance or disappearance of a local link). However, the complexity of algorithms is more often than not characterized in the sole terms of the number of nodes or edges, thus ignoring the impact of this essential parameter. Much remains to be done around these questions. In fact, the field is just opening up and many research avenues still wait to be explored. We hope this report contributed to developing the interest of the reader in formal approaches to dynamic networks, and makes the point for such approaches in particular when targeting critical applications whose outcome must be guaranteed. Besides, we believe this research area raises many interesting questions which deserve to be explored in their own, scientific, right.

This page intentionally left blank.

References

- Albers, S. and Henzinger, M. R. (2000), Exploring unknown environments, *SIAM Journal on Computing*, 29(4), 1164–1188.
- Ambühl, C., Gasieniec, L., Pelc, A., Radzik, T., and Zhang, X. (2011), Tree exploration with logarithmic memory, *ACM Transactions on Algorithms*, 7(2), 17:1–17:21.
- Angluin, D. (1980), Local and global properties in networks of processors, In *Proceedings of the 12th annual ACM symposium on Theory of Computing (STOC)*, pp. 82–93, ACM.
- Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M., and Peralta, R. (2006), Computation in networks of passively mobile finite-state sensors, *Distributed Computing*, 18(4), 235–253.
- Ausiello, G., Bonifaci, V., and Escoffier, B. (2011), Complexity and Approximation in Reoptimization, In Cooper, S. and Sorbi, A., (Eds.), *Computability in Context: Computation and Logic in the Real World*, pp. 101–130, World Scientific.
- Averbakh, I. and Berman, O. (1996), A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree, *Discrete Applied Mathematics*, 68(1–2), 17–32.
- Awerbuch, B., Betke, M., Rivest, R. L., and Singh, M. (1999), Piecemeal graph exploration by a mobile robot, *Information and Computation*, 152(2), 155–172.
- Barrière, L., Flocchini, P., Fraigniaud, P., and Santoro, N. (2003), Can we elect if we cannot compare?, In *Proceedings of the 15th ACM symposium on Parallel algorithms and architectures (SPAA)*, pp. 324–332, ACM.
- Bender, M. A., Fernández, A., Ron, D., Sahai, A., and Vadhan, S. (1998), The power of a pebble: Exploring and mapping directed graphs, In *STOC '98: Proceedings of the 30th ACM Symposium on Theory of Computing*, pp. 269–278, New York, NY, USA: ACM.
- Bender, M. A. and Slonim, D. K. (1994), The power of team exploration: Two robots can learn unlabeled directed graphs, In *FOCS 1994: Proceedings of the 35th Symposium on Foundations of Computer Science*, pp. 75–85.
- Blum, M. and Kozen, D. (1978), On the power of the compass (or, Why mazes are easier to search than graphs), In *FOCS 1978: Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pp. 132–142, IEEE.
- Brejová, B., Dobrev, S., Kráľovič, R., and Vinař, T. (2011), Routing in carrier-based mobile networks, In *Proceedings of the 18th international conference on Structural information and communication complexity, SIROCCO'11*, pp. 222–233, Berlin, Heidelberg: Springer-Verlag.
- Bui-Xuan, B., Ferreira, A., and Jarry, A. (2003), Computing shortest, fastest, and foremost journeys in dynamic networks, *International Journal of Foundations of Computer Science*, 14(2), 267–285.

Casteigts, A. (2006), Model Driven capabilities of the DA-GRS model, In *Proceedings of the 1st International Conference on Autonomic and Autonomous Systems (ICAS'06)*, pp. 24–32, Washington, DC, USA: IEEE Computer Society.

Casteigts, A. (2007), Contribution à l'Algorithmique Distribuée dans les Réseaux Mobiles Ad Hoc - Calculs Locaux et Réétiquetages de Graphes Dynamiques, Ph.D. thesis, University of Bordeaux.

Casteigts, A. and Chaumette, S. (2005), Dynamicity-Aware Graph Relabeling Systems (DA-GRS), a local computation-based model to describe MANet algorithms, In *Proceedings of the 17th Conference on Parallel and Distributed Computing and Systems (PDCS'05)*, pp. 231–236, Dallas, USA.

Casteigts, A., Chaumette, S., and Ferreira, A. (2009), Characterizing Topological Assumptions of Distributed Algorithms in Dynamic Networks, In *Proceedings of the 16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pp. 126–140, Piran, Slovenia: Springer. (Full version on *arXiv:1102.5529*).

Casteigts, A., Chaumette, S., and Ferreira, A. (2012), Distributed Computing in Dynamic Networks: Towards a Framework for Automated Analysis of Algorithms, *CoRR*, Vol. abs/1102.5529.

Casteigts, A. and Flocchini, P. (2013), Deterministic Algorithms in Dynamic Networks: Formal models and metrics, (CR 2013-020) Defence R&D Canada – Ottawa.

Casteigts, A., Flocchini, P., Mans, B., and Santoro, N. (2010), Deterministic Computations in Time-Varying Graphs: Broadcasting under Unstructured Mobility, In *Proceedings of 5th IFIP Conference on Theoretical Computer Science (TCS)*, pp. 111–124, Brisbane, Australia: Springer.

Casteigts, A., Flocchini, P., Mans, B., and Santoro, N. (2012), Measuring Temporal Lags in Delay-Tolerant Networks, *IEEE Transactions on Computers*. In press.

Casteigts, A., Flocchini, P., Mans, B., and Santoro, N. (2012), Shortest, Fastest, and Foremost Broadcast in Dynamic Networks, Technical Report University of Ottawa.

Casteigts, A., Flocchini, P., Quattrociocchi, W., and Santoro, N. (2012), Time-varying graphs and dynamic networks, *International Journal of Parallel, Emergent and Distributed Systems*, 27(5), 387–408.

Casteigts, A., Mans, B., and Mathieson, L. (2011), On the Feasibility of Maintenance Algorithms in Dynamic Graphs, *CoRR*, Vol. abs/1107.2722.

Casteigts, A., Nayak, A., and Stojmenovic, I. (2010), Topology Control in Sensor, Actuator and Mobile Robot Networks, Ch. 7 of *Wireless Sensor and Actuator Networks - Algorithms and Protocols for Scalable Coordination and Data Communication*, Nayak, A. and Stojmenovic, I. (eds), Wiley.

Casteigts, A., Chaumette, S., Guinand, F., and Pigné, Y. (2009), Distributed Maintenance of Anytime Available Spanning Trees in Dynamic Networks, *CoRR*, Vol. abs/0904.3087.

- Chalopin, J., Godard, E., Métivier, Y., and Ossamy, R. (2006), Mobile agent algorithms versus message passing algorithms, *Principles of Distributed Systems*, pp. 187–201.
- Chatzigiannakis, I., Dolev, S., Fekete, S., Michail, O., and Spirakis, P. (2009), Not all fair probabilistic schedulers are equivalent, In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)*, pp. 33–47, Springer.
- Chatzigiannakis, I., Michail, O., Nikolaou, S., Pavlogiannis, A., and Spirakis, P. (2010), All symmetric predicates in NSPACE (n^2) are stably computable by the mediated population protocol model, In *Proceedings of the 35th Symposium on Mathematical Foundations of Computer Science (MFCS)*, pp. 270–281, Springer.
- Cooper, C., Elsässer, R., Ono, H., and Radzik, T. (2012), Coalescing random walks and voting on graphs, In *Proceedings of the 31st ACM symposium on Principles of distributed computing (PODC)*, pp. 47–56, ACM.
- Das, S., Flocchini, P., Santoro, N., and Yamashita, M. (2007), Fault-tolerant simulation of message-passing algorithms by mobile agents, *Structural Information and Communication Complexity*, pp. 289–303.
- Das, S., Flocchini, P., Kutten, S., Nayak, A., and Santoro, N. (2007), Map construction of unknown graphs by multiple agents, *Theoretical Computer Science*, 385(1–3), 34–48.
- Das, S., Flocchini, P., Nayak, A., and Santoro, N. (2005), Distributed exploration of an unknown graph, In *SIROCCO 2005: Proceedings of the 12th International Colloquium on Structural Information and Communication Complexity*, Vol. 3499 of *Lecture Notes in Computer Science*, pp. 99–114, Springer.
- Deng, X. and Papadimitriou, C. H. (1990), Exploring an unknown graph, In *FCS 1990: Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, pp. 355–361.
- Deng, X. and Papadimitriou, C. H. (1999), Exploring an unknown graph, *Journal of Graph Theory*, 32(3), 265–297.
- Dessmark, A. and Pelc, A. (2004), Optimal graph exploration without good maps, *Theoretical Computer Science*, 326(1–3), 343–362.
- Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. (2002), Searching for a black hole in arbitrary networks: optimal mobile agent protocols, In *Proceedings of the 21st symposium on Principles of distributed computing (PODC)*, pp. 153–162, ACM.
- Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. (2006), Searching for a black hole in arbitrary networks: Optimal mobile agents protocols, *Distributed Computing*, 19(1), 1–18.
- Dubois-Ferriere, H., Grossglauser, M., and Vetterli, M. (2003), Age matters: efficient route discovery in mobile ad hoc networks using encounter ages, In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, p. 266.

- Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1991), Robotic exploration as graph construction, *IEEE Transactions on Robotics and Automation*, 7(6), 859–865.
- Dynia, M., Lopuszański, J., and Schindelhauer, C. (2007), Why robots need maps, In *SIROCCO 2007: Proceedings of the 14th International Colloquium Structural Information and Communication Complexity*, Vol. 4474 of *Lecture Notes in Computer Science*, pp. 41–50, Springer.
- Flocchini, P., Kellett, M., Mason, P., and Santoro, N. (2009), Map construction and exploration by mobile agents scattered in a dangerous network, In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pp. 1–10, IEEE.
- Flocchini, P., Kellett, M., Mason, P., and Santoro, N. (2010), Mapping an unfriendly subway system, In *Proceedings of the 5th International Conference on Fun with Algorithms (FUN)*, pp. 190–201.
- Flocchini, P., Kellett, M., Mason, P., and Santoro, N. (2012), Fault-Tolerant Exploration of an Unknown Dangerous Graph by Scattered Agents, In *Proceedings of the 14th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*.
- Flocchini, P., Kellett, M., Mason, P., and Santoro, N. (2012), Finding Good Coffee in Paris, In *Proceedings of the 6th Int. Conference on Fun with Algorithms (FUN)*, pp. 154–165, Springer.
- Flocchini, P., Kellett, M., Mason, P., and Santoro, N. (2012), Searching for black holes in subways, *Theory of Computing Systems*, 50(1), 158–184.
- Flocchini, P., Kellett, M., Mason, P., and Santoro, N. (2012), Searching for Black Holes in Subways, *Theory of Computing Systems*, 50(1), 158–184.
- Flocchini, P., Mans, B., and Santoro, N. (2009), Exploration of periodically varying graphs, In *Proceedings of 20th International Symposium on Algorithms and Computation (ISAAC)*, pp. 534–543.
- Flocchini, P., Ilcinkas, D., Pelc, A., and Santoro, N. (2007), Computing without communicating: Ring exploration by asynchronous oblivious robots, In *OPODIS 2007: Proceedings of the 11th International Conference on Principles of Distributed Systems*, Vol. 4878 of *Lecture Notes in Computer Science*, pp. 105–118, Springer.
- Flocchini, P., Ilcinkas, D., Pelc, A., and Santoro, N. (2008), Remembering without memory: Tree exploration by asynchronous oblivious robots, In *SIROCCO 2008: Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity*, Vol. 5058 of *Lecture Notes in Computer Science*, pp. 33–47, Springer.
- Flocchini, P., Ilcinkas, D., Pelc, A., and Santoro, N. (2010), Remembering without memory: Tree exploration by asynchronous oblivious robots, *Theoretical Computer Science*, 411(14-15), 1583–1598.

- Fraigniaud, P., Gasieniec, L., Kowalski, D. R., and Pelc, A. (2006), Collective tree exploration, *Networks*, 48(3), 166–177.
- Fraigniaud, P., Gasieniec, L., Kowalski, D. R., and Pelc, A. (2004), Collective tree exploration, In *LATIN 2004: Proceedings of the 6th Latin American Symposium Theoretical Informatics*, Vol. 2976 of *Lecture Notes in Computer Science*, pp. 141–151, Springer.
- Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., and Peleg, D. (2005), Graph exploration by a finite automaton, *Theoretical Computer Science*, 345(2–3), 331–344.
- Fraigniaud, P., Ilcinkas, D., and Pelc, A. (2006), Tree exploration with an oracle, In *MFCSS 2006: Proceedings of the 31st International Symposium on the Mathematical Foundations of Computer Science*, Vol. 4162 of *Lecture Notes in Computer Science*, pp. 24–37, Springer.
- Gasieniec, L., Pelc, A., Radzik, T., and Zhang, X. (2007), Tree exploration with logarithmic memory, In *SODA 2007: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 585–594, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.
- Ginsburg, S. and Spanier, E. (1966), Semigroups, Presburger formulas and languages, *Pacific Journal of Mathematics*, 16(2), 285–296.
- Grossglauser, M. and Vetterli, M. (2003), Locating nodes with EASE: Last encounter routing in ad hoc networks through mobility diffusion, In *Proceedings of 22nd Conference on Computer Communications (INFOCOM)*, Vol. 3, pp. 1954–1964, San Francisco, USA: IEEE.
- Holm, J., de Lichtenberg, K., and Thorup, M. (2001), Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity, *Journal of the ACM*, 48(4), 723–760.
- Holme, P. (2005), Network reachability of real-world contact sequences, *Physical Review E*, 71(4), 46119.
- Ilcinkas, D. and Wade, A. (2011), On the Power of Waiting when Exploring Public Transportation Systems, *Proceedings of the 15th International Conference on Principles of Distributed Systems (OPODIS)*, pp. 451–464.
- Israeli, A. and Jalfon, M. (1990), Token management schemes and random walks yield self-stabilizing mutual exclusion, In *Proceedings of the 9th ACM symposium on Principles of distributed computing (PODC)*, pp. 119–131, ACM.
- Jain, S., Fall, K., and Patra, R. (2004), Routing in a delay tolerant network, In *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 145–158.
- Jones, E., Li, L., Schmidtke, J., and Ward, P. (2007), Practical routing in delay-tolerant networks, *IEEE Transactions on Mobile Computing*, 6(8), 943–959.
- Kellett, M. (2012), Black hole search in the network and subway models, Ph.D. thesis, University of Ottawa.

- Kossinets, G., Kleinberg, J., and Watts, D. (2008), The structure of information pathways in a social communication network, In *Proceedings of 14th International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 435–443, Las Vegas, USA: ACM.
- Kostakos, V. (2009), Temporal graphs, *Physica A*, 388(6), 1007–1023.
- Kuhn, F., Lynch, N., and Oshman, R. (2010), Distributed computation in dynamic networks, In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pp. 513–522, Cambridge, USA: ACM.
- Lindgren, A., Doria, A., and Schelén, O. (2003), Probabilistic routing in intermittently connected networks, *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3), 19–20.
- Litovsky, I., Métivier, Y., and Sopena, E. (1999), *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, pp. 1–53.
- Lynch, N. (1989), A hundred impossibility proofs for distributed computing, In *Proceedings of the 8th annual ACM Symposium on Principles of distributed computing (PODC)*, pp. 1–28, ACM.
- Marchand de Kerchove, F. and Guinand, F. (2012), Strengthening Topological Conditions for Relabeling Algorithms in Evolving Graphs, Technical Report Université du Havre.
- Mazurkiewicz, A. (1997), Distributed enumeration, *Information Processing Letters*, 61(5), 233–239.
- Miltersen, P., Subramanian, S., Vitter, J., and Tamassia, R. (1994), Complexity models for incremental computation, *Theoretical Computer Science*, 130(1), 203–236.
- Naor, M. and Stockmeyer, L. (1993), What can be computed locally?, In *Proceedings of the 25th annual ACM symposium on Theory of computing*, pp. 184–193, ACM.
- O’Dell, R. and Wattenhofer, R. (2005), Information dissemination in highly dynamic graphs, In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pp. 104–110, Cologne, Germany: ACM.
- Panaite, P. and Pelc, A. (1999), Exploring unknown undirected graphs, *Journal of Algorithms*, 33(2), 281–295.
- Patnaik, S. and Immerman, N. (1997), Dyn-FO: A Parallel, Dynamic Complexity Class, *Journal of Computer and System Sciences*, 55(2), 199–209.
- Ramanathan, R., Basu, P., and Krishnan, R. (2007), Towards a formalism for routing in challenged networks, In *Proceedings of 2nd ACM Workshop on Challenged Networks (CHANTS)*, pp. 3–10.
- Santoro, N. (2007), Design and analysis of distributed algorithms, Vol. 509, Wiley Online Library.

Shannon, C. (1951), Presentation of a maze-solving machine, In *Proceedings of the 8th Conference of the Josiah Macy Jr. Foundation (Cybernetics)*, pp. 173–180.

Suomela, J. (2011), Survey of local algorithms, *ACM Computing Surveys*. *To appear*.

Weber, V. and Schwentick, T. (2007), Dynamic Complexity Theory Revisited, *Theory of Computing Systems*, 40(4), 355–377.

Yamashita, M. and Kameda, T. (1996), Computing on anonymous networks: Part I and II, *IEEE Trans. on Par. and Distributed Systems*, 7(1), 69 – 96.

Zhang, Z. (2006), Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges, *IEEE Communications Surveys & Tutorials*, 8(1), 24–37.

This page intentionally left blank.

List of acronyms/abbreviations

DTN delay-tolerant network

MANET mobile ad hoc networks

TVG time-varying graph

UDG unit disk graph

This page intentionally left blank.

DOCUMENT CONTROL DATA		
(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)		
1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) University of Ottawa School of Electrical Engineering and Computer Science 800 King Edward Avenue Ottawa, Ontario K1N 6N5	2a. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED	
	2b. CONTROLLED GOODS (NON-CONTROLLED GOODS) DMC A REVIEW: GCEC JUNE 2010	
3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.) Deterministic algorithms in dynamic networks: Problems, analysis, and algorithmic tools		
4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.) Casteigts, A.; Flocchini, P.		
5. DATE OF PUBLICATION (Month and year of publication of document.) April 2013	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 88	6b. NO. OF REFS (Total cited in document.) 85
7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.) Contract Report		
8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.) Defence R&D Canada – Ottawa 3701 Carling Avenue, Ottawa ON K1A 0Z4, Canada		
9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 15by03	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) W7714-115111/001/SV	
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Ottawa CR 2013-021	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)	
11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.) <input checked="" type="checkbox"/> Unlimited distribution <input type="checkbox"/> Defence departments and defence contractors; further distribution only as approved <input type="checkbox"/> Defence departments and Canadian defence contractors; further distribution only as approved <input type="checkbox"/> Government departments and agencies; further distribution only as approved <input type="checkbox"/> Defence departments; further distribution only as approved <input type="checkbox"/> Other (please specify):		
12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)		

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The number of telecommunication networks deployed in a dynamic environment is quickly growing. This is true in our everyday life (e.g., smartphones, vehicles, or satellites) as well as in the military context (e.g., dismounted soldiers or swarms of UAVs). Unfortunately, few theoretical tools enable, to date, the study of dynamic networks in a formal and rigorous way. As a result, it is hard and sometimes impossible to guarantee, mathematically, that a given algorithm will reach its objectives once deployed in real conditions. Having such guarantees would seem to be crucial in a military context. In a previous report we identified a collection of recent theoretical tools whose purpose is to model, describe, and leverage dynamic networks in a formal way. This report focuses on problems, algorithms, and analysis techniques. We review recent efforts towards the design and analysis of distributed algorithms in dynamic networks, with an emphasis on those results that are of a deterministic and analytical nature. The topics include a discussion on how mobility impacts the very definition of problems; a set of generic tools to be used in the design or analysis of dynamic network algorithms; a discussion on the impact of various types of dynamics on the feasibility and complexity of given problems; a classification of dynamic networks based on dynamic graph properties; and finally, a discussion on how real-world mobility contexts relate to some of these classes.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

dynamic networks
wireless networks
vehicular ad hoc networks
VANET
mobile ad hoc networks
MANET
time-varying graphs
algorithms
deterministic algorithms

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale



www.drdc-rddc.gc.ca