Thèse de doctorat
de l'Université Sorbonne Paris Cité
Préparée à l'Université Paris Diderot

Ecole doctorale Sciences Mathématiques de Paris Centre (ED 386)
Institut de Recherche en Informatique Fondamentale (IRIF)

Spécialité : Informatique

Local certification in distributed computing: error-sensitivity, uniformity, redundancy, and interactivity

Par :
Laurent Feuilloley

Dirigé par Pierre Fraigniaud

Soutenue publiquement le 19 septembre 2018 *devant le jury constitué de :*

Jérémie Chalopin,	CR	Université d'Aix-Marseille	*Examinateur*
Bernadette Charron-Bost,	DR	École polytechnique	*Examinatrice*
Carole Delporte,	PU	Université Paris Diderot	*Examinatrice*
Pierre Fraigniaud,	DR	Université Paris Diderot	*Directeur de thèse*
Nicolas Hanusse,	DR	Université de Bordeaux	*Examinateur*
Sophie Laplante,	PU	Université Paris Diderot	*Examinatrice*
Boaz Patt-Shamir,	PU	Tel Aviv University	*Rapporteur*
Franck Petit,	PU	Université Pierre et Marie Curie	*Rapporteur*

Abstract.

This dissertation is about local certification, a central topic in distributed decision, a subfield of distributed computing. The distributed decision mechanism consists, for the nodes of a network, in deciding in a distributed manner whether the network is in a proper configuration or not, with respect to some fixed predicate. This decision is said to be local because the nodes of the network can communicate only with their neighbours. After communication, every node outputs a decision, stating whether the network is locally correct, that is, correct given the partial information gathered so far by this node. The network is declared to be globally correct, if and only if, it is declared to be locally correct by every node.

Most predicates cannot be verified by this type of computation, due to the locality constraint. Local certification is a mechanism that enables to circumvent this difficulty, and to check any property. It consists in providing the nodes of the network with labels, called certificates, that can be verified locally by a distributed algorithm. A local certification scheme is correct if only the networks that satisfy the predicate can be certified. In addition to its theoretical appeal, as a form of distributed non-determinism, the concept of local certification is especially relevant in the study of fault-tolerant distributed algorithms, where a key step consists in checking the status of the network, based on information stored at the nodes.

This dissertation deals with four aspects of local certification: error-sensitivity, uniformity, redundancy, and interactivity. The study of these four topics is motivated by the same essential question: How to reduce the resources needed for certification, and/or ensure a better fault-tolerance? In order to tackle this question we have to understand fundamental properties of certification. In particular, in this dissertation we answer questions such as: How redundant the certificates need to be for a proper certification? Are the classic certification protocols robust to a strengthening of the acceptance condition? and, How does introducing interactivity in the process changes the complexity of certification?

Keywords: Distributed network computing, distributed decision, local certification, proof-labelling scheme, fault-tolerance.

Résumé.

Cette thèse porte sur la notion de certification locale, un sujet central en décision distribuée, un domaine du calcul distribué. Le mécanisme de la décision distribuée consiste, pour les nœuds d'un réseau, à décider de manière distribuée si le réseau est dans une configuration correcte ou non, selon un certain prédicat. Cette décision est dite locale, car les nœuds du réseau ne peuvent communiquer qu'avec leurs voisins. Après avoir communiqué, chaque nœud prend une décision, exprimant si le réseau est correct ou non localement, c'est-à-dire correct étant donné l'information partielle récoltée jusque-là. Le réseau est déclaré correct globalement s'il est déclaré correct localement par tous les nœuds.

Du fait de la contrainte de localité, peu de prédicats peuvent être vérifiés de cette manière. La certification locale est un moyen de contourner cette difficulté, et permet de décider tous les prédicats. C'est un mécanisme qui consiste à étiqueter les nœuds du réseau avec ce que l'on appelle des certificats, qui peuvent être vérifiés localement par un algorithme distribué. Un schéma de certification locale est correct si seuls les réseaux dans une configuration correcte peuvent être certifiés. L'idée de la certification locale est non seulement séduisante d'un point de vue théorique, comme une forme de non-déterminisme distribué, mais c'est surtout un concept très utile pour l'étude des algorithmes tolérants aux pannes, où une étape-clé consiste à vérifier l'état du réseau en se basant sur des informations stockées par les nœuds.

Cette thèse porte sur quatre aspects de la certification locale : la sensibilité aux erreurs, l'uniformité, la redondance et l'interactivité. L'étude de ces quatre sujets est motivée par une question essentielle : comment réduire les ressources nécessaires à la certification et/ou permettre une meilleure tolérance aux pannes? Pour aborder cette question, il est nécessaire de comprendre le mécanisme de certification en profondeur. Dans cette optique, dans cette thèse, nous apportons des réponses aux questions suivantes. À quel point les certificats doivent-ils être redondants, pour assurer une certification correcte? Les schémas de certification classiques sont-ils robustes à un changement de la condition de correction? Le fait d'introduire de l'interactivité dans le processus change-t-il la complexité de la certification?

Mots-clefs: Calcul distribué sur réseau, décision distribuée, certification locale, schéma d'étiquetage de preuve, tolérance aux pannes.

Aknowledgments.

This thesis has only one author, but many people participated in this project, and it is time to thank them. I would first like thank my advisor, Pierre Fraigniaud, for his wise and cheerful guidance, and for giving me both the freedom and the support that I needed. Also I would like to thank Pierre for constantly reminding me that clarity is the key in scientific writing. I hope this thesis shows that I am trying to follow this mantra!

I am grateful to Boaz Patt-Shamir and Franck Petit for reviewing this document and giving a nice feedback. Their reviews made me improve the manuscript in various ways. My sincere thanks to Jérémie Chalopin, Bernadette Charron-Bost, Carole Delporte, Nicolas Hanusse and Sophie Laplante, for accepting to participate in my thesis committee.

I would like to thank the researchers (and friends) who co-authored the papers on which this thesis is built, and who shared my office for some time: Juho Hirvonen (we have already four papers together, I hope others will follow!), Ami Paz and Mor Perry (one paper is the first step, let's continue!).

My life in research is certainly not restricted to the content this manuscript. First, my PhD would not have been the same without its prelude: various internships with nice and talented people. I would like to warmly thank Marie-France Sagot, José Correa, Jukka Suomela, and Ola Svensson, for giving me the opportunity to work with them (and their colleagues and students), and for making me discover the world of research. Also I express my gratitude for the teachers I had along the years that lead to this thesis. Then I thank my colleagues who did not participate directly to the works presented in this dissertation, but were very important anyway, through numerous discussions. I also learned a lot from the teaching, and I thank the students, the professors, and the UFR staff for this. Special greetings to my fellow PhD seminar organizers, Clément and Théo (I think we did a good job!), and to my successor Baptiste. Thanks to the speakers of the seminar, especially the ones who participated in the semidoc blog! And thanks to Victor for accepting to take care of the blog. On a more general perspective, the research and teaching community, and the staff around, are essential to any scientific work, including this one.

During my three years at IRIF, an indispensable breath of fresh air came from the CROUS lunches and endless pause-cafés, with wonderful friends from the fourth and (more recently) the third floors. I do not want to list them, they know who they are! I am also grateful to the great people outside the lab who made my life better.

Enfin, un merci immense et ému à mes parents, Georges et Bénédicte, à mon frère Éric, et à Julie. Merci d'être là, tout simplement.

Structure.

This document begins with a fairly long introduction that surveys the results and techniques of both the previous work and the thesis. A more complete review of the previous work can be found at the end of this document, in Chapter 7, which is a survey of the literature. After a chapter about the model and the definitions (Chapter 2), the main technical contributions of the thesis are developed in Chapters 3, 4, 5 and 6. These core chapters correspond to four papers that are either published or accepted for publication. They are structurally quite close to the original versions, but the introductions have been changed and some proof sketches and figures have been added. Chapter 8 is the conclusion, with open problems and perspectives.

Contents

Chapter 1

Introduction

This introduction has four parts. The first part is an introduction to distributed decision, the subfield of distributed computing this thesis is concerned with. The second part is an overview of the results and techniques from the literature that are essential for what follows. The third part reviews the contributions of the thesis. The fourth part is an annotated list of my works in (or related with) distributed computing.

1.1 Introduction to distributed decision

This thesis is about *distributed decision*, a subfield of network distributed computing. This section is an introduction to this domain. We present informally the essential concepts in a narrative style, keeping just the key elements, and avoiding citations in purpose. Discussions of the original papers and of some technical points can be found in the last two subsections, *Historical notes* and *Complements*. A broader, more systematic and exhaustive review of the domain can be found towards the end of this document, in the form of a survey of the literature (Chapter 7).

1.1.1 Distributed computing and local algorithms

A domain of distributed computing. Distributed computing is the field of computer science that studies the systems in which several entities communicate in order to complete a task, without central control. These entities can be agents in a networks, animals, robots etc., but we will simply consider machines. There are plenty of models to study such setting, but one can distinguish two main challenges: asynchrony, and locality. *Asynchrony* is a problem of time. Entities may not have the same notion of time, may compute at different speed, and may crash. *Locality* is a problem of space. The machines may be far apart, and it may take a long time to communicate. Thus in general, a machine would not get any information from a far-away machine, and still it has to compute something meaningful globally. These two challenges are usually decoupled, and this thesis is focused on the second one, that is, locality.

Modelling networks as graphs. We consider networks of computing machines, modelled as *graphs*. The vertices of the graph represent the machines of the network, and the edges represent communication links. This (connected) graph is the *communication graph*. A concrete example would be to model a set of sensors with radio communication, by assigning a node for each sensor, and adding an edge between two sensors that are close enough to communicate by radio one with other.

Locality. The natural model of distributed algorithms on graphs uses *messages*, that the nodes can send and receive in rounds. We use a more abstract model with the nodes having a *partial view* of the graph. These are equivalent, as we show in the later Subsection 1.1.6. We consider that every node can see the subgraph induced by the nodes at some distance t from itself, and has to output a relevant value, only based on this knowledge. That is, every node has a partial view of the graph, and its output is independent of the parts of graph that are outside this view. See Figure 1.1. An algorithm in this model may be called a *local algorithm*. The smaller the radius of the view, the more local the algorithm. The constraint of not being able to communicate with nodes that are far away can be called the *locality constraint*, and the theory that focus on this constraint is usually refered to as the *study of locality*.

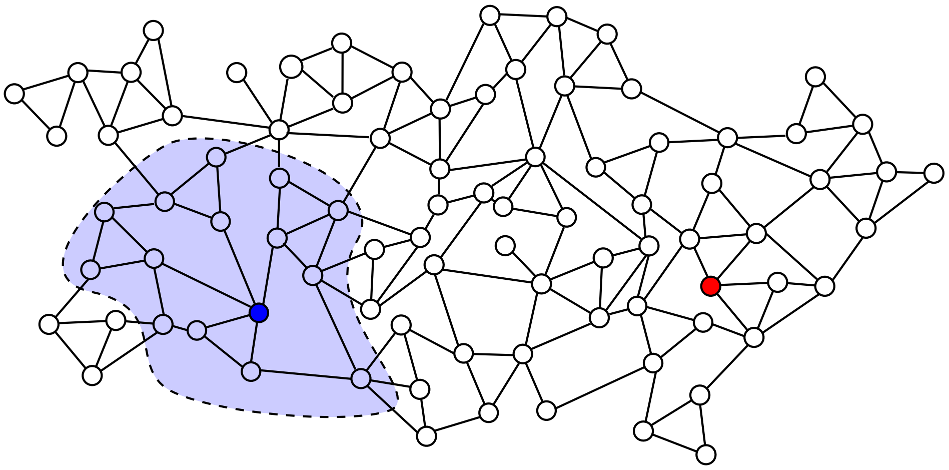


Figure 1.1: In a local algorithm, every node has a local view. For example the dark blue node can see only the node in the light blue area. In particular it has no knowledge about the red node: the output of the blue node on this graph, or on the graph where we have changed the red node in some way, should be the same.

A typical question about locality is: can every node output a colour such that the colouring has some global property, if we impose that every node can only see at a bounded distance? Typical research in the field consists in designing local algorithms for computing such a colouring with the smallest possible views. Remark that the graph is both the input (it is this one graph that we want to colour) and the structure that defines the neighbourhoods of the nodes. The approach we take here is a bit different from this colouring question, in the sense that we do not seek to construct a solution to a problem. Instead we want to check whether the current solution is correct. That is, we consider *decision problems* and not construction problems.

From construction to decision. In centralized computing, decision problems are the foundation of the theory of complexity. The current decade has seen efforts to define a theory of complexity for distributed computing, based on a distributed version of decision problems. In a problem of distributed decision, the input is the communication graph, with some additional labels on the nodes and edges, and the machines have to collaborate for deciding whether this input corresponds to a proper configuration with respect to a fixed predicate. For example, in a setting where every node is given a colour, we want to check that no edge has the same colour on both endpoints. The way the nodes take a global decision will be specified later, but it is based on a local process: every node will just have access to a small view of the graph.

Motivations for a new field. Motivations for the study of distributed decision range from practical to theoretical. On the practical side, deciding whether a configuration of the network is in a proper state is a natural primitive when the network is subject to faults. Focusing on this kind of problems is useful as a building block for more complex systems (concrete examples will come later in this section). On a more theoretical point of view, distributed decision gives insights about distributed construction (we will see an example in the paragraph about probabilistic decision). Nonetheless the links between construction and decision are not as tight as in centralized computing, which is why distributed decision is a field in itself. Another motivation comes from the division we have mentioned earlier, between asynchrony and locality in distributed computing. It is unfortunate that the two aspects are so rarely tackled together. A concrete reason why this happens is that the algorithmic problems studied in these two subfields are of very different nature. For example, locality-oriented research would typically focus on combinatorial optimization problems, while the asynchrony-oriented research will typically focus on reaching some form of consensus between the processes. An expectation of distributed decision is that checking whether some configuration satisfies some predicate is general enough to fit in both frameworks. This thesis will only deal with the locality perspective, but we give in Subsection 1.1.6 a few pointers to papers aiming at bridging the gap.

How to decide when there are several machines. Decision in centralized computing is easy to define: there is one input to consider, and one decision to make, *accept* or *reject*. In distributed computing, there are several machines, thus different partial views of the input, and different outputs. Every node will output a local decision and those local decisions will be aggregated to define a global decision. For the aggregation of the decisions, a natural choice is to accept a configuration if, and only if, all the nodes locally accept, and this is the choice for this thesis. Other aggregation mechanisms have been studied, but the literature mostly focuses on this one, because of its simplicity and its practical relevance. Indeed, in a practical scenario, every node is checking regularly its neighbourhood, and that if something looks wrong, then it raises an alarm or launch a recovery procedure that resets the whole network.

Languages and examples. So far, we have informally described decision as the task of determining whether the graph is in a correct state with respect to some predicate. Usually, in distributed decision, one considers *languages*, just like in centralized computing. A language is a set of *configurations*, that is a set of graphs whose nodes (and more rarely edges) can be labelled. The labels can be called *inputs*. The instances in the language may be called *yes-instances* and the others *no-instances*. Examples of languages are: the graphs that *can be* properly coloured with some given number of colours (this language has empty input labels), graphs that *are* properly coloured (the inputs are colours), graphs that are trees, etc.

Identifiers. We will always assume that the nodes have distinct *identifiers* (or *ID* for short), that is, every node is given a number, and these numbers are different. This will be detailed and justified later. An important point is that a language should not refer to the identifiers, for example a set of graphs where the node with identifier 1 has some special property, is not a language.

1.1.2 Basic decision and its limits

Basic decision mechanism. The basic decision mechanism for a given language is deterministic. Every node gathers its neighbourhood at some distance, and then decides to accept or reject based on this view. More precisely, for a given language, a basic scheme consists in a local algorithm, that first gathers the neighbourhood at some fixed constant distance t (that is, it gets a snapshot containing the structure of the graph, the identifiers, and the inputs of the nodes within this radius t), and then decides to output *accept* or *reject*. This algorithm is the same at every node. See Figure 1.2 for a concrete example.

Limits: acyclicity. Now one can try to define a local algorithm for checking that the communication graph is a tree, that is, checking the *acyclicity* of the underlying graph. This is actually impossible, that is, there is no such algorithm. The high-level reason for this is that, locally, one cannot distinguish between a cycle and a path. See Figure 1.3 for an illustrated proof of this result. The type of reasoning used in the proof is common in distributed computing, and is called the *indistinguishability technique*.

The main way to cope with this limitation of the basic schemes is to consider non-determinism, which we will describe soon. But let us do a small detour to take a look at the probabilistic side.

1.1.3 Probabilistic decision

Allowing some randomness. In the light of the great power of randomness for distributed graph algorithms, one is tempted to ask whether we can decide more

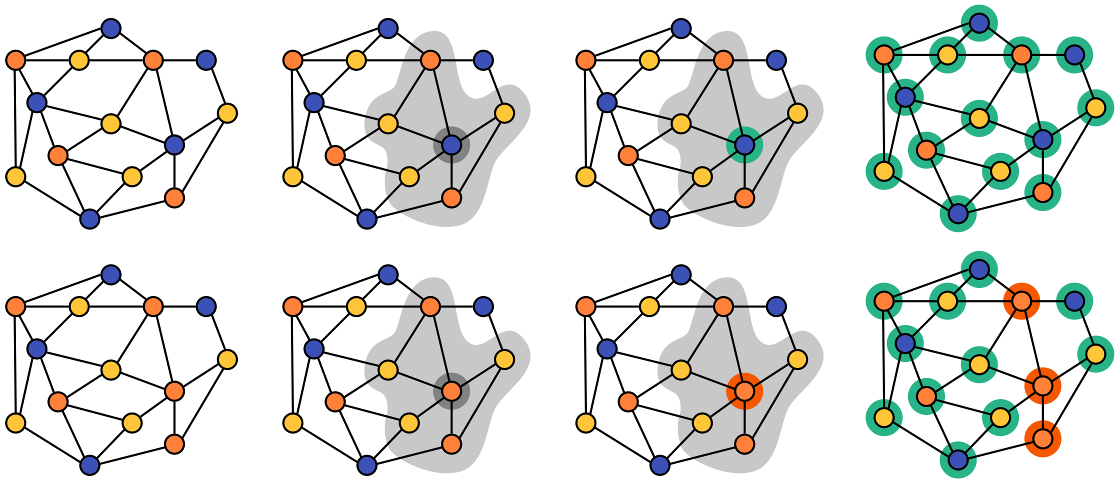


Figure 1.2: The figure illustrates the basic decision scheme for this language of *properly coloured graphs* (Subsection 1.1.2). The local algorithm of this scheme is simple: every node gets a view at distance 1, and rejects if and only if one of its neighbours has the same colour as its own. The first raw describes the behaviour of the algorithm on a *yes*-instance (that is on a graph that is properly coloured). For this instance, the node that is highlighted is blue, and its neighbours are either yellow or orange, thus it accepts. Actually no node has a neighbour with the same colour, thus every node accepts, and the instance is (globally) accepted. The second raw illustrates the behaviour of the local algorithm on a *no*-instance. Here the highlighted node has colour orange and has two orange neighbours, thus it reject. This automatically lead to the (global) rejection of the configuration.

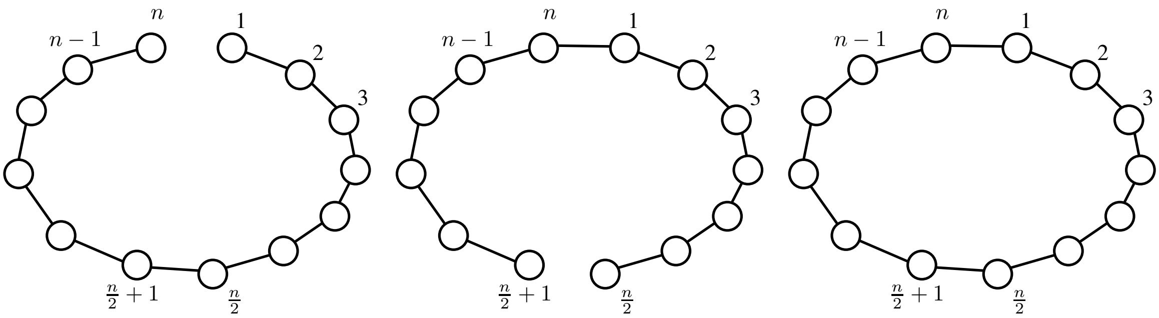


Figure 1.3: This figure illustrates the fact that some languages cannot be decided by a basic decision scheme (Subsection 1.1.2). The language studied is the set of *trees*. Consider the two paths on n vertices of the two first pictures. They only differ by their identifier assignments. The first path has identifiers from 1 to n in increasing order. The second one has identifiers from $n/2 + 1$ (suppose n is even) up to n, and then from 1 to $n/2$. Both are *yes*-instances (because paths are trees). Thus if there exists a correct local algorithm to decide the language, then it should accept on all nodes, in both instances. Now consider a cycle where the identifiers go from 1 to n and then loop, as in the third picture. This is a *no*-instance, thus the local algorithm should reject on at least one node. Consider some node where it rejects, and the view of this node. By construction this view also appears in one of the two paths we mentioned before. This means that with exactly the same view, the local algorithm is accepting in the path and rejecting in the cycle. This is impossible because the algorithm is deterministic and cannot distinguish between the two instances. Thus a basic decision is not enough for this language.

languages if we allow the local algorithm to use random bits. In a similar way as in centralized computing (or, more precisely, as in the complexity class BPP), a scheme will be considered correct if, on *yes*-instances, there is a good probability of acceptance, and on *no*-instances, there is a good probability of rejection. By *good probability*, we mean larger than some fixed constant. With this definition, randomness does help. In particular, it provides the ability to basically count up to a constant, as the following example shows.

At most one node selected. In the language we consider, every node has an input bit stating whether it is selected or not. The labelled graphs in the language are the ones that have at most one selected node. The language is called *At most one selected*, or AMOS for short. Having at most one selected node is a global property, in the sense that two far away nodes cannot detect that they are both selected, thus no basic decision scheme can decide this language. Nevertheless, with randomness, it is decidable, and it does not even require the nodes to have any knowledge of the graph, except their own input. The trick is the following. The algorithm consists simply in stating that every unselected node accepts, and every selected node accepts with some probability p. If there are no selected node then the probability that every node accepts is 1. If there is just one selected node, then it is p. If two or more nodes are selected, the probability that all nodes accept is at most p^2. This scheme is correct if we consider the following decision thresholds. If an instance is in the language, we require that the nodes accept with probability at least p. If an instance is not in the language, we require that the nodes reject with probability at least $1-p^2$. (Note that this makes sense only for p inside some interval.)

Impact on construction tasks. The study of probabilistic decision has led to an improvement in the understanding of construction algorithms that is worth stressing here, as such new light is also a motivation for the field. It has long been known that randomness does not help for construction tasks if: (1) the solutions form a language that can been checked locally (that is, checked with a basic decision mechanism) (2) we consider only constant-time construction. This result has been extended to more problems, by proving that the first condition can be lifted from languages decided with *basic decision*, to languages decided with *probabilistic decision*.

1.1.4 Non-determinism

The core topic of this thesis is non-determinism in distributed decision. Let us first recall the form of non-determinism in centralized computing.

Non-determinism in centralized computing. We will do an analogy between our model and the well-known complexity class NP. The class NP was originally defined with non-deterministic Turing machines, but the modern definition is often more handy. A decision scheme for NP is based on two entities: the machine, called the *verifier*, and an external oracle, called the *prover*. It follows the following sequence of steps. First, given the instance, the prover provides a certificate of polynomial size to the verifier. Second, the verifier runs a polynomial-time algorithm that takes as input both the instance and the certificate. Third, the verifier outputs a decision, *accept* or *reject*. Such a scheme is correct if the following condition holds for every instance: the instance is in the language, if and only if, there exists a certificate such that the verifier accepts. This can be pictured in a less formal way as: an omniscient prover trying to make the limited verifier accept, regardless of whether or not the instance belongs to the language. The verifier has to distinguish between the cases in which the prover is honest, and the cases where it is dishonest.

Proof-labelling schemes. The classic form of non-determinism in distributed decision is called a *proof-labelling scheme*.[1] In such a scheme, there is a prover and every node acts as a verifier. The prover provides a local *certificate*, also called *proof*, to each vertex. In general, these certificates are all different. In this new framework, the view that a node gathers, includes the certificates of the nodes within the view; in addition to the structure of the graph, the identifiers and inputs, as in a basic scheme. The local decisions are aggregated as before. The acceptance rule is similar to the one of NP, that is, for every instance: the configuration is in the language if and only if there exists an assignment of certificates that makes every node accept. The usual way to describe a scheme is to describe the way the prover crafts certificates on *yes*-instances, and then to explain how the nodes can distinguish between certificates of good instances, and certificates designed to mimic those on bad instances.

A concrete example: acyclicity. Let us go back to the example we used to highlight the limitations of basic deterministic decision in Figure 1.3, the set of trees. We describe a proof-labelling scheme for this language. The strategy for the prover on a *yes*-instance (that is on a tree) is to pick an arbitrary node, the *leader*, and to give to every node, as a certificate, its distance to the leader. The local verifier algorithm used by the nodes is the following. The node considers the distance d written on its certificate, and accepts if and only if, either (1) $d = 0$ and all the neighbours have distance 1, or (2) it has exactly one neighbour with distance $d-1$ and the other neighbours have distance $d+1$. It is easy to be convinced that on *yes*-instances, the verifier algorithm outputs accept on every node. It then requires a proof to be convinced that, if there exists a cycle in the graph, then there is always one node to catch an inconsistency, regardless of the certificate assignment the prover has designed. A sketchy proof of this fact is the following. Consider a cycle in the graph, and consider the certificates assigned by the prover for the nodes of this cycle. The verifier understand these certificates as numbers. Consider the node that is assigned the largest number in the cycle. This node has two neighbours with equal or smaller numbers. This matches none of two cases we have described above. Thus this node rejects, and the configuration is globally rejected. See Figure 1.4, for a similar scheme designed for a different, but similar, language: acyclicity of a selected subgraph.

[1] In this paragraph, we give a general definition of proof-labelling schemes, see the *Historical notes* in Subsection 1.1.5, for the original (more restricted) definition.

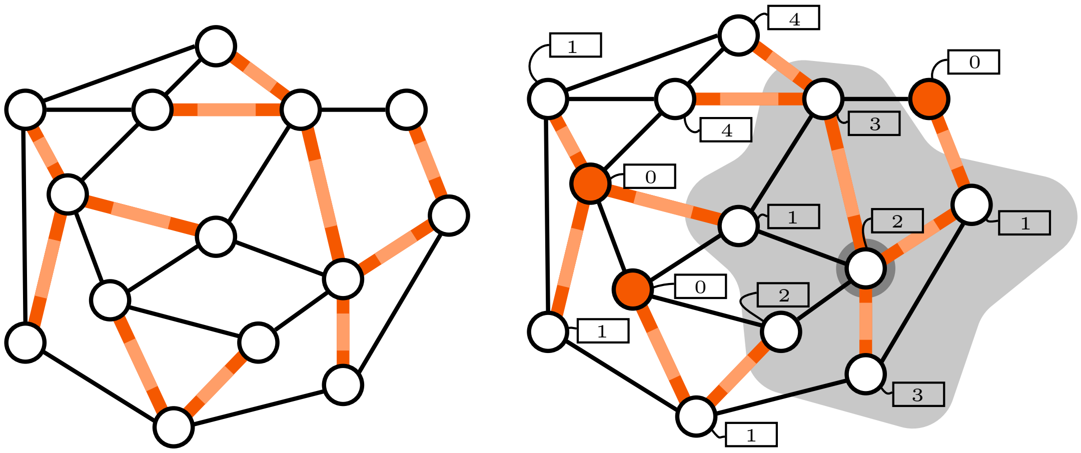


Figure 1.4: In the text, we have used the set of trees to illustrate the notion of proof-labelling scheme. It is more common to consider properties of a labelling of the graph, instead of properties of the graph itself. Here, this translates into having an edge labelling describing a set of selected edges, and deciding whether the subgraph defined by these edges is acyclic. The picture on the left, where the selected edges are coloured in orange, is an example of *yes*-instance for this language. The picture on the right illustrates the certificate assignment given by the prover, and the view of a particular node. The strategy for the prover on *yes*-instances consists in first choosing a leader in each tree. This choice is represented by the orange nodes in the picture on the right. Then the prover provides every node with its distance to the leader of its tree. The verifier simply checks that the distances are consistent along the selected edges, just as in the scheme for trees. For example, here, the highlighted node has label 2, which is consistent with the numbers given to its neighbours in the subgraph: one of them has distance 1, and the others have distance 3.

Link with self-stabilizing algorithms. Proof-labelling schemes can be described as local non-determinism inside distributed decision, but this notion was introduced before the field of distributed decision was formally created. They were considered first as a way to study some specific parts of so-called *self-stabilizing* algorithms. Basically, a (construction) algorithm is self-stabilizing if, starting from an arbitrary state of the network, it can reach a proper configuration (with respect to a given language). At any time, such an algorithm has to check that the configuration is correct, and if it is not the case, then it has to fix it. For example, consider an algorithm whose role is to select an acyclic subset of edges (as in Figure 1.4). Suppose it has done so, but then some register is corrupted, a new edge is selected, and the addition of this edge creates a cycle. The algorithm has to detect that the configuration has changed, and that it is not correct any more, and has to reach a new acyclic configuration quickly. An essential step in this scenario is to detect that something is wrong, and this is of course where decision appears. For acyclic subgraph, for example, we know from Figure 1.3 that we cannot decide the correctness locally. A strategy to cope with this is to write some information at the nodes during the computation. That is, not only does the algorithm build an acyclic labelling, but it also mark the distances at the nodes. This way, if the graph is modified in a way that does not match the information available at the nodes, then the nodes can detect it, and launch a recovery procedure.

Considering proof-labelling schemes is a way to focus on the fault detection phase. In this framework, the additional information left at the nodes is abstracted as an external prover.

Proof sizes and measure of the locality. An essential aspect of a proof-labelling scheme is the size of the certificates used, that we call the *proof size*. The first reason for this importance is practical: small certificates are better, because in applications they use less resources. The second reason is more theoretical. If we consider only the (deterministic) basic decision mechanisms, then we can easily define a dichotomy between local and global languages. A local language would be a language for which there exists such a mechanism (as the language of properly coloured graphs in Fig. 1.2) and a global language would be a language such that no such mechanism exists (such as the set of trees in Fig. 1.3). The study of proof sizes allows a more fine-grain analysis. Remember that the certificates are assigned by an oracle that has full knowledge of the graph, thus they are, in some sense, pieces of global information given to the nodes. From this perspective, the larger the certificates need to be, the more global information we have to give to the nodes to decide, and thus the more global the language. The optimal size of the proofs for a language is then a measure (or, more rigorously, an inverse measure) of its locality. Note that this definition extends the first rough definition of local/global we had: languages that can be decided with a basic decision mechanism have in some sense proof size zero, and languages that cannot be decided this way, need non-zero proof size.

1.1.5 Historical notes

This subsection is a personal and partial historical view of the field. Reader not interested in history or citations may skip this part. This text is not meant to be exhaustive, see the survey for the citations of all the works of the domain.

Study of locality and locally checkable labellings. The model of network distributed computing dates back to the 1990s with the papers of Linial [87], and Naor and Stockmeyer [89]. A fundamental book on the topic, by Peleg, appeared in 2000 [95]. In their paper, Naor and Stockmeyer define a class of (construction) problems called *locally checkable labelling*, LCL for short. For these problems, the nodes may have inputs, and they have to compute outputs. A problem is in LCL if there exists an algorithm that, given a view at some constant distance, with the inputs and outputs, can decide whether the solution computed is correct. Also for LCL problems, inputs, outputs and the degrees of the graphs are assumed to be constant. This class includes many classic problems such as computing colourings, independent sets, matchings etc. Note that the definition of LCL implies that the correctly labelled instances form a language that can be decided via a basic decision mechanism.

Proof-labelling schemes. Proof-labelling schemes were introduced by Korman, Kutten and Peleg, in 2005 (see [81] for the conference version, and [82] for the 2010 journal version). The definition of a proof-labelling scheme in this original paper is a bit different from the one we gave in Subsection 1.1.4. The authors of [81, 82] consider that the nodes are exchanging messages instead of having a view, but more importantly, that the only communication a node can perform is to send its certificates to its neighbours.

Locally checkable proofs. To sum up, in the original definition of proof-labelling schemes, a node only knows the following: its degree, input, and identifier, and its certificate and the ones of all its neighbours (meaning the nodes it is adjacent to). In [66, 67], Göös and Suomela, used the terms *Locally checkable proofs*, to describe proof-labelling schemes, in the general form we have described before. This implies allowing arbitrary constant radius for the view, and accessing all the inputs, IDs and certificates within this view. In later works, the term "proof-labelling schemes" refers to the original model, or to locally checkable proofs. The two cases are pretty different when it comes to lower bounds, as in the second definition (that is the one used in this thesis) the nodes have more information about the graph, which implies that it is harder for the prover to fool them.

Distributed decision as a subfield of distributed computing. In 2012-2013, Fraigniaud, Korman and Peleg argue in favour of a complexity theory for distributed decision [54]. This was followed by a handful of papers studying randomization in this context, or on the impact of the model of identifiers. Lately several groups have studied various aspects of proof-labelling schemes. Let us cite some relevant recent works, excluding the ones directly related to this thesis, that will be mentioned later. In [24], Censor-Hillel, Paz and Perry study the interplay between approximation and non-determinism. In [14] and [94], Fraigniaud, Perry and Patt-Shamir focus on message communication during the verification (with the original definition of proof-labelling scheme). The first paper deals with randomization, while the second deals with the type of communication at hand (broadcast versus unicast). Finally, proof-labelling schemes can be seen as certification mechanism, and Fraigniaud and Balliu [9] use them to certify routing tables.

1.1.6 Complements

In this subsection we answer (or give pointers that answer) some natural questions, that the previous sections may have raised. In particular this subsection provides some indications about topics that are at the boundary of the current work. These complements are not essential for the rest of the text, therefore they may also be skipped.

On representing networks by graphs. We have modelled networks as graphs, and this will be the setting for the whole thesis. Readers may consider that this modelling is too rough. A more realistic model for wireless communication, for example, is the *SINR model*, that takes into account interference. Even such models are reasonably well approximated by simple graphs [71], thus it is reasonable to begin with graphs. Nevertheless distributed decision in more realistic models is an interesting avenue of research.

From messages to views. In the *Locality* paragraph of Subsection 1.1.1, we have mentioned the fact that our quite high-level model with views (let us call it the *view model*) is equivalent to the more realistic message-passing model. Let us first specify which message-passing model we consider, and then sketch the equivalence. In the message-passing model, computation proceeds in rounds. In the context of distributed decision, the number of rounds is usually constant. At each round every node can receive messages from its neighbours in the graph, compute new messages, and send them to its neighbours. The size of the messages is not bounded. We now sketch the proof of the equivalence between the two models, without mentioning the possible inputs, for simplicity. Let us first show that the message-passing model is at least as strong as the view model. At each round, every node can send, to all its neighbours, all the information about the graph it has. At the very beginning of the process, a node knows only its degree and its ID. It can send them to its neighbours. Then it receives the analogue messages from its neighbours, and is able to reconstruct the view at distance 1. The next step is to send this view at distance 1 to all its neighbours, and to receive their views at distance 1. [2] By merging these views, every node can reconstruct its own view at distance 2, etc. Thus, with basically t rounds every node can reconstruct its view at distance t. Now in the other direction, suppose a node has

[2] Note that at this step, if there is no bound on the degree, the messages used can be polynomially large in n. Thus this transformation does not work in the so-called CONGEST model, where the messages must have logarithmic size.

a view at some distance t. It can compute the first messages sent by the nodes in its $(t-1)$-neighbourhood, because it knows their IDs and degrees. Then, the node can compute which messages are received by the nodes in its $(t-2)$-neighbourhood. And so on and so forth, until it simulates (only) its own behaviour for the t-th round.

Relation between decision and construction. We have evoked the fact that decision and construction are not as closely related in distributed computing as in centralized computing. Let us consider colouring to exemplify this claim. In centralized computing, if one has access to an oracle deciding if some partial 3-colouring can be completed to a correct 3-colouring, then it is easy to build a construction algorithm of similar complexity. Indeed one can follow the following trial-and-error strategy. First ask whether the graph can be 3-coloured, if not, stop. Otherwise, select a node arbitrarily, give it an arbitrary colour, and ask whether this colouring can be completed. If the answer is positive, one can continue with a second vertex, otherwise one can try another colour, and ask again for approval etc. If it does not stop after the first step, this procedure produces a proper 3-colouring, and the running time is only multiplied by a linear factor. In distributed computing, it is not that simple. The main reason is that the process described above is inherently sequential, and does not correspond to the distributed setting. For example, for colouring, all the nodes have to choose their colours at the same time. Thus the fact that verifying a colouring is easy in distributed manner, does not give much information on the difficulty of computing a colouring.

Dealing with asynchrony. As said, distributed decision was partly justified as a path to gather together asynchrony and locality in distributed computing. We refer to results in that direction, or more precisely results about wait-free distributed decision, in Subsection 7.6.1 in the survey.

Other decision mechanisms. The decision mechanism studied in this thesis (accept globally, if and only if, every node accepts locally), is justified by practical scenarios, but from a theoretical perspective it relevant to investigate other mechanisms. For example, one can allow more bits of output, or different ways to gather the outputs into one decision. This has been investigated in [4] and [5]. See Subsection 7.3.3 in the survey.

Alarm scenarios. A practical motivation for our decision mechanism is the scenario where the nodes can raise an alarm. A reference about such decentralized monitoring is [16].

Indistinguishability technique. A proof technique that is essential in distributed computing, and in particular in this thesis, is the *indistinguishability technique.* In a distributed environment, the machines often have a partial knowledge of the current state of the system. Thus there may be several global configurations that, from the point of view of a specific machine, are exactly the same; then the behaviour of this machine must be the same in all these configurations. The indistinguishability technique consists in using this simple fact to show lower bounds. A usual reasoning is: if some quantity is not large enough, then there exists two instances that cannot be distinguished, although the task to solve requires the algorithm to have two different output, thus this quantity must be large. A reference on this technique (among others) is [43].

Randomness and difficulty of boosting. We have described probabilistic decision in Subsection 1.1.3. The paper that introduced the concept is [54]. In this paper, the schemes require some precise threshold for the decision, which is intriguing because in centralized computing these numbers are usually unimportant, as they can be boosted to obtain any (meaningful) number. The question of whether boosting is possible in the distributed setting was answered negatively in [56].

The result about derandomization mentioned in Subsection 1.1.3 is the main theorem of [35], and is an improvement over a result of [89].

Not only can randomness enable to decide more languages, but also it helps to decrease the message size, when considering the message-passing model. In [14] the authors show that one can gain an exponential factor in the size of the messages, by using some hashing techniques.

Role of the identifiers. So far, we have not used the identifiers a lot, but those are important. In the first place, identifiers are essential for the transformation from the message-passing model to the view model. Indeed without identifiers, it is impossible to reconstruct the view of a node, as we do not know if two nodes that appear in two different messages are different or not. Later in the text, identifiers will also be very useful to design schemes for some languages. For example, the prover may give the ID of a leader to all the nodes. Actually, there exists a parallel line of research that considers proofs that do not use IDs, that is, proofs that are correct whatever the ID-assignment is. In both our framework and this other framework, the languages studied do not depend on the identifiers.

The polynomial bound on the identifiers is a classic assumption in network distributed computing. On the one hand, it is small enough to allow to encode the identifiers on a logarithmic number of bits. On the other hand it is large enough to ensure that the numbers are distinct, and having identities between 1 and n would be too strong in some applications, and would allow the algorithm to abuse the power these identifiers.

The impact of the precise model of identifiers on decision can be a pretty subtle issue, as shown for example in [59]. See Subsection 7.4.1 in the survey.

Message size. In the papers that consider the message-passing models, the size of the messages is a key parameter. One can for example study the interplay between message and proof sizes. See Subsection 7.3.2 in the survey, for the basic and probabilistic decision in presence of congestion, and Subsection 7.4.2 for the non-deterministic analogue.

Weaker computational power. We have not mentioned any restriction on the computational power of the nodes of the network, and we actually do not bound it in general. This is because our focus is on the proof sizes, and on the locality of the verification. One result of the thesis needs the assumption that the language is computable (in a centralized manner), but in general we do not even need this assumption. Much more restricted models have been studied, where the nodes basically compute with finite automata. In this framework, a fruitful approach to use modal logic to characterize the languages that can be recognized. See for example the thesis [98], and the references therein.

Advice versus certificate. A concept close to local certificates is the notion of *advice*. An advice also comes as an assignment of labels to the nodes, given by an oracle, but this oracle can be trusted. That is, the information just helps for the computation, and does not need to be verified. This concept has been introduced to measure how much global information is needed to complete some task. See for example [50].

Differences with the self-stabilizing approach We have mentioned that proof-labelling schemes have been defined in the first place to study the fault detection phase of self-stabilizing algorithms, and that these two concepts share the same flavour. Yet, there are some aspects on which these models differ. On the one hand, in a self-stabilizing algorithm, the additional information required to check the correctness must be efficiently computable. This is not the case in proof-labelling schemes where the certificates are given by an external prover. On the other hand, a proof-labelling scheme is required to work for every configuration of the language, which may not be necessary for self-stabilizing algorithms. Let us illustrate this statement on the language of acyclic subgraphs. In a proof-labelling scheme, any acyclic subgraph must be properly decided. Whereas a self-stabilizing algorithm may always choose to output some constrained form of acyclic subgraph. This may make the certification easier for the algorithms, as it does not have to detect errors in arbitrary configurations.

Use in (centralized) parameterized complexity. The concept of local proofs is natural enough to ask whether it can be of any use in a centralized setting. It is actually useful in parameterized complexity. More precisely, a simple argument allows to transform automatically the existence of local proofs (of some type) into algorithms that are polynomial in the size of the graph, and single exponential in the treewidth. This mechanism works for example in the case of maximum independent set and vertex cover. The lower bounds established on local proofs can rule out the possibility of using this tool, which can in turn justify the use of other techniques [19].

Complexity theory for construction tasks. One of the original motivations for distributed decision was to have a complexity theory similar to the one of centralized computing. Indeed until 2010, network distributed computing was mainly about pushing further and further the bounds on particular problems (maximal matching, colouring, spanning tree constructions etc.), and not on putting these problems into neat classes. While distributed decision was growing, research has also been done to classify the problems of constructing LCL languages (as described in Subsection 1.1.5). For example, it has long been known that a whole body of problems (such as $(\Delta+1)$-colouring and maximal independent set), can be solved in time $O(\log^* n)$, whereas for some other problems the best known algorithms had complexities in $\Omega(\log n)$. It was unknown whether there exist problems in between. In [25], it is proved that there is a gap between the two types of problems, thus defining classes of complexity for LCL. See [11] for a recent paper on this line of research.

1.2 Proof-labelling schemes toolbox

This section is a toolbox of techniques and results used in the investigation of proof-labelling schemes. It takes the form of questions and answers, with proof sketches. These techniques will be used in the thesis, either as building blocks for new proofs, or as benchmarks, to which we can compare our techniques.

What are the languages that admit a proof-labelling scheme? All distributed languages do. The technique to prove this statement is now classic, and was first evoked in [81]. Basically, there exists a *universal scheme* that works for every language. On *yes*-instances the prover provides the whole adjacency matrix of the graph, with the correspondence between the raws and the identifiers of the nodes, and also the input assignment. The nodes first check that they are given the same certificate. Then they check that this description of the graph is consistent with their local views. It is easy to show that if this step succeeds, then the configuration described in the certificates is indeed the actual configuration. Finally all the nodes check in parallel and without communication, that the configuration is in the language. This scheme uses certificates of size $\theta(n^2)$ (plus a possible term for the inputs).

Can we do better? Yes, for many languages we can have smaller proofs. For example the scheme for acyclicity sketched in Subsection 1.1.4 uses certificates of size $O(\log n)$. One of challenges of distributed decision is to show upper and lower bounds on the proof sizes for diverse languages.

What about spanning trees? The language of trees, we have studied first is not very common in distributed computing, as it deals with the communication graph itself, and not with a labelling. We have then used the example of acyclic subgraphs, like in Figure 1.4. An even more important language is the language of *spanning trees*. We want to decide whether a subset of selected edges form a spanning tree of the communication graph. The classic scheme for this language tackles acyclicity with distances, as in the scheme of Figure 1.4, but, to ensure the connectivity, every node is also given the ID of the root, called root-ID. In addition to the verification of the distances, the verifier checks that it is given the same root-ID as its neighbours. Also if a node has distances zero, then it checks that its ID is equal the root-ID. To prove the correctness of this scheme, we just have to prove the connectivity of the subgraph of selected edges, as the acyclicity is guaranteed by the distances. Consider an instance with several trees. Then, in case the prover does not give the same root-ID to every nodes, two adjacent nodes are given different root-ID, and reject. Otherwise, every node is given the same root-ID, but then, there is one node that is given distance 0, but whose ID is not the root-ID, and this node rejects. Thus the scheme is correct. Note that as the identifiers are supposed to be bounded by a polynomial in n, the size of the certificate is in $O(\log n)$), like for acyclic subgraphs.

Is $\Theta(\log n)$ optimal for spanning trees? Yes it is. This has first been proved for the original definition of proof-labelling scheme in [81], with a technique called

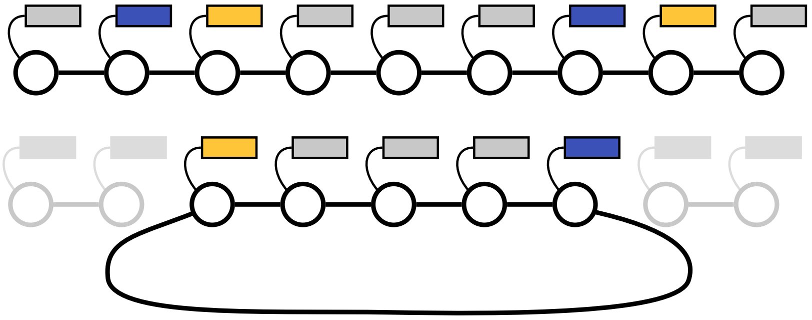
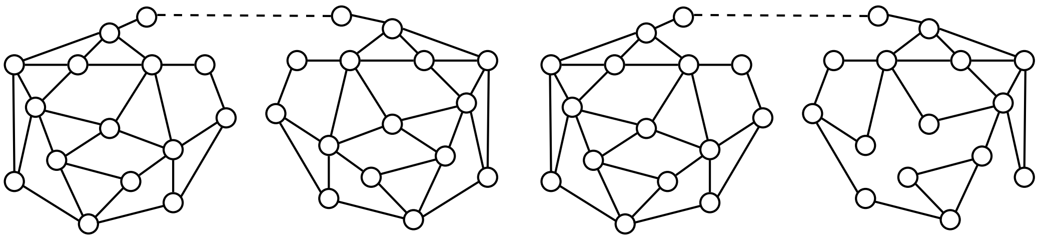


Figure 1.5: This figure illustrates the *crossing technique*, for the language of trees. It extends to spanning trees. The main point is that if $o(\log n)$ bits certificates are used, then there exist two edges, (a, b) and (c, d), such that a and c (respectively b and d) are assigned the same certificate (by the pigeon-hole principle). On the top picture, we illustrate this with blue and yellow certificates (the other certificates are all in grey, but they are not equal in general, they are arbitrary). Then we *cross* the edges: we remove, (a, b) and (c, d), add (b, c). We get a cycle, and some disconnected parts that can be discarded. The point is, if the prover provides the same certificates as in the first instance, the nodes receive exactly the same messages (remember that we consider the original definition of proof-labelling scheme). Then the nodes cannot distinguish the top and the bottom picture, although one is a *yes*-instance and the other is a *no*-instance.

edge-crossing. Then it has been proved for the more general definition in [67], with a technique called *gluing*. The crossing technique basically consists in proving that it is possible to take two edges and cross them without the nodes noticing. We give more details in Figure 1.5. This technique does not apply to our framework, but it is an opportunity to provide a simple application of the indistinguishability technique, in a non-deterministic setting.

The reason why this technique does not work for the general definition of proof-labelling schemes used in this thesis, is that if the nodes do have the knowledge of the IDs of their neighbours, then any change in the topology, such as crossing some edges, is detected. To deal with the more general framework, a more involved technique based on gluing different instances has been developed. We describe it now. For simplicity, we consider languages that are dealing with inputs, and not with the communication graph itself. We sketch the proof without any specific problem in mind, but one can think about spanning trees for example.

The high-level idea is to consider a large family of *yes*-instances, and their accepting certificates, and to show that if the certificates are not large enough, there exists a *no*-instance for which we can craft an accepting certificate assignment. More precisely, each *yes*-instance we consider is formed of two long paths, linked together by the endpoints to create a cycle. As this is a *yes*-instance, there exists a certificate assignment that makes the nodes of the cycle accept. Now consider a new instance formed by taking two paths from two different *yes*-instances and connecting them into a cycle. This may be *no*-instance. If the prover provides to the nodes of each path, the same certificates as in the original *yes*-instance it comes from, then only the nodes



(a) Symmetric dumbbell graph (b) Non-symmetric dumbbell graph

Figure 1.6: Consider a special class of graphs that we call *dumbbell graphs*. These are graphs formed by (1) taking two graphs, that we call *primary graphs*, that have the same number of nodes, and that are both are non-symmetric, and (2) linking them by a path that is at least twice as large as the verification radius. Basically, in such graphs, if a non-trivial automorphism exists, then it must exchange the two primary graphs (that is the permutation must map one to the other). This can happen only if the two primary graphs are isomorphic. On the picture above, the graph on the left has a non-trivial automorphism, whereas the graph on the right does not have one. Now the high-level idea, is that, in order to check if the two graphs are isomorphic, one needs to gather all the information about both primary graphs at one node. But there is no node that can see both primary graphs, because the path is too long. Thus the information has to be conveyed by the certificates. As the primary graphs have roughly $n/2$ nodes, describing one of them takes $\Omega(n^2)$ bits. This implies a lower bound on the size of the certificates. This intuitive reasoning about communicating information is made rigorous via a counting argument. See [67] for the details.

at the border (that is the nodes close to the endpoints of the paths) can detect the difference. Then in such instances, the main challenge for the prover is to convince the border nodes that the connection is legal. One can prove that if the certificate size is not at least $\Omega(\log n)$, then paths from several *yes*-instances, equipped with the certificates inherited from these *yes*-instances, can be connected such that: (1) the resulting graph is a *no*-instance; (2) every connection is accepted by the nodes. This results in a *no*-instance being accepted by all nodes. This is a contradiction, and the lower bound follows. See [67] for the details and illustrations.

Are there languages that require the quadratic proof size? Yes, there are. The classic examples are the ones based on isomorphism-flavoured questions. The language of *symmetric graphs* consists in the set of graphs that have a non-trivial automorphism. That is, a graph is in the language if there exists a permutation of the identifiers, different from the identity permutation, that preserves the adjacency relation. For this language, we are basically required to describe the whole graph in the certificates of all nodes. The proof of this statement is in [67]. We sketch it in Figure 1.6.

Another example of language that requires (almost) quadratic certificates is the set of graphs that cannot be properly coloured with three colours. The proof is more involved, and explicitly uses communication complexity and intricate colouring gadgets (see [67]).

Any language with $\Theta(1)$ proofs? Yes, the main examples are colourings. After quadratic size, we look at the other end of the spectrum of proof sizes. There are languages with empty proofs: the ones that are decidable with the basic decision mechanism. Just above basic decision, there are problems that have optimal proofs of size $\Theta(1)$. An example is the set of bipartite graphs: the prover just has to provide a 2-colouring, and the nodes can check that their neighbours have a colour that is different from their own. More generally deciding if a graph can be coloured with some number k of colours falls into this category.

What about minimum spanning trees? Minimum spanning trees are fundamental objects in networks and in distributed computing. We consider the minimum spanning tree language, that is the set of graphs whose edges are weighted, and some of them selected, such that the selected edges form a minimum spanning tree of the communication graph, with respect to the input weights. The optimal proof size for this language $\Theta(\log^2 n)$. This result, especially the lower bound, requires advanced proof techniques [79]. We sketch the upper bound. For simplicity, we assume that all the weights are distinct, which implies that there is a unique minimum spanning tree.

The scheme for this language is based on the GHS algorithm [63], which is itself based on Borůvka algorithm. Let us recall the principles of Borůvka algorithm, and see Figure 1.7 for the run of this algorithm on a small instance. It proceeds in rounds, adding edges to a set of selected edges, until this set is a minimum spanning tree. At the beginning no edge is selected. Then, in parallel, every node chooses the lightest edge it is adjacent to, and all the chosen edges are added to the set. These form a subgraph of the network, and the connected components of this subgraph are called *fragments*. At the next round, every fragment chooses an outgoing edge (that is an edge with exactly one endpoint in the fragment) of minimum weight. These newly chosen edges are added to the set, and consequently some fragments are merged. New rounds are played until there is only one fragment. This fragment is the minimum spanning tree. There are at most $O(\log n)$ rounds.

We now sketch the scheme based on this algorithm. The certificates describe locally the run of the algorithm. There is a field of $O(\log n)$ bits for each of the $O(\log n)$ rounds. Each field, basically contains (1) the identifier of a so-called leader of the fragment (to identify the fragment), (2) a tree of selected edges pointing to this leader (to ensure its connectivity, and the existence of the leader) (3) the weight of the lightest outgoing edge chosen for the next round, with the identifiers of its endpoints (to allow the verification of the minimality), along with (4) a spanning tree pointing to the edge (to ensure its existence). These four elements can be encoded with $O(\log n)$ bits locally. This is because by assumption the identifiers are on $O(\log n)$ bits, and the trees, as said above can be verified with $O(\log n)$ bits too. As there are $O(\log n)$ rounds, these local proofs have size $O(\log^2 n)$.

Is there any natural problem with proof size between $\Theta(\text{polylog}n)$ and $\Theta(n^2)$? Until recently, only artificial languages were known with proof size within this range. In particular, the known techniques were not able to prove such intermediate lower bounds. This is not the case any more, with the introduction in distributed decision

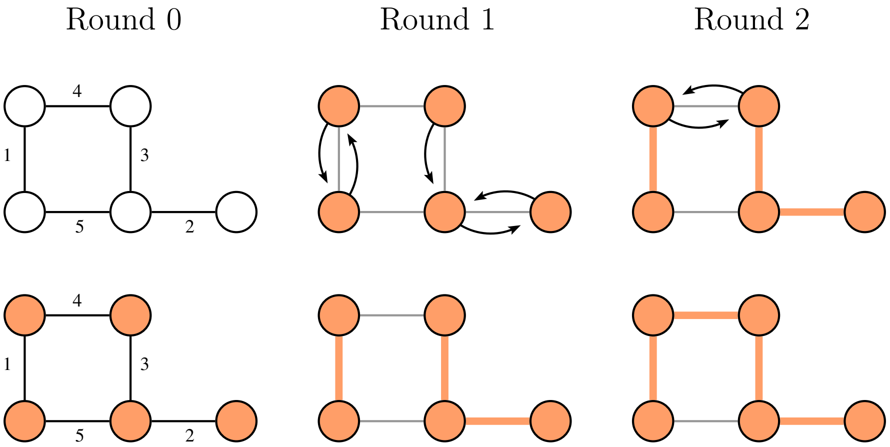


Figure 1.7: Illustration of Borůvka algorithm. In the first column, the top picture is the original instance with the weights, and the bottom picture illustrates the original fragments, that is, the nodes. In the second column, we depicted the outgoing edges chosen by the nodes, and the fragments at the end of Round 1. In the third column, we drew the edges chosen by the two fragments, and the final output. This output is the minimum spanning tree.

of tools from communication complexity, with gadgets developed for lower bounds for construction tasks. Specifically, verifying that the diameter of the communication graph is bounded by some integer k is proved to be in $\Omega(n/k)$ and $O(n \log n)$ in [24].

Do a language and its complement have the same proof size? As a language is a set (of labelled graphs), one can ask whether deciding the complement of this set is as hard as deciding the original language. In terms of proof-labelling scheme, this means establishing whether they have the same optimal proof size. In the centralized setting, for deterministic computation, complement languages have the same complexity. However for non-deterministic (centralized) computation it is (a priori) not the same.[3] In distributed decision, the deterministic setting is already non trivial. For example, consider the language where the nodes of the graph are given colours, and we want to decide whether this colouring is a proper colouring. This can be done with a basic decision scheme. But what about accepting only *bad* colourings? We cannot just make every node flip its decision. Indeed, consider a colouring with a single monochromatic edge. In the scheme for proper colourings, every node accepts, except the ones close to the monochromatic edge. After flipping, every node rejects except the two endpoints of the monochromatic edge, which means rejecting the instance. The question is then: how to *reverse decision*? In general, this comes at a cost, for example for bad colourings, one can reverse decision with $O(\log n)$ proofs (by pointing to the monochromatic edge with a spanning tree).

This difference between a language and its complement also appears when none of them is locally decidable. For example, the language of the graphs that can be 3-

[3] See for example the NP vs co-NP question.

coloured has constant size proofs, but that its complement requires almost quadratic proofs.

1.3 Overview of the thesis

The thesis can be summarized as a study of different alternative definitions of proof-labelling schemes. The goal of this approach is twofold. First, these new models are justified by diverse practical scenarios, that are better captured by a different form of non-determinism. Second, these models enable us to study diverse aspects of distributed non-determinism: error-sensitivity, uniformity, redundancy and the impact of interactivity. The core of the thesis consists in four chapters one for each of these concepts. These are adaptations of four papers, either published or accepted for publication. The only significant change in the structure is Section 6.5, that originally appeared along with the content of Chapter 4. For each of these four chapters, we now give a high-level description of the model, along with sketches of the main results and techniques. Formulas and technicalities are avoided to keep a narrative style ; readers more comfortable with more formal statements can jump right to the technical content.

1.3.1 Error-sensitivity *via* a stronger rejection (Chapter 3)

Two questions. This work originates from two seemingly unrelated questions. The first one is related to *property testing.* Property testing (of graphs) is a field of centralized computing that, very roughly speaking, studies whether one can test if a graph has a property or not, by just querying the adjacency of a few nodes. That is, in this scenario, the (centralized) algorithm does not have access to the whole graph, but is probing a database to get enough information about it. A task can be solved in such a restricted model only if we relax the usual algorithmic problems. The definition of graph property testing allows to make a mistake in the decision if the graph is *close to have the property* (for some notion of *close*). We note that property testing has a flavour of both locality and decision, thus it is natural to see if we can relate it with distributed decision.[4] The second seminal question of this chapter is: can we make more than one node reject a bad instance? Indeed, the definition of proof-labelling scheme only requires that for every certificate assignment at least one node rejects, but in practice, the more nodes rejecting, the faster we can fix the situation. In a nutshell we ask what happens to the results on proof-labelling schemes if we require a stronger rejection?

Error-sensitivity. We define a notion of *error-sensitivity* for proof-labelling schemes. Basically, a scheme is error-sensitive, if the number of rejecting nodes is at least linear in the distance from the instance to the language. We will precise this definition soon, but let us give a motivating example first. We use our running example of verifying acyclicity. Every node is given as input an adjacency list, and we want to check whether these lists define a proper acyclic subgraph or not. Consider three instances

[4] Relating property testing to distributed graph algorithm has led us to define the model of Chapter 3, but it has also motivated another field of research, somehow closer to the original property testing, that focuses on congestion and uses randomization. See the seminal papers [21, 23], the papers citing them, and the paragraph on the topic in the survey chapter in Subsection 7.3.2.

(see Figure 1.8): in the first one the subgraph is a proper forest, in the second it has only one cycle, and in the third it is a very dense subgraph.

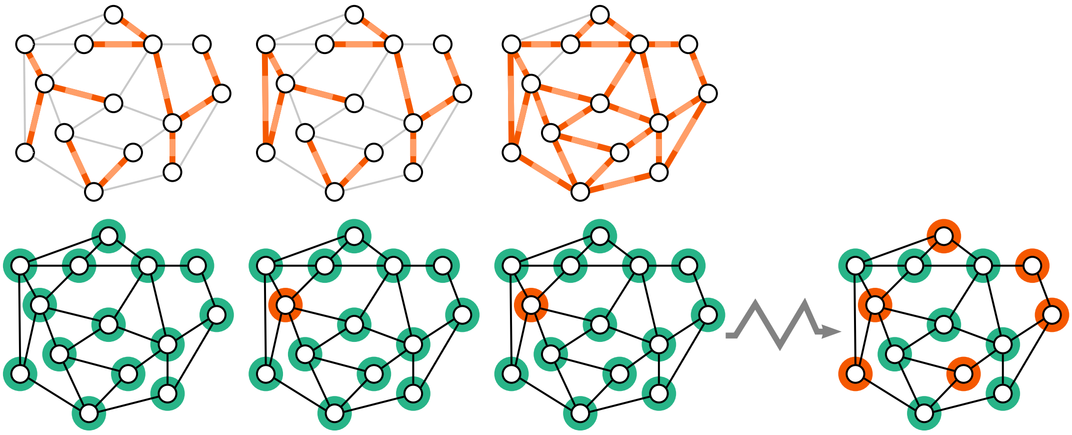


Figure 1.8: The three instances of the paragraph *Error-sensitivity*, along with the decisions of the nodes. First column: the top picture represents a *yes*-instance, thus all the nodes have to accept if given the right certificates (bottom picture). Second column, a *no*-instance, close to the language, and the nodes decisions on the bottom: at least one node must reject, whatever the certificate assignment is. The third instance is similar to the second, but it is far from the language, thus we can hope to have more than one node rejecting, as pictured on the graph on the far right.

In a proof-labelling scheme, the verifier is required to accept the first instance if provided the right certificates, and to reject on at least one node, for every certificate assignment on the two other instances. In the second instance, a very large portion of the subgraph is acyclic, that is by changing just a little bit the inputs we can get a *yes*-instance. Then we cannot hope that there exists a scheme where many nodes are rejecting, because of the following prover strategy. The prover first decides which is the accepting certificate assignment of the close *yes*-instance, and then uses the same assignment on the current instance. Because few nodes can detect the change in the input (because we took a close *yes*-instance), then few nodes can reject. However, in the third instance there is no very large part that could appear in a proper instance, thus we can hope that with any certificate assignment many node will catch inconsistencies.

A distance on instances. To be more precise we need to define what it means for an instance to be *far* or *close* to a language. Let us first precise the type of languages we consider. In this chapter, we only consider languages that deal with *node inputs*. That is we will not consider properties of the communication graphs, nor properties of edge labellings. This is actually the type of language that makes more sense in practice. Indeed it corresponds to the scenario where a construction algorithm builds some solution to a problem (with every node holding its part of the output), and where we want to check that this solution is correct. Also in this context every graph has a solution (*e.g.* every weighted graph has a minimum spanning tree), thus we require that for every graph, there exists a node labelling such that the labelled graph belongs to the language.

Now the distance is actually a distance between node labellings. The distance between two input labellings of the same graph is the number of inputs that differ. The distance from a configuration to a language, is then the minimum number of nodes we have to edit to transform the labelling into a labelling of the language, for this graph. In the three instances of the previous paragraph and of Figure 1.8, the first instance is at distance zero, the second is at distance two (because editing the two endpoints of an edge of the cycle is necessary and sufficient to get acylicity), and the third instance is at a distance that is linear in the size of the graph, because to remove all the cycles, one has to edit the adjacency lists of almost all the nodes.

The precise requirement for a scheme to be error-sensitive, is that on *no*-instances, the number of rejecting nodes is lower bounded by the distance of the instance to the language, up to a multiplicative constant, for every possible certificate assignment.

Not every language is error-sensitive. Playing with the definition, one quickly notice that not every proof-labelling is error-sensitive. Then one can show that there are actually languages that do not admit any error-sensitive proof-labelling scheme. This stands in contrast with the situation without error-sensitivity constraint, for which the universal scheme (as described in the first paragraph of Section 1.2) is a proper scheme for every language.

We sketch a situation for which even the universal proof-labelling scheme, that provides the most information to the nodes is not error-sensitive. Consider a communication graph made by joining two labelled graphs A and B by an edge, that we denote $A - B$. The prover can always make sure that almost all the nodes of A believe they are in some labelled graph $A - C$, and that almost all the nodes in B believe they are in some labelled graph $D - B$. Only the nodes close to the bottleneck will notice the difference. The scheme can be error-sensitive only if the language is such that, if $A - C$ and $D - B$ are in the language then $A - B$ is close to the language. For the problem of deciding if a selected subgraph is symmetric (the subgraph version of the language we considered in Section 1.2), if the subgraphs of A and C (respectively D and B) are isomorphic, but the one of A is very far from the one of B, then this condition is not satisfied. This means that there is no hope to get error-sensitivity for this language.

The first main result of the chapter is a characterization of the languages that admit an error-sensitive proof-labelling scheme (Theorem 3.1). We call such languages, *error-sensitive languages*.

Characterization by local stability. To state this characterization we need to define the concept of *local stability*. The definition is based on a hybridization operation on graphs, pictured on Figure 1.9, that we detail now. Let $G = (V, E)$ be a graph, and let $(H_i)_i$ be a family of subgraphs of G, such that the vertex sets of the $(H_i)_i$ form a partition of V (top left picture in Figure 1.9). For every i, consider a graph G_i, that has H_i as subgraph, and has a correct labelling with respect to the

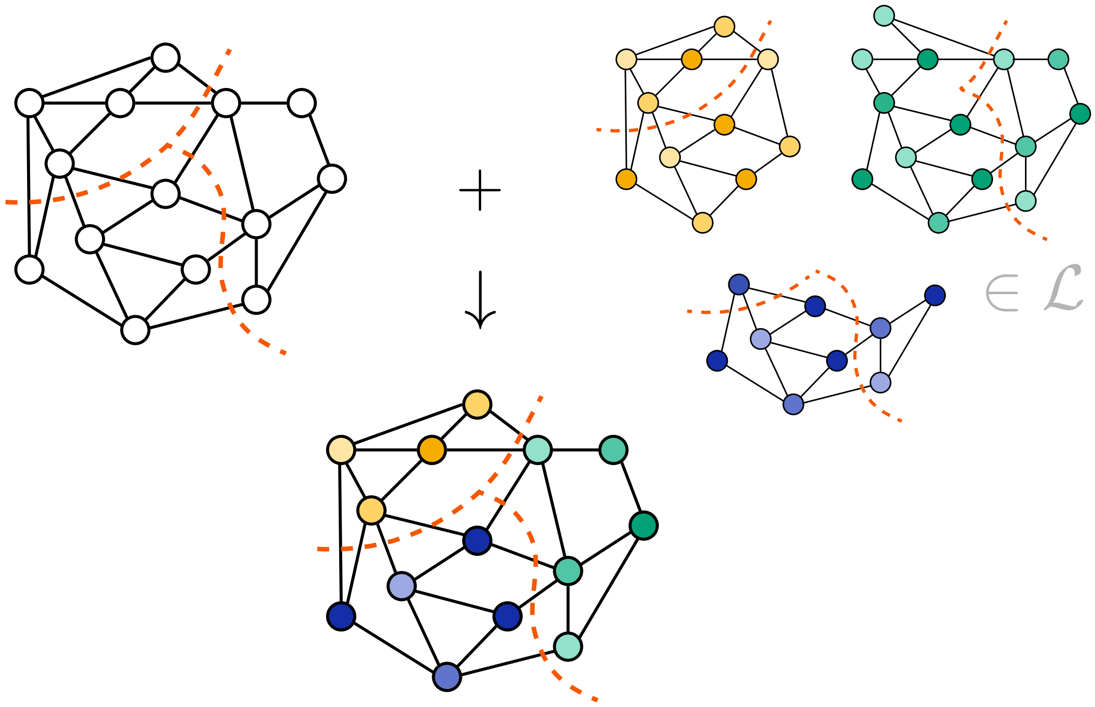


Figure 1.9: The hybridization operation used in the definition of local stability. The colours are meant to identify from which G_i a labelling comes from in the bottom picture.

language (top right picture in Figure 1.9). Finally consider the graph G with the hybrid labelling inherited from the $(G_i)_i$ on the $(H_i)_i$ (bottom picture in Figure 1.9).

Note that in general such an hybrid is not in the language. We say that a language is locally stable, if for every such hybrid, the distance to the language, is upper bounded by the number of *border nodes* up to multiplicative constants. A node is a border node, if it belongs to some H_i and has a neighbour in a subgraph H_j (with $i \neq j$). To help get a grip on this definition, see the example of acyclic subgraph in Figure 1.10. The characterization of error-sensitive languages is now expected:

Theorem. *A language admits an error-sensitive proof-labelling if and only if it is locally stable.*

Note that the concept of local stability is purely graph-theoretical, which contrasts with proof-labelling schemes that are much more algorithmic. Local stability is helpful because it provides a (more) mechanical way to prove or disprove error-sensitivity, as it deals with graph structures and not with the existence of a prover strategy and a local verifier. With local stability, we can show not only that the language of acyclic spanning subgraphs is error-sensitive (as sketched in Figure 1.10), but also that the languages of spanning trees and minimum spanning trees are error-sensitive too.

The proof of this characterization (that we will not discuss here), builds on the fact that if there exists an error-sensitive proof-labelling scheme, then the universal proof-labelling scheme is also error-sensitive. The drawback of this proof strategy is

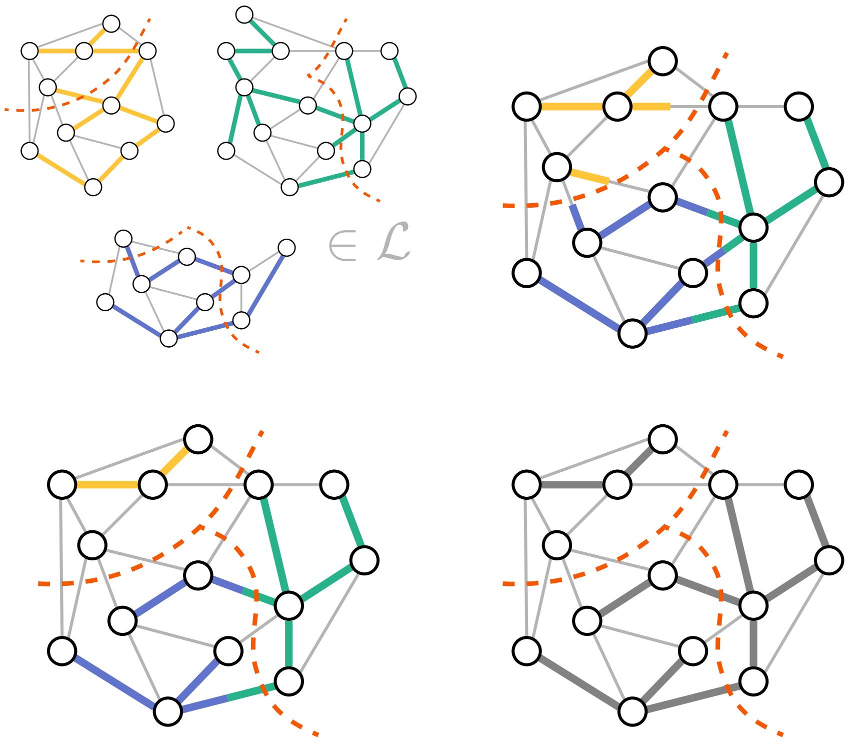


Figure 1.10: We sketch a proof of local stability for the acyclic subgraph language. Consider three instances of the language and a partition, as in Figure 1.9, but for the language at hand (top left picture, with three little graphs). As the selected edges are encoded via an adjacency list at every node, doing the hybridization (top right picture) creates half edges: edges that are in the adjacency list of exactly one of the endpoints. Also it can create cycles, as the green-blue cycle on the bottom part. All these cycles must cross the border (the orange dashed line) at least twice, by construction. There is an easy way to transform this hybrid into a proper acyclic subgraph: remove all the half-edges, and, remove the cycle edges crossing the boundary, one by one, until there is no cycle (bottom pictures). The distance between the original hybrid and the final instance is bounded by the number of border nodes, because we have only edited border nodes.

that if a language is proved to be locally stable, then we are only guaranteed that the universal scheme is error-sensitive. This means that the only upper bound on the optimal proof size we have, when restricting to error-sensitivity, is the trivial one: $O(n^2)$. This leads to the second main contribution.

Main result 2: Optimal error-sensitive schemes for spanning tree problems. Because of their local stability, spanning trees and minimum spanning trees are error-sensitive, and we know from Section 1.2, that the optimal proof size for these languages are respectively $\Theta(\log n)$ and $\Theta(\log^2 n)$. A natural question is whether it is necessary to allow larger proof to get error-sensitivity. We answer by the negative.

Theorem. *The classic proof-labelling schemes for spanning trees and minimum spanning trees are error-sensitive.*

The proofs are quite technical and cannot be described here. However we can give a glimpse of the difficulty by explaining a surprising phenomenon that appears for these languages (and does not occur in the simple example of Figure 1.10). It can be the case that, to transform a *no*-instance into a *yes*-instance, it is necessary to edit nodes that (for a given certificate assignment) are accepting, and that are actually very far from a rejecting a node. That is, in a *no*-instance, there is always a node to catch an inconsistency, but this node can be very far from the place where we can actually fix the problem. The typical case for spanning trees is an instance with two acyclic subgraphs that together span the graph (which is a *no*-instance, because it is not connected). In this case the prover can have the following strategy. First choose a root in each tree, far away from the edges of the communication graphs that could connect the two trees if selected. Second, give the correct distances to the nodes. Third, in the first tree, give the name ID_v of the root-node v of this tree. Fourth, in the second tree use again ID_v (which is then not correct, as v cannot be the root of the two trees). Then the only node to detect the problem will be the root node of the second tree, but this node is very far from a place where one could reconnect the two trees.

1.3.2 Uniformity *via* the price of locality (Chapter 4)

Uniformity of proofs. Looking at the classic proof-labelling scheme from the toolbox of Section 1.2, one can notice that there is often a part of the certificates that is common to every node. For example, in the universal scheme, all the nodes have the same description of the graph. In other schemes, it is common that the name of a particular node is known to every vertex. The seminal question of this chapter is: what if we require that the certificates are completely *uniform*, that is, that they are exactly the same for every node? Or in other word what if, instead of giving each node its own *local proof*, the prover writes a single *global proof* on some register that every node can access?

This scenario matches some practical settings. For example, consider the case where a monitor is observing the whole network, and broadcasts a piece of global information to the nodes to help them decide the state of the network. As this monitor may be corrupted, the nodes have to check this update locally, which corresponds to our model. A more theoretical motivation for this chapter, and also for Chapter 3, is to study how much do the distributed proofs overlap. Here, we study the simplest relation there can be between proofs of different nodes: equality.

Price of locality. In order to understand uniformity in proof labelling schemes, we study the relations between three kinds of proofs: the (purely) local proofs (the usual ones in proof-labelling schemes), the (purely) global proofs, where the nodes only have access to the same global proof, and the mixed proofs, that allow both local and global proofs. The first main question we ask is: can mixed proofs be much more efficient than local proofs? Or from the other point of view, is there a large price to pay when going from mixed proofs to local proofs? To answer this first question, we define the *price of locality* of a language, as the ratio between the sum of the optimal local proofs and the size of an optimal mixed proof, which is the sum of the local proofs and the global proof. See Figure 1.11 for an illustration of the notions of this paragraph. We can define the *uniformity* of a language as the inverse of the price of locality: if an (optimal) scheme is uniform, then this parameter is 1 (or constant), whereas if the price is high, it means that the certificates given to the nodes in an optimal scheme are far from being uniform, and then the uniformity is goes to zero. The price of locality is somehow more handy, thus we focus on this one, although using uniformity would just be a change of vocabulary.

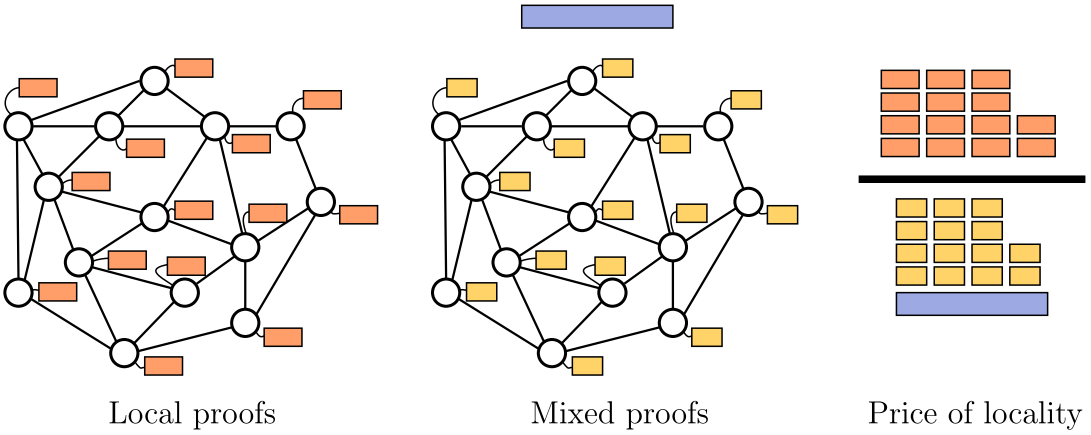


Figure 1.11: On the left, the usual scenario for proof-labelling scheme: local (orange) proofs. In the middle the scenario with mixed proofs: local (yellow) proofs, and a global (blue) proof (that every node can see). On the right, the price of locality ratio: the sum of the (optimal) local proofs, divided by the size of the optimal mixed proofs (that is the sum of local proofs and of the global proof).

Main result: the price of locality can reach the extreme values. It is easy to see that the price of locality is between 1 and n. The n bound comes from the fact that one can craft local proofs from a global proof, by simply copying the global proofs on the n local certificates. The other bound follows from the fact that mixed proofs generalize local proofs. We prove that these extreme values can be reached.

Consider the two following dual problems. For both problems, every nodes has an input stating whether it is selected or not. Then we consider the language where *at most* one node is selected, Amos for short, and *at least* one node is selected, Alos for short. Note that these simple problems are fundamental as their conjunction form the classic problem of checking *leader election*.

Theorem. *The price of locality is $\Theta(n)$ for* Amos*, and $\Theta(1)$ for* Alos*.*

The proof for Amos is simple. One can show that the minimum size of local proofs is obtained when, on *yes*-instances, the prover provides the name of the selected node (or an empty label if no one is selected) to all the nodes. And an obvious mixed proof consists doing exactly the same, but on the global certificate (this is actually a purely global proof). The ratio between the sum of the local proof and the mixed proof is the number of nodes, that is n.

The case of Alos is more complex. We prove that a mixed proof requires at least $\Omega(n \log n)$ bits. This means that we gain nothing by using a mixed proof, because a simple local proof scheme certifies the language with $O(\log n)$ bits certificates. In other words, in this case distributing the proof is for free, and the price is in $\Theta(1)$. The proof of this result is a kind of dual of the technique of Göös and Suomela [67] sketched in Section 1.2. On a very high level, the proof of [67] consists in gluing portions of *yes*-instances to get a *no*-instance that cannot be distinguished from a *yes*-instance, whereas our proof consists in showing that if there are many *yes*-instances, then we can shorten one of them, and create a *no*-instance that cannot be distinguished from a *yes*-instance. In [67] it is proved that a bunch of languages require $\Omega(\log n)$ local proofs, and we can lift these results to $\Omega(n \log n)$ mixed proofs. In particular, we can show that spanning trees require $\Omega(n \log n)$ mixed proofs.

Intermediate cost. We can prove that there are languages with price of locality in $\omega(1)$ and $o(n)$. More precisely, we prove that minimum spanning trees have price of locality $\Theta(\log n)$. For this, we design an optimal mixed proof of size $\Theta(n \log n)$, which is to be compared with the optimal $O(\log^2 n)$ local proofs described in Section 1.2.

Purely global proof. The second question of this chapter, is whether, for a given language, the optimal global proof can be strictly larger than the optimal mixed proof. At first, it seems that the answer is negative. Indeed one thinks that the local proofs of a mixed scheme can be transformed into a global proof, by concatenating them. But this is not enough because, a priori, a node has no way to find out which part of this new global proof is supposed to be its own certificate. A correct global proof is the concatenation of all the local certificates, along with the identifiers of the corresponding nodes. However, for some languages this global proof is larger than the mixed proof. For bipartiteness for example, the local proofs are on $\Theta(1)$ bits, and the strategy above gives a global proof on $\Theta(n \log n)$ bits, which is larger than the $O(n)$ mixed proof. We actually show that for bipartiteness, if we change the model to allow arbitrarily large IDs, then the global proof can be arbitrarily large, and in particular strictly larger than mixed proofs.

1.3.3 Redundancy *via* a larger radius (Chapter 5)

Redundancy in the tree scheme. In the previous section, the main theorem about ALOS implies that in some sense a scheme for acyclicity has no global correlation, because making it global basically requires to copy all the local proofs. However, when looking at the scheme with distances, it does not seem to be true that the proofs are not correlated: the distances of two adjacent nodes are very related, indeed they only differ by one unit. The concept of uniformity is not fine-grained enough to discuss this aspect. The reason is basically that the relation between the distances fade away when we consider nodes further apart, thus it is not captured by a global point of view. We need a more local tool. The variant of proof-labelling schemes that fits this purpose best is to consider that the verifier can look at non-constant distance.

Proof-labelling schemes with non-constant distance. We consider the modification of proof-labelling schemes where the radius of the nodes' views can grow with the size of the network n. For example, we consider schemes where the nodes can look at distance $\Theta(\log n)$. The rationale is that by looking further, the nodes get to know more certificates given by the prover ; then if some piece of information present in the certificate of a node is very related with a piece present in a neighbours' certificate, we can probably avoid to repeat it, and it should reduce the optimal proof size. In other words, we want to study redundancy by measuring how fast the optimal certificate size decreases when we increase the radius.[5] Seminal work on this concept was done in [90].

Three scenarios. For the rest of this section, t will denote the radius, and k the number of bits used in the proof-labelling scheme at distance 1, for the language at hand. For the size of the certificates for a scheme at distance t, three natural scenarios come to mind.

- There is no trade-off: whatever the radius is (strictly below the diameter), we need proofs of the size $\Theta(k)$.
- There is a *linear scaling*: there is a scheme that uses $O(k/t)$ bits certificates.
- There is a *scaling by the size of the ball* around the node: if the nodes have at least $b(t)$ nodes in their neighbourhood at distance t, then the certificates can be as small as $O(k/b(t))$.

A language for which we are in the first, second and third scenario respectively can be called *non-redundant*, *redundant* and *highly redundant*. And we can define the the redundancy of a language as the denominator of the proof size.

[5] A subtlety is that when we increase the radius, not only the verifier has access to more certificates, thus more pieces of the information from the prover, but also it has a better knowledge of the structure of the graph in its neighbourhood.

The main questions of this chapter are: (1) is one of these scenarios is the right one for all the languages? and (2) if not, in which category are the classic languages?

General results. We give two partial answers to the first question. The first answer is that, when the local proofs are uniform, we achieve scaling by the size of the ball.

Theorem. *Uniform (radius-1) proof-labelling schemes can be turned into (non-uniform) radius-t schemes with certificates of size $O(k/b(t))$.*

This is somehow expected: in this case the proof is very redundant, thus there must be good scaling (or otherwise the scaling would not be a proper measure of redundancy). In particular in graphs that are quite dense, the universal scheme can give very small proofs if turned into a scheme at distance t. Note that even if the scheme is not uniform, but has only a uniform part, typically an ID shared by all the nodes, then at least this part of the proof can be made much smaller if we allow larger radius. The proof is sketched in Figure 1.12.

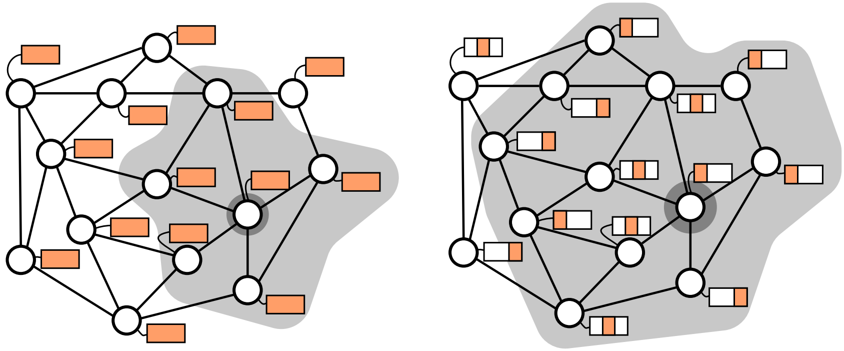


Figure 1.12: Illustration of the scaling of uniform proofs. On the left the classic framework for a proof-labelling scheme at distance 1. On the right, the illustration of the radius t scheme. In this scheme the prover strategy is the following. First decide what is the uniform certificate assignment for distance 1. Then split the uniform label into $\Theta(b(t))$ equal parts. Finally assign to every node one of these parts, in such a way that every node has at least one copy of every part in its neighbourhood. The existence of such a certificate distribution is proved by a probabilistic method.

For the second result we do not restrict the type of proofs, but the topology of the communication graph. In restricted topologies, such as cycles and trees, we get linear scaling for all languages.

Theorem. *On cycles, grids and trees, every language has proofs of size $O(k/t)$.*

The proof of this result has several parts; let us just state and sketch the proof of the key lemma. The proof is based on the fact that if the prover takes the certificates of the proof-labelling scheme at distance 1, but assigns them only to a $1/t$-fraction of the nodes, then there is still a way for the verifier to decide. The nodes that will be given a certificate have to be carefully chosen. On a path for example: if we number the positions of the nodes of the path from 1 to n, then the prover selects the node v_i, if and only if i is equal to 0 or 1 modulo t. This guarantees that we have subpaths of non-selected nodes of length roughly t, separated by blocks of two nodes. The verifier algorithm, if not given a certificate, accepts. If the node is given a certificate, it tries all possible certificate assignments of the nodes of the adjacent subpaths. If it finds an assignment that makes the verifier of the radius-1 proof-labelling scheme accept, then it accepts, otherwise it rejects. A key point is that if the nodes that are given certificates are properly chosen, then the success of these independent simulations ensures that there exists an accepting certificate assignment for the whole graph.

Once this simulation argument is proved, we have a proof-labelling scheme at distance t with proofs of size $O(k/t)$ *on average*. We then design a way to spread the certificates, a bit like in the previous theorem but deterministically, to ensure that every certificate has size in $O(k/t)$.

Specific languages. In addition to these general results, we show diverse scalings for spanning trees, minimum spanning trees, and diameter, which require other techniques, but that we will not detail here.

1.3.4 Interactivity *via* alternation (Chapter 6)

Changing the way of measuring performance. So far the performance of a scheme has been measured by the size of its certificates. As we have seen in Section 1.2, some languages require very large certificates. A reasonable approach is to fix a maximum size for the certificates, and to relax other constraints, or change the model. The approach we take for this chapter is to consider *interactivity* in the proof process. More specifically, we will allow interaction between *two* provers. The nodes are going to receive not only one message from one prover, but the transcript of a conversation between the two provers.[6] Once again this is inspired by the complexity theory of centralized computing.

Polynomial hierarchy in centralized computing. In centralized computing, the basic class is P, and adding non-determinism defines NP. In NP, a prover provides a proof to the machine, and the machine verifies it in polynomial time. In some sense the prover is trying to prove that the instance is in the language. What if we take the definition of NP but allow two provers to provide proofs? More precisely, what if we consider that a prover and a disprover (that is an oracle who tries to prove that the instance is *not* in the language) can discuss, and that the machine has the transcript of the discussion? This is what the polynomial hierarchy is about. The hierarchy is made of an infinity of level. Each level corresponds to a type of prover-disprover interaction, and contains all the languages that can be decided with such a protocol. For example the level called Π_2 is the set of languages that can be decided by scheme where the disprover provides a first certificate and the prover provides a

[6] Note that we do not consider interactivity in the sense of exchange of messages between the prover and the network, but only between provers.

second certificate to the machine. The level Σ_2 is the same except that the prover speaks first. Note that NP is included in both Π_2 and Σ_2 because in both cases if the machine considers only the certificate given by the prover, we are back to the definition of NP. The differences between the levels of the hierarchy is the identity of the first to speak, but more importantly the number of times there is a change of speaker. Such change is called an *alternation.* By increasing the number of alternations between prover and disprover, we get larger and larger classes.

Distributed hierarchy. We adapt the concept of the polynomial hierarchy to distributed decision, and define a *local hierarchy.* The notion is exactly the same as in the centralized setting, except that the prover and the disprover provide local proofs to a local verifier. Depending on the level of the protocol, there is some number of alternations between the two oracles, which implies that every node is given a bunch of certificates. For example, in a scheme of the type prover-disprover-prover, when it takes its local decision every node is holding three certificates: the first certificate comes from the prover, then the disprover can craft a second certificate assignment to answer, and finally the prover finishes the discussion by giving a third certificate.

$O(\log n)$ threshold on reversing decision. As said in the introduction of this part, we want to bound the size of the certificates, and use another measure efficiency. This other measure will be the number of alternations. The question now is about the good threshold for the certificate size. The size $O(\log n)$ was identified in [67] as an important threshold as it is the minimum size to certify trees and to encode IDs and polynomial numbers. We then choose $O(\log n)$ as the size of the certificates used by the prover and disprover.

Example: symmetric graphs. To make the notion of alternation more concrete, consider the language of symmetric graphs (defined in Section 1.2). We can show that this language has a protocol of the form prover-disprover-prover. First the prover will (supposedly) give to every node the name of its image by the non-trivial automorphism. Then the disprover will (supposedly) point to an error in this automorphism (for example it may prove that two nodes have the same image, by basically having a spanning tree pointing to each of them, along with the shared image ID). Finally the prover will (supposedly) certify that the disprover is cheating (for example by pointing to a place where the distances in the tree of the disprover are inconsistent).

Structural results. For this introduction we focus on the structure of the hierarchy, but the chapter also contains alternating schemes for different languages from combinatorial optimization, logic etc. We first show that there is a collapse in the hierarchy that does not exists in centralized computing.

Theorem. *The class corresponding to a protocol where the disprover speaks last is equal to the class of the same protocol without the last disprover message.*

The basic reason for this is that each node can locally simulate the last certificates of the disprover. That is, the nodes can test locally whether there exists a certificate assignment of the nodes of its view that would make the verifier reject; and if node detects this, it rejects. Consequently, given a certain number of alternations, there exists exactly one type of scheme. The level zero corresponds to basic decision, level one corresponds to proof-labelling schemes, level two corresponds to disprover-prover protocols, level three to prover-disprover-prover etc. In addition, it is easy to show that using a tree construction the following holds.

Theorem. *The complement of the class of level i is always contained in the class of level $i+1$.*

Big open question and relation with communication complexity. The main open question of this chapter is whether the second level (disprover-prover) and third level (prover-disprover-prover) are separated. If they are not, then the whole hierarchy collapses, and there are just three levels (this is another structural result). A priori, a good strategy to prove such separation, is to use the communication-complexity-like framework of [67] (used to show that some tasks require $\Theta(n^2)$ bits, see Section 1.2). We could not make this technique work with alternation, but, as we discovered later, there is a good reason for this. In communication complexity, a similar hierarchy has been defined, and the question of separating the different levels has been open for more than 30 years. We are able to prove tight links between the proof strategy of [67] and the communication complexity framework (in particular if we consider a variant of the hierarchy with global proofs as in Chapter 4).

1.4 Annotated list of publications

This section is an annotated list of my works in distributed computing. It does not contain some earlier works on approximation algorithms.

- Laurent Feuilloley and Pierre Fraigniaud. Randomized local network computing. In *27th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 340–349, 2015. doi:10.1145/2755573.2755596.

  This paper is the main work of my masters thesis. It is the derandomization result we mentioned in Subsection 1.1.6. It generalizes a result that was known for languages that can be checked with a basic deterministic scheme to the ones that can be checked with a probabilistic scheme.

- Laurent Feuilloley, Juho Hirvonen, and Jukka Suomela. Locally optimal load balancing. In *29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 544–558, 2015. doi: 10.1007/978-3-662-48653-5

  This work is about distributed (construction) algorithms, for a load balancing problem in bounded degree graphs.

- Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 118:1–118:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.118

  This paper corresponds to Chapter 6, and to Subsection 1.3.4 in the introduction.

- Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119, 2016. url: bulletin.eatcs.org link, arXiv: 1606.04434

  This article corresponds to the survey at the end of this thesis, Chapter 7.

- Laurent Feuilloley. How long it takes for an ordinary node with an ordinary ID to output? In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, pages 263–282, 2017. doi: 10.1007/978-3-319-72050-0

  This work is about construction and not decision, thus for the sake of consistency it does not appear in the thesis, although it was done during my PhD. It basically asks the question: what if we allow different nodes to have views with different radiuses, and measure the performance by the average of these radiuses, instead of the maximum?

- Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 16:1–16:15, 2017. doi:LIPIcs.DISC.2017.16

  This corresponds to Chapter 3 and Subsection 1.3.1 in the introduction.

- Laurent Feuilloley and Juho Hirvonen. Local verification of global proofs. In *DISC 2018 (To appear)*, 2018. arXiv: 1803.09553

  This paper corresponds to Chapter 4, and Subsection 1.3.2 in the introduction.

- Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. In *DISC 2018 (To appear)*, 2018. arXiv: 1803.03031

  This paper corresponds to Chapter 5, and Subsection 1.3.3 in the introduction.

- Laurent Feuilloley and Michel Habib. Graph classes and forbidden patterns on three vertices, 2018. Manuscript

  This work is still in progress, and is mainly about classic graph theory thus does not appear in the thesis. But as it has a nice link with proof-labelling schemes, we take a few paragraphs to describe it.

  This work is at the interface between two research lines in graph theory. On the one hand, there is a large body of research focused on studying the graph classes where some subgraph or some minor is forbidden. These graph classes have very interesting structures and are described by deep theorems, such as the celebrated Robertson-Seymour theorem. On the other hand, many algorithms recognizing useful graph classes are not detecting forbidden subgraphs but forbidden ordered structures in some ordering of the nodes of the graph. A typical form for a fast recognition algorithm is to first perform one or several tree traversals, and then to check that some structure does not appear in the ordering of the nodes produced by these traversals. For example one wants to reject if there are three nodes a, b and c in this order in the traversal such that (a, c) is an edge but (a, b) is not an edge.

  The goal of our work is to understand the classes that can be defined the following way: the graph is in the class, if and only if, there exists an ordering of the nodes such that some pattern does not appear. Such characterizations already exist for some well-known classes such as interval graphs, comparability graphs, split graphs etc. We differ from previous work by taking a systematic approach to the question.

  The link with proof-labelling schemes may appear to the reader after this long introduction: if the ordering can be given by the prover, checked locally, and if the patterns can also be checked locally, then the class can be recognized by a proof-labelling scheme with $O(\log n)$ proofs. We show that this is the case for several well-known classes: trees (with a fairly different scheme than the one we have described before), interval graphs, chordal graphs, path and bipartite graphs (although in this last case the $O(\log n)$ proofs are not optimal).

Chapter 2

Model and definitions

In this chapter, we describe the model we study, and define the main objects used in the thesis. The definitions that are specific to a chapter are deferred to the beginning of the chapter in question. The survey of Chapter 7 has its own notations that are more complex, because they describe more variants of the objects.

Graphs. Throughout the thesis, the communication network is modelled as a graph, the *communication graph*. All graphs are undirected, connected, and simple (no self-loops, and no parallel edges). The set of vertices is denoted by V, and the set of edges by E. The size of the network is the size of V, and is denoted by n. Given a node v of a graph G, we denote by $N(v)$ the open neighbourhood of v, i.e., the set of nodes that are adjacent to v in G. Depending on the context, the word *neighbourhood* may also refer to the nodes that are at distance at most d from v, where d is a constant, or to the subgraph induced by these nodes.

A graph is *(input) labelled*, if there are labels associated with the nodes, or, more rarely, with the edges. Note that an edge labelling can always be turned into a vertex labelling: the endpoints of the edge can hold the label of the edge. An input labelling can be formalized with a function $x : V \rightarrow \{0,1\}^*$, and an edge labelling as a function $x' : E \rightarrow \{0,1\}^*$. An example of edge labelling is edge weights, when considering minimum spanning trees. The input labels (or simply *inputs*) often have constant size, but in some cases (such as edge weights) they are only bounded by a polynomial in n. A labelled graph is called a *configuration.*

Identifiers. In every graph we will consider, each node v has a name, called *identifier* or *identity*, denoted by $\mathrm{ID}(v)$, taken from the set $\{1, \dots, N\}$, where N is polynomial in n. In other words, all identities are stored on $O(\log n)$ bits.[1] In a same graph, all names are pairwise distinct.

LOCAL model as local snapshots. We use a quite high-level model of communication. In the diverse scenarios we consider, a node always gets a snapshot of its

[1] In Section 4.4 we will not bound the identifiers by a polynomial, but consider that the upper bound N is a parameter of the problem, that can appear in the complexity.

neighbourhood at some distance in the graph. This is equivalent to use the classic LOCAL model [87, 95], where nodes exchange messages in rounds, with no constraint of message size, nor of computational power or space. This model is suited to our approach, which is focused on locality, and does not consider message complexity. The neighbourhood that a node has access to, is called its *view*. It is a ball centred at the node running the algorithm. The algorithms dealing with such local views are often called *local algorithms*.

Distributed languages. A *distributed language* is a set of configurations, that is a collection of labelled graphs. Most of the time, we consider languages where only the nodes are labelled, and not on the edges, thus a language is a set of configurations (G, x), with x a node labelling function.[2] Note that the identifiers do not appear in this definition. A language is often denoted $\mathcal{L}$, and we use capital letters for the names of the languages (such as Spanning Tree). A distributed language is *constructible* if, for every graph G, there exists x such that $(G, x) \in \mathcal{L}$. It is *Turing decidable* if there exists a (centralized) Turing machine that decides $\mathcal{L}$.

Languages are one of the ways to consider a set of graphs having a property. We may also consider *predicates* to be the key object, and talk about the graphs satisfying some predicate $\mathcal{P}$.

Examples of languages. Let us list a few important languages that appear in this thesis.

- k-coloured: the set of graphs whose nodes are labelled by an integer between 1 and k, called the colour, such that there is no monochromatic edge, that is, no edge with the same colour at both endpoints.

- k-colourable: the set of graphs (with no input labels) such that there exists a proper colouring (that is a colouring with no monochromatic edge). A special case is the language Bipartite, which is the set of graphs that can be 2-coloured.

- Acylicity: the set of trees.

- Acyclic subgraph: the set of graphs where the edges are labelled by a bit, and where the edges with bit 1 (called *selected* edges) form an acyclic subgraph of the communication graph.

- Spanning Tree: the set of graphs, such that the selected edges form a spanning tree of the graph. This name can be shortened into St. (In Chapter 3, two variants, corresponding to two on different encodings of the selected edges, will be defined.)

- Minimum Spanning Tree: the set of graphs, where the set of selected edges form a minimum spanning tree of the graph. A short-cut is Mst.

[2] We may also use the letter ℓ for the node labelling function.

- Symmetry: the set of graphs that have a non-trivial automorphism.

Other examples are: Regular, where in a selected subgraph every node has the same degree; and Leader, where exactly one node is selected. The languages Alos and Amos that are essential in Chapter 4, will be defined there.

Local deterministic decision. A *deterministic local decision algorithm* with radius t, is a local algorithm in which every node v outputs a local decision, *accept* or *reject*, based on all the information contained in its t-neighbourhood. This information consists in: the local structure of the graph, the IDs of the nodes, and the local inputs. The distance t is constant with respect to the size of the network n (except in Chapter 5).

A *basic decision scheme* for a language $\mathcal{L}$ is defined by a deterministic local decision algorithm A, such that, for every configuration: all the nodes accept, if and only if, the configuration belongs to the language $\mathcal{L}$. We say that an algorithm (globally) accepts a configuration if all nodes accept. For a configuration (G, x), we denote the fact that a node v accepts (respectively rejects) by $A(G, v, x) = 1$ (respectively $A(G, v, x) = 0$), and the fact that the whole configuration is accepted (respectively rejected) by $A(G, x) = 1$ (respectively $A(G, x) = 0$).

Proof-labelling schemes. Given a distributed language $\mathcal{L}$, a *proof-labelling scheme* (PLS for short) for $\mathcal{L}$ is a pair prover-verifier $(\mathbf{p}, \mathbf{v})$. The prover $\mathbf{p}$ is an oracle assigning a certificate function $c : V(G) \to \{0,1\}^*$ to every labelled graph (G, x). The verifier $\mathbf{v}$ is a t-round local algorithm, with t being constant,[3] that behaves as a deterministic local decision algorithm, but takes into account the certificates of the prover. That is, the verifier has access to the graph topology, the identifiers, the inputs and the certificates, within its constant radius view. Then for every labelled graph (G, x) the following two conditions should be satisfied:

- If $(G, x) \in \mathcal{L}$ then $\mathbf{v}$ outputs *accept* at every node of G whenever all nodes of G are given the certificates provided by $\mathbf{p}$;
- If $(G, x) \notin \mathcal{L}$ then, for every certificate function $c : V(G) \to \{0,1\}^*$, $\mathbf{v}$ outputs *reject* in at least one node of G.

The first condition guarantees the existence of certificates allowing the given legally labelled graph (G, ℓ) to be globally accepted. The second condition guarantees that the verifier cannot be "cheated", that is, an illegally labelled graph will systematically be rejected by at least one node, whatever "fake" certificates are given to the nodes. Note that the prover and the verifier can access and use the IDs, but the language does not depend on the identifier assignment.

Given an instance (G, x, ID) (where ID denotes the identifier assignment) with (G, x) in the language $\mathcal{L}$, we denote by $\mathbf{p}(G, x, \mathrm{ID}, v)$ the certificate assigned by the

[3] In Chapter 3, we assume that the radius is 1, in order to have cleaner proofs. The topic of Chapter 5, is the impact of allowing non-constant radius.

prover $\mathbf{p}$ to node $v \in V$, and by $|\mathbf{p}(G, x, \mathrm{ID}, v)|$ its size. In a graph all the certificates have the same size, thus this size can be denoted $|\mathbf{p}(G, x, \mathrm{ID})|$. For a given language $\mathcal{L}$, the *certificate size* of a proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for the configurations of size n, is defined as the maximum of $|\mathbf{p}(G, x, \mathrm{ID})|$, taken over all instances (G, x, ID) of size n. It is denoted by $\mathsf{size}(\mathbf{p}, \mathbf{v})$.

Locality. In this dissertation, the words *local* and *locality* can basically refer to three related but different aspects. It will always be clear from the context which definition we are using. First, it can refer to the constraint that the nodes can communicate only with nodes that are close by in the network. In this perspective, the smaller the radius of the view is, the more local the algorithm is. This is the most classic meaning. Second, it can refer to the size of the certificates. As explained in the introduction (paragraph *Proof sizes and measure of the locality*, in Subsection 1.1.4), one can argue that we can classify the languages by locality, stating that the smaller the optimal proof size, the more local the language. Third, in Chapter 4, and in particular in the term *price of locality*, we are dealing with locality of proofs. In a classic proof-labelling scheme, the certificates are local, in the sense that every node has its own local certificate. In this later setting, local certificates are opposed to global certificates that are certificates that every node can access.

Knowledge of n. In general we assume that the nodes *do not* know n. This is crucial for some proofs where we consider instances of different sizes, and argue that they cannot be distinguished. Actually, if we allow certificates of size $\Omega(\log n)$, then the prover can give and certify n (with a spanning tree and counters).

The reader will find the definitions specific to a section, at the beginning of the said section.

Chapter 3

Error-sensitivity

This section is devoted to the study of the *error-sensitivity* of proof-labelling schemes. Basically a proof-labelling scheme is error-sensitive, if on *no*-instances the number of rejecting nodes is lower bounded by the edit distance from the graph to the languages (up to multiplicative constants), regardless of the certificate assignment.

We did an overview of this work in Subsection 1.3.1 of the introduction. It is joint work with Pierre Fraigniaud, and corresponds to the paper [36] published in 2016. Before diving in the technical parts, we restate and develop the motivation behind this study, and give the definitions and cite the related work that are specific to this chapter.

3.1 Introduction

Motivations and specific related work. One weakness of proof-labelling schemes is that they do not allow to distinguish between a configuration which is slightly erroneous, and a global state which is completely bogus. In both cases, it is only required that at least one node detects the illegality of the state. In the latter case though, having only one node raising an alarm, or launching a recovery procedure for bringing the whole system back to a legal state, might be quite inefficient. Instead, if many nodes would detect the errors, then bringing back the system into a legal state will be achieved by a collection of local resets running in parallel, instead of a single reset traversing the whole network sequentially.

Moreover, in several contexts like, e.g., *property-testing*[1], monitoring an error-prone system is implemented via an external mechanism involving a monitor that is probing the system by querying a (typically small) subset of nodes chosen at random. *Non-deterministic* property-testing has been recently investigated in the literature [69, 70, 88], where a certificate is given to the property-testing algorithm. Such a certificate is however global. Global certificates will be the topic of Chapter 4, but here we focus on local proofs. In the context of property-testing, such decentralized

[1] We sketched what property testing is in the introduction. For a proper survey of property-testing on graphs, see [64, 65]. Another type of computation that has a local flavour although it is not fully decentralized is centralized local computing, see e.g., [32, 68, 93].

certificates can be viewed as, say, annotations provided to the nodes of a network, or to the entries of a database. The correction of the network, or of the database, is then checked by a property-testing algorithm querying nodes at random for recovering the individual states of these nodes, including their certificates. To be efficient, such distributed certificates must guarantee that, if the monitored system is far from being correct, then many nodes are capable to detect the error. Indeed, if just one node is capable to detect the error, then the probability that the monitoring system will query that specific node is very low, resulting in a large amount of time before the error is detected.

In this chapter, we aim at designing *error-sensitive* proof-labelling schemes, which guarantee that system states that are far from being correct can be detected by many nodes, providing faster recovery if the error detection mechanism is decentralized, or faster discovery if this error detection mechanism is centralized.

More specifically, the distance between two global states of a distributed system is defined as the *edit-distance* between these two states, i.e., the minimum number of individual states one has to modify in order to move from one global state to the other. A proof-labelling scheme is *error-sensitive* if there exists a constant $\alpha > 0$ such that, for any erroneous system state S, the number of nodes detecting the error is at least $\alpha\, d(S)$, where $d(S)$ is the shortest edit-distance between S and a correct configuration. The choice of a linear dependency between the number of nodes detecting the error, and the edit-distance to legal states is not arbitrary, but motivated by the following two observations.

- On the one hand, a linear dependency is somewhat the best that we may hope for. Indeed, let us consider a k-node network G in some illegal state S for which r nodes are detecting the illegality of S — think about, e.g., the spanning tree predicate. Then, let us make n copies of G and its state S, potentially linked by $n-1$ additional edges if one insists on connectivity. In the resulting kn-node network, we get that $O(rn)$ nodes are detecting illegality, which grows linearly with the number of nodes, as n grows.

- On the other hand, while a sub-linear dependency may still be useful in some contexts, this would be insufficient in others. For instance, in the context of property testing, for systems that are ϵ-far from being correct (i.e., essentially, an ϵ fraction of the individual states are incorrect), the linear dependency enables to find a node capable to detect the error after $O(1/\epsilon)$ expected number of queries to random nodes. Instead, a sub-linear dependency would yield an expected number of queries that grows with the size of the system before querying a node capable to detect the error.

Specific definition. To define the novel notion of *error-sensitive* proof-labelling schemes, we introduce the following notion of distance between labelled graphs. Let ℓ and ℓ' be two (input) labellings of a same graph G. The *edit distance* between (G,ℓ) and (G,ℓ') is the minimum number of elementary operations required to transform

(G, ℓ) into (G, ℓ'), where an elementary operation consists of replacing a node label by another label. That is, the edit distance between (G, ℓ) and (G, ℓ') is simply

$$|\{v \in V(G) : \ell(v) \neq \ell'(v)\}|.$$

The edit-distance from a labelled graph (G, ℓ) to a language $\mathcal{L}$ is the minimum, taken over all labellings ℓ' of G satisfying $(G, \ell') \in \mathcal{L}$, of the edit-distance between (G, ℓ) and (G, ℓ'). Roughly, an error-sensitive proof-labelling scheme satisfies that the number of nodes that reject a labelled graph (G, ℓ) should be (at least) proportional to the distance between (G, ℓ) and the considered language.

Definition 3.1. *A proof-labelling scheme* $(\mathbf{p}, \mathbf{v})$ *for a language* $\mathcal{L}$ *is* error-sensitive *if there exists a constant* $\alpha > 0$*, such that, for every labelled graph* (G, ℓ)*,*

- *If* $(G, \ell) \in \mathcal{L}$ *then* $\mathbf{v}$ *outputs* accept *at every node of* G *whenever all nodes of* G *are given the certificates provided by* $\mathbf{p}$*;*
- *If* $(G, \ell) \notin \mathcal{L}$ *then, for every certificate function* $c : V(G) \to \{0,1\}^*$*,* $\mathbf{v}$ *outputs* reject *in at least* $\alpha\,\mathsf{d}$ *nodes of* G*, where* d *is the edit distance between* (G, ℓ) *and* $\mathcal{L}$*, i.e.,* $\mathsf{d} = \textsc{dist}\big((G, \ell), \mathcal{L}\big)$*.*

Note that the at least $\alpha\,\mathsf{d}$ nodes rejecting a labelled graph (G, ℓ) at edit-distance d from $\mathcal{L}$ do not need to be the same for all certificate functions.

3.2 Basic properties of error-sensitive proof-labelling schemes

Let us first illustrate the notion of error-sensitive proof-labelling scheme with the example of Acyclic Subgraph, that we have already come across in the introduction. Here, we use a notation with pointers (as we will see later, in this chapter the encoding of the input is important). That is, the input of each node v, denoted $\ell(v)$, is either the name of one of its neighbours, or an empty label, and the edges $\{v, \ell(v)\}$ (for non-empty $\ell(v)$) form an acyclic subgraph of G.

We show that Acyclic Subgraph has an error-sensitive proof-labelling scheme. The proof of this result is easy, as the fixing of the labels can be done locally, at the rejecting nodes. Nevertheless, the proposition and its proof serve as a basic example illustrating the notion of error-sensitive proof-labelling scheme.

Proposition 3.1. Acyclic Subgraph *has an error-sensitive proof-labelling scheme.*

Proof. Let $(G, \ell) \in$ Acyclic Subgraph. Every node $v \in V(G)$ belongs to a subtree rooted at a node r such that $\ell(r) = \bot$. The prover $\mathbf{p}$ provides every node v with its distance $d(v)$ to the root of its subtree (i.e., number of hops to reach the root by following the pointers specified by ℓ). The verifier $\mathbf{v}$ proceeds at every node v as follows: first, it checks that $\ell(v) \in N(v) \cup \{\bot\}$; second, it checks that, if $\ell(v) \neq \bot$ then $d(\ell(v)) = d(v) - 1$, and if $\ell(v) = \bot$ then $d(v) = 0$. If all these tests are passed, then v accepts. Otherwise, it rejects. By construction, if (G, ℓ) is acyclic, then all nodes accept with these certificates. Conversely, if there is a cycle C in (G, ℓ), then let v be a node with maximum value $d(v)$ in C. Its predecessor in C (i.e., the node $u \in C$ with $\ell(u) = v$) rejects. So $(\mathbf{p}, \mathbf{v})$ is a proof-labelling scheme for Acyclic Subgraph.

We show that $(\mathbf{p}, \mathbf{v})$ is error-sensitive. Suppose that $\mathbf{v}$ rejects (G, ℓ) at $k \geq 1$ nodes. Let us replace the label $\ell(v)$ of each rejecting node v by the label $\ell'(v) = \perp$, and keep the labels of all other nodes unchanged, i.e., $\ell'(v) = \ell(v)$ for every node where $\mathbf{v}$ accepts. We have $(G, \ell') \in$ Acyclic Subgraph. Indeed, by construction, the label of every node u in (G, ℓ') has a well-formatted label $\ell'(v) \in N(v) \cup \{\perp\}$. Moreover, let us assume, for the purpose of contradiction, that there is a cycle C in (G, ℓ'). By definition, every node v of this cycle is pointing to $\ell'(v) \in N(v)$. Thus $\ell'(v) = \ell(v)$ for every node of C, from which it follows that no nodes of C was rejecting with ℓ, a contradiction with the fact that, as observed before, $\mathbf{v}$ rejects every cycle. Therefore $(G, \ell') \in$ Acyclic Subgraph. Hence the edit-distance between (G, ℓ) and Acyclic Subgraph is at most k. It follows that $(\mathbf{p}, \mathbf{v})$ is error-sensitive, with sensitivity parameter $\alpha \geq 1$. □

The definition of error-sensitivity is based on the existence of a proof-labelling scheme for the considered language. However, two different proof-labelling schemes for the same language may have different sensitivity parameters α. In fact, we show that every non-trivial language admits a proof-labelling schemes which is *not* error-sensitive. That is, the following result shows that demonstrating the existence of a proof-labelling scheme that is *not* error-sensitive for a language does not prevent that language to have another proof-labelling scheme which *is* error-sensitive. We say that a distributed language is *trivially approximable* if there exists a constant d such that every labelled graph (G, ℓ) is at edit-distance at most d from $\mathcal{L}$.

Proposition 3.2. *Let $\mathcal{L}$ be a distributed language. Unless $\mathcal{L}$ is trivially approximable, there exists a proof-labelling scheme for $\mathcal{L}$ that is not error-sensitive.*

Proof idea. The idea is to take an arbitrary scheme and to turn it into a scheme where, whatever the configuration is, the prover has a strategy to make at most one node reject. In this new scheme the prover, in addition to the certificates of the original scheme, will define and certify a spanning tree of the graph and provide a boolean to each node. At every node, the verifier will consider the original certificates, but will not output its decision. Instead it will do a verification of the consistency of the booleans. The boolean corresponds to the predicate: for the current node and its descendant, no node is rejecting the original certificates. Note that the information available to the nodes is sufficient to check the consistency. Then the root of the tree will output the decision that corresponds to its boolean. □

Proof. Let $\mathcal{L}$ be a non trivially approximable distributed language. Given a labelled graph $(G, \ell) \in \mathcal{L}$, let T be a spanning tree of G. It is folklore (cf., e.g., [7, 82], and Section 1.2) that T can be certified by a proof-labelling scheme where the certificate assigned to each node u consists of a pair $(I(u), d(u))$ where $I(u)$ is the ID of a node r picked as the root of T, and $d(u)$ the hop-distance in T from u to r. The verifier checks the distances the same way as it does in the proof of Proposition 3.1 (which guarantees the absence of cycles). In addition, every node checks that it agrees with its neighbours in the graph about the ID of the root (which guarantees that T is not a forest with more than one tree). At every node, if all these tests are passed at that node, then it accepts, else it rejects.

We now prove that every proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for $\mathcal{L}$ can be transformed into a proof-labelling scheme $(\mathbf{p}', \mathbf{v}')$ for $\mathcal{L}$ which is not error-sensitive. On a legal instance $(G, \ell) \in \mathcal{L}$, the prover $\mathbf{p}'$ selects a spanning tree T of G, and provides every node u with its certificate $\mathbf{p}(u)$ for (G, ℓ), with the local description of the tree T together with the appropriate certificate, and with a boolean $b(u)$ set to *true*. The verifier $\mathbf{v}'$ checks the correctness of the spanning tree T, and rejects if it is not correct. From now on, we assume that T is correct. The verifier $\mathbf{v}'$ then outputs accept or reject according to the following rules.

- At every node u distinct from the root of T, $\mathbf{v}'$ accepts if and only if one of the two conditions below is fulfilled:
  - $b(u) = \mathit{false}$, and either $\mathbf{v}$ rejects at u, or a child v of u in T satisfies $b(v) = \mathit{false}$;
  - $b(u) = \mathit{true}$, $\mathbf{v}$ accepts at u, and $b(v) = \mathit{true}$ for every child v of u in T.
- At the root of T, the verifier $\mathbf{v}'$ rejects if and only if
  - $\mathbf{v}$ rejects, or a child v of u satisfies $b(v) = \mathit{false}$.

By construction, if $(G, \ell) \in \mathcal{L}$ then all the nodes accept when provided with the appropriate certificates, because, with these certificates, all booleans b are *true*, and $\mathbf{v}$ accepts at all nodes.

If $(G, \ell) \notin \mathcal{L}$, then $\mathbf{v}'$ rejects in at least one node if the given certificates do not encode a spanning tree T. Therefore, let us assume that the given certificates correctly encode a spanning tree T, rooted at r. Since $(G, \ell) \notin \mathcal{L}$, there exists at least one node where $\mathbf{v}$ rejects. Let u be a node where $\mathbf{v}$ rejects, such that $\mathbf{v}$ rejects at no other nodes on the shortest path from u to r in T. If $u = r$, then, since $\mathbf{v}$ rejects, we get that $\mathbf{v}'$ rejects as well. So, let us assume that $u \neq r$. Let $u_0, u_1, \dots, u_t$ with $u_0 = u$, $t \geq 1$, and $u_t = r$ be the shortest path from u to r in T. For $\mathbf{v}'$ to accept at u_0, it must be the case that $b(u) = \mathit{false}$. The same holds at each node along the path: For $\mathbf{v}'$ to accept at u_i, $i = 0, \dots, t-1$, it must be the case that $b(u_i) = \mathit{false}$. This leads $\mathbf{v}'$ to reject at $u_t = r$. Therefore, $(\mathbf{p}', \mathbf{v}')$ is a proof-labelling scheme for $\mathcal{L}$.

We now show that $(\mathbf{p}', \mathbf{v}')$ is not error-sensitive. Let $(G, \ell) \notin \mathcal{L}$. Let T be a spanning tree of G, rooted at node r. We provide the nodes with the proper description of T and the certificates to certify T. We also provide the nodes with arbitrary certificates for $\mathbf{v}$. Then we provide the nodes with the following "fake" boolean certificates that we assign by visiting the nodes of the tree T bottom-up, as follows. Let u be a node:

- if $\mathbf{v}$ rejects at u or a child v of u in T satisfies $b(v) = \mathit{false}$, then set $b(u) = \mathit{false}$;
- else set $b(u) = \mathit{true}$.

In this way, only the root of T can reject. Therefore, with such certificates, even instances (G, ℓ) that are arbitrarily far from $\mathcal{L}$ will be rejected by a single node. It follows that $(\mathbf{p}', \mathbf{v}')$ is not error-sensitive, as claimed. □

Recall that the fact that every distributed language has a proof-labelling scheme can be established by using a *universal* proof-labelling scheme $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ (see [67] and Subsection 1.2 in the introduction). Given a distributed language $\mathcal{L}$, and a labelled graph $(G, \ell) \in \mathcal{L}$ on an n-node graph G, a universal certificate $c : V(G) \to \{0,1\}^*$ for that labelled graph is defined for every node $u \in V(G)$ by the triple $c(u) = (T, M, L)$ where nodes are ordered from 1 to n in arbitrary order, T is a vector with n entries indexed from 1 to n where $T[i]$ is the ID of the ith node v, $L[i]$ is the label $\ell(v)$ of the ith node v, and M is the adjacency matrix of G. The prover $\mathbf{p}_{univ}$ assigns $c(u)$ to every node $u \in V(G)$. The verifier $\mathbf{v}_{univ}$ then checks at every node u that its certificate is consistent with the certificates given to its neighbours (i.e., they all have the same T, L, and M, the indexes matches with the IDs, and the actual neighbourhood of v is as it is specified in T, L and M). If this test is not passed, then $\mathbf{v}_{univ}$ outputs *reject* at u, otherwise it outputs *accept* or *reject* according to whether the labelled graph described by (M, L) is in $\mathcal{L}$ or not. It is easy to check that $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ is indeed a proof-labelling scheme for $\mathcal{L}$. The universal proof-labelling scheme has the following nice property, that we state as a lemma for further references in the text.

Lemma 3.1. *If a distributed language $\mathcal{L}$ has an error-sensitive proof-labelling scheme, then the universal proof-labelling scheme applied to $\mathcal{L}$ is error-sensitive.*

Proof. Let $(\mathbf{p}, \mathbf{v})$ be an error-sensitive proof-labelling scheme for $\mathcal{L}$, and let $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ be the universal proof-labelling scheme for $\mathcal{L}$. Let $(G, \ell) \notin \mathcal{L}$. We show that $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ is at least as good as $(\mathbf{p}, \mathbf{v})$ with respect to the number of rejecting nodes. Specifically, we show that if $\mathbf{v}_{univ}$ rejects (G, ℓ) at r nodes for some certificate function c, then there exists a certificate function c' such that $\mathbf{v}$ rejects (G, ℓ) in at most r nodes.

Let u be a node in which $\mathbf{v}_{univ}$ accepts (G, ℓ), and let $c(u) = (T, M, L)$ be the certificate of node u leading to this acceptance. Note that it must be the case that $(M, L) \in \mathcal{L}$. We set $c'(u)$ as the certificate assigned to node u by $\mathbf{p}$ in labelled graph (M, L). We do so for all nodes at which $\mathbf{v}_{univ}$ accepts. We then go over every node u at which $\mathbf{v}_{univ}$ rejects (G, ℓ), but that is adjacent to at least one node v at which $\mathbf{v}_{univ}$ accepts (G, ℓ). Let $c(v) = (T, M, L)$ be the corresponding certificates at the accepting node v. As before, we set $c'(u)$ as the certificate assigned to node u by $\mathbf{p}$ in labelled graph (M, L). Note that if u is adjacent to two different nodes v and v' at which $\mathbf{v}_{univ}$ accepts, the two nodes v, and v' share the same certificates (T, M, L). Hence the definition of c' at u is well defined.

Now, we observe that for a node u in which $\mathbf{v}_{univ}$ accepts, its certificate $c(u)$ is consistent with the certificates of all its neighbours, and thus, in particular, u and its neighbours share the same labelled graph (M, L). Therefore, the certificates c' assigned to u and its neighbours are consistent with respect to $\mathbf{v}$. It follows that every node u at which $\mathbf{v}_{univ}$ accepts (G, ℓ) with certificate function c satisfies that $\mathbf{v}$ accepts (G, ℓ) at u with certificate function c'. □

While every distributed language has a proof-labelling scheme, we show, using Lemma 3.1, that there exist languages for which there are no error-sensitive proof-labelling schemes.

Proposition 3.3. *There exist languages that do not admit any error-sensitive proof-labelling scheme.*

We gave an overview of the proof in Subsection 1.3.1 of the introduction, with an general reasoning, and then taking the language of symmetric graphs as an example. The following proof uses a different language.

Proof. We show that there exist languages $\mathcal{L}$ such that, for every proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for $\mathcal{L}$, and every $d \geq 1$, there exists a labelled graph (G, ℓ) at edit-distance at least d from $\mathcal{L}$, and a certificate function c, such that $\mathbf{v}$ rejects (G, ℓ) with certificate c in at most a constant number of nodes. We consider labelled graphs (G, ℓ) where ℓ encodes a subgraph H_ℓ of G as follows: $\ell(u) \subseteq N(u)$ is a list of neighbours of u in G that are adjacent to u in the subgraph H_ℓ. We consider the following language:

$$\textsc{regular} = \{(G, \ell) : H_\ell \text{ is regular.}\}$$

Let us assume, for the purpose of contradiction, that there exists an error-sensitive proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for $\textsc{regular}$. From Lemma 3.1, it follows that the universal scheme $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ is error-sensitive for $\textsc{regular}$. We show that this is not the case.

Let d_1 and d_2 be two distinct integers. Let G_1 be a regular graph of degree d_1, and let G_1' be a copy of G_1. Let $\{u_1, v_1\} \in E(G_1)$, and let $\{u_1', v_1'\}$ be the corresponding edge in G_1'. We construct the graph G_1^* obtained from G_1 and G_1' by removing $\{u_1, v_1\}$ and $\{u_1', v_1'\}$, and adding $\{u_1, u_1'\}$ and $\{v_1, v_1'\}$. By construction, G_1^* is d_1-regular. Similarly, we can construct a d_2-regular graph G_2^* from a d_2-regular graph G_2 and its copy G_2'. We denote by $\{u_2, u_2'\}$ and $\{v_2, v_2'\}$ the edges connecting G_2 to its copy G_2' in G_2^*. For $i \in \{1, 2\}$, let ℓ_i be the labelling of the nodes of G_i^* such that $H_{\ell_i} = G_i^*$. We have

$$(G_1^*, \ell_1) \in \textsc{regular}, \text{ and } (G_2^*, \ell_2) \in \textsc{regular}.$$

Let G_3^* be the graph obtained from G_1 and G_2 by removing $\{u_1, v_1\}$ from G_1, removing $\{u_2, v_2\}$ from G_2, and adding the edges $\{u_1, u_2\}$ and $\{v_1, v_2\}$. Again let us consider the labels ℓ_3 assigned to the nodes of G_3^* with $H_{\ell_3} = G_3^*$. Since $d_1 \neq d_2$, we have

$$(G_3^*, \ell_3) \notin \textsc{regular}.$$

Now let us assign to the nodes of G_1 in G_3^* the certificates assigned by $\mathbf{p}_{univ}$ to the nodes of G_1 in G_1^*. Similarly, let us assign to the nodes of G_2' in G_3^* the certificates assigned by $\mathbf{p}_{univ}$ to the nodes of G_2' in G_2^*. With such certificates, only the nodes u_1, v_1, u_2, and v_2 may reject when running $\mathbf{v}_{univ}$. Therefore, at most $2d_1 + 2d_2 + 4$ nodes reject. On the other hand the distance between (G_3^*, ℓ_3) and $\textsc{regular}$ is at least as large as $\min\{|V(G_1)|, |V(G_2)|\}$. This distance can thus be made arbitrarily large, while the number of rejecting nodes remains constant. Hence, the universal proof-labelling scheme is not error-sensitive. □

Remark. The language REGULAR used in the proof of Proposition 3.3 to establish the existence of languages that do not admit any error-sensitive proof-labelling schemes actually belongs to the class LCL of locally checkable labellings [89]. Therefore, the fact that a language is easy to check locally does not help for the design of error-sensitive proof-labelling schemes.

We complete this warmup section by some observations regarding the encoding of distributed data structures. Let us consider the following two distributed languages, both corresponding to spanning tree. The first language, $\textsc{st}_p$, encodes the spanning trees using pointers to parents, while the second language, $\textsc{st}_l$, encodes the spanning trees by listing all the incident edges of each node in these tree.

$$\begin{aligned}\textsc{st}_p &= \Big\{(G,\ell):\ \forall v \in V(G),\, \ell(v) \in N(v) \cup \{\bot\} \\ &\qquad \text{and } \Big(\bigcup_{v\in V(G)\,:\,\ell(v)\neq\bot} \{v,\ell(v)\}\Big) \text{ forms a spanning tree}\Big\}\end{aligned}$$

$$\begin{aligned}\textsc{st}_l &= \Big\{(G,\ell):\ \forall v \in V(G),\, \ell(v) \subseteq N(v) \text{ and } u \in \ell(v) \text{ iff } v \in \ell(u), \\ &\qquad \text{and } \Big(\bigcup_{v\in V(G)} \bigcup_{u\in\ell(v)} \{u,v\}\Big) \text{ forms a spanning tree}\Big\}.\end{aligned}$$

Obviously, $\textsc{st}_p$ is just a compressed version of $\textsc{st}_l$ as the latter can be constructed from the former in just one round. However, note that $\textsc{st}_p$ cannot be recover from $\textsc{st}_l$ in a constant number of rounds, because $\textsc{st}_p$ provides a consistent orientation of the edges in the tree. It follows that $\textsc{st}_p$ is an encoding of spanning trees which is actually strictly richer than $\textsc{st}_l$. This difference between $\textsc{st}_p$ and $\textsc{st}_l$ is not anecdotal, as we shall prove later that $\textsc{st}_l$ admits an error-sensitive proof-labelling scheme, while we show hereafter that $\textsc{st}_p$ is not appropriate for the design of error-sensitive proof-labelling schemes.

Proposition 3.4. $\textsc{st}_p$ *does not admit any error-sensitive proof-labelling scheme.*

We illustrate the underlying idea of the following proof with Figure 3.1.

Proof. Let P_n be the n-node path $u_1, u_2, \ldots, u_n$ with n even. Let ℓ_0, ℓ_1, and ℓ_2 be labellings defined by $\ell_1(u_i) = u_{i+1}$ for all $1 \leq i < n$, and $\ell_1(u_n) = \bot$; $\ell_2(u_i) = u_{i-1}$ for all $1 < i \leq n$, and $\ell_2(u_1) = \bot$; and $\ell_3(u_i) = u_{i-1}$ for all $1 < i \leq \frac{n}{2}$, $\ell_3(u_i) = u_{i+1}$ for all $\frac{n}{2} + 1 \leq i < n$, and $\ell_3(u_1) = \ell_3(u_n) = \bot$. We have $(P_n, \ell_1) \in \textsc{st}_p$ and $(P_n, \ell_2) \in \textsc{st}_p$, while the distance from (P_n, ℓ_3) to $\textsc{st}_p$ is at least $\frac{n}{2}$. Let $(\mathbf{p}, \mathbf{v})$ be a proof-labelling scheme for $\textsc{st}_p$. Consider the case of (P_n, ℓ_3) where every u_i, $i = 1, \ldots, \frac{n}{2}$, is given the certificate assigned by $\mathbf{p}$ to u_i in (P_n, ℓ_2), and every u_i, $i = \frac{n}{2} + 1, \ldots, n$, is given the certificate assigned by $\mathbf{p}$ to u_i in (P_n, ℓ_1). With such certificates, (P_n, ℓ_3) is rejected by $\mathbf{v}$ at $u_{\frac{n}{2}}$ and $u_{\frac{n}{2}+1}$ only. □

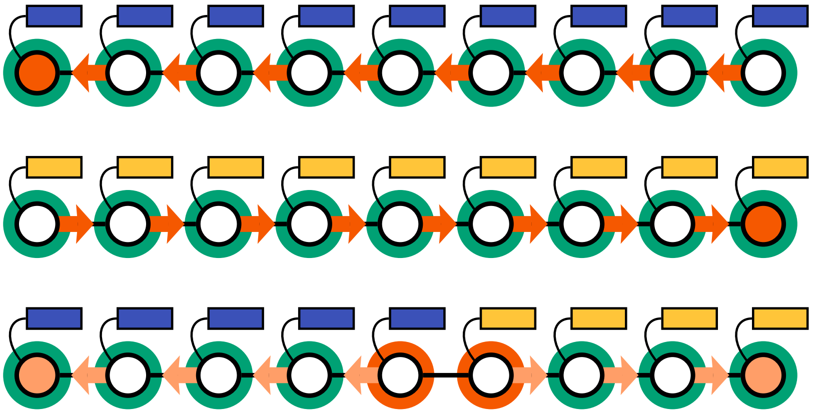


Figure 3.1: We describe the idea of the proof of Proposition 3.4. The two first instances are *yes*-instances for the language (spanning trees encoded with pointers). Thus there exist two certificate assignments (the blue and the yellow ones) such that every node accept in both instances. Consider now the third instance, which is a *no*-instance. By copying the certificates in a smart way, the verifier can make sure that only two nodes will reject.

3.3 Characterization

We now define the notion of *local stability*, which allows us to characterize the distributed languages admitting an error-sensitive proof-labelling scheme. This notion naturally pops up in the context of proof-labelling schemes [82] and locally checkable proofs in general [67]. Indeed, in these latter frameworks, languages that are "hard" to prove, in the sense that they require certificates of large size (typically of $\Omega(n^2)$ bits), are not locally stable, in the sense that glueing together two legal labelled graphs, say by connecting them by an edge, results in a labelled graph which can be very far from being legal. Local stability also naturally pops up in the context of the classical construction tasks which admit local algorithms, such as $(\Delta+1)$-colouring and MIS [87]. Indeed, those tasks share the property that any partial solution can be extended to a larger solution without modifying the already assigned labels. Extending the partial solution actually only depends on the "border" of the current partial solution.

More specifically, let G be a graph, and let H be a subgraph of G, that is, a graph H such that $V(H) \subseteq V(G)$, and $E(H) \subseteq E(G)$. We denote by $\partial_G H$ the set of nodes at the *boundary* of H in G, that is, which belongs to $V(H)$, and are incident to an edge in $E(G) \setminus E(H)$. Given a labelling ℓ of a graph G, and a subgraph H of G, the labelling ℓ_H denotes the labelling of H induced by ℓ restricted to the nodes of H:

$$\ell_H(v) = \begin{cases} \ell(v) & \text{if } v \in V(H) \\ \emptyset & \text{otherwise (where } \emptyset \text{ denotes the empty string).} \end{cases}$$

See Figure 1.9 in the introduction. Roughly, a distributed language $\mathcal{L}$ is locally stable if, by copy-pasting parts of legal labellings with small cuts between these parts, the resulting labelled graph is not too far from being legal. More precisely, let G

be a graph, and let $H_1, \ldots, H_k$ be a family of connected subgraphs of G such that $(V(H_i))_{i=1,\ldots,k}$ is a partition of $V(G)$. For every $i \in \{1, \ldots, k\}$, let us consider a labelled graph $(G_i, \ell_i) \in \mathcal{L}$ such that H_i is a subgraph of G_i. Let ℓ be the labelling of G defined as $\ell = \sum_{i=1}^{k} \ell_i$, i.e. for every $v \in V(G)$, $\ell(v) = \ell_i(v)$ where i is such that $v \in V(H_i)$. We say that such a labelled graph (G, ℓ) is *induced* by the labelled graphs (G_i, ℓ_i), $i = 1, \ldots, k$, *via* the subgraphs $H_1, \ldots, H_k$.

Definition 3.2. *A language $\mathcal{L}$ is* locally stable *if there exists a constant $\beta > 0$, such that, for every labelled graph (G, ℓ) induced by labelled graphs $(G_i, \ell_i) \in \mathcal{L}$, $i = 1, \ldots, k$, via subgraphs $H_1, \ldots, H_k$, the edit-distance between (G, ℓ) and $\mathcal{L}$ is at most $\beta \mid \cup_{i=1}^{k} \partial_G H_i \cup \partial_{G_i} H_i|$.*

That is, the labelled graph resulting from cut-and-pasting parts of legally labelled graphs (G_i, ℓ_i), $i = 1, \ldots, k$, is at edit-distance from $\mathcal{L}$ upper bounded by the number of nodes at the boundary of the subgraphs H_i in G and G_i, and is independent of the number of nodes in each of these subgraphs H_i, $i = 1, \ldots, k$.

We have now all ingredients to state our characterization result:

Theorem 3.1. *Let $\mathcal{L}$ be a distributed language. $\mathcal{L}$ admits an error-sensitive proof-labelling scheme if and only if $\mathcal{L}$ is locally stable.*

Proof idea. We first show that having an error-sensitive proof-labelling scheme implies local stability. Consider a labelled graph (G, ℓ) induced by labelled graphs (G_i, ℓ_i) in $\mathcal{L}$. As the (G_i, ℓ_i) are *yes*-instances, for each of them there exists a certificate assignment such that all nodes accept. Consider a proof assignment for (G, ℓ) that is a patch-work of these assignments for the $((G_i, \ell_i))_i$. When we run the verifier, only the nodes at the border will reject. Because of the error-sensitivity, this implies that the distance from the instance to the language is roughly the number of border nodes. Thus the language is locally stable.

For the other direction, we consider the universal scheme. Remember that in this scheme, the verifier considers its certificate as a description of the current labelled graph, check that the described graph is in the language, and that it is consistent with its neighbourhood. The argument is the following. Loosely speaking, if there is an area of the graph where the nodes accept, then it means that the nodes agree on the on the description of the graph, and that this is a graph in the language. In general, there are several such areas in the graph, each of them with a distinct graph description. Only the nodes at the border of these areas reject (because they realize that the prover does not provide the same description to every node). These areas can be interpreted as the component for local stability, and the description correspond to the different $((G_i, \ell_i))_i$. As only the border nodes are rejecting, the distance must be small, because we assume local stability. This in turn implies that the universal scheme is error-sensitive. □

Proof. We first show that if a distributed language $\mathcal{L}$ admits an error-sensitive proof-labelling scheme then $\mathcal{L}$ is locally stable. So, let $\mathcal{L}$ be a distributed language, and let $(\mathbf{p}, \mathbf{v})$ be an error-sensitive proof-labelling scheme for $\mathcal{L}$ with sensitivity parameter α. Let (G, ℓ) be a labelled graph induced by labelled graphs $(G_i, \ell_i) \in \mathcal{L}$, $i = 1, \ldots, h$, via the subgraphs $H_1, \dots, H_h$ for some $h \geq 1$. Since, for every $i \in \{1, \dots, h\}$, $(G_i, \ell_i) \in \mathcal{L}$, there exists a certificate function c_i such that $\mathbf{v}$ accepts at every node of (G_i, ℓ_i) provided with the certificate function c_i. Now, let us consider the labelled graph (G, ℓ), with certificate $c_i(u)$ on every node $u \in V(H_i)$ for all $i = 1, \dots, h$. With such certificates, the nodes in $V(H_i)$ that are not in $\partial_G H_i \cup \partial_{G_i} H_i$ have the same close neighbourhood in (G, ℓ) and in (G_i, ℓ_i). Therefore, they accept in (G, ℓ) the same way they accept in (G_i, ℓ_i). It follows that the number of rejecting nodes is bounded by $| \cup_{i=1}^{h} \partial_G H_i \cup \partial_{G_i} H_i|$, and therefore (G, ℓ) is at edit-distance at most $\frac{1}{\alpha} | \cup_{i=1}^{h} \partial_G H_i \cup \partial_{G_i} H_i|$ from $\mathcal{L}$. Hence, $\mathcal{L}$ is locally stable, with parameter $\beta = \frac{1}{\alpha}$.

It remains to show that if a distributed language is locally stable then it admits an error-sensitive proof-labelling scheme. Let $\mathcal{L}$ be a locally stable distributed language with parameter β. We prove that the universal proof-labelling scheme $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ for $\mathcal{L}$ (cf. Section 3.2) is error-sensitive for some parameter α depending only on β. Let $(G, \ell) \notin \mathcal{L}$, and let us fix some certificate function c. The verifier $\mathbf{v}_{univ}$ rejects in at least one node. We show that if $\mathbf{v}_{univ}$ rejects at k nodes, then the edit-distance between (G, ℓ) and $\mathcal{L}$ is at most k/α for some constant $\alpha > 0$ depending only on β. For this purpose, let us consider the outputs of $\mathbf{v}_{univ}$ applied to (G, ℓ) with certificate c, and let us define the graph G' as the graph obtained from G by removing all edges for which $\mathbf{v}_{univ}$ rejects at both extremities. Note that the graph G' may not be connected.

Let C be a connected component of G', with at least one node u at which $\mathbf{v}_{univ}$ accepts. Let $c(u) = (T, M, L)$ be the certificate of node u, as it should be in the universal proof-labelling scheme as described in section 3.2. Since $\mathbf{v}_{univ}$ accepts at u, node u shares the same triple (T, M, L) with all its neighbours in G', as $\mathbf{v}_{univ}$ would reject at u otherwise. Similarly, for every neighbour v of u, it must be the case that v agrees on (T, M, L) with each of its neighbours w in G', as otherwise $\mathbf{v}_{univ}$ would have rejected at both v and w, and the edge $\{v, w\}$ would have been removed from G. It follows that all nodes in C share the same triple (T, M, L) as the one given to the accepting node u. Also (M, L) coincides with the local structure of C and its labelling ℓ at all accepting nodes in C. Moreover, since $\mathbf{v}_{univ}$ accepts at u, we have $(M, L) \in \mathcal{L}$. We denote by (G_C, ℓ_C) this labelled graph in $\mathcal{L}$.

Let C be a connected component of G' where all nodes reject. In fact, by construction, such a component is composed of just one isolated node. For every such isolated rejecting node u, let us denote by (G_C, ℓ_C) a labelled graph composed of a unique node, with ID equal to the ID of u, and with labelling $\ell_C(u)$ such that $(G_C, \ell_C) \in \mathcal{L}$.

Let $\mathcal{C}$ be the set of all connected components of G'. Let (G, ℓ') be the graph induced by labelled graphs (G_C, ℓ_C) via the subgraphs $C \in \mathcal{C}$. Note that (G, ℓ) and (G, ℓ') coincide, but for the isolated rejecting nodes. By local stability, (G, ℓ') is at edit-distance at most $\beta\, |\cup_{C \in \mathcal{C}} \partial_G C \cup \partial_{G_C} C|$ from $\mathcal{L}$. Now, the nodes in $\cup_{C \in \mathcal{C}} \partial_G C \cup \partial_{G_C} C$ are exactly the rejecting nodes. Thus the number k of rejecting nodes satisfies $k = | \cup_{C \in \mathcal{C}} \partial_G C \cup \partial_{G_C} C|$, and the edit-distance from (G, ℓ') to $\mathcal{L}$ is at most $\beta\, k$. On the other hand, by construction, the edit-distance between (G, ℓ') and (G, ℓ) is at most the number of isolated rejecting nodes, that is, at most k. Therefore, the edit-distance between (G, ℓ) and $\mathcal{L}$ is at most $(\beta + 1)\, k$. Thus, the universal proof-labelling scheme is error-sensitive, with parameter $\alpha = \frac{1}{\beta+1}$. □

Proposition 3.3 can be viewed as a corollary of Theorem 3.1 as it is easy to show that REGULAR is not locally stable. Nevertheless, local stability may not always be as easy to establish, because it is based on merging an arbitrary large number of labelled graphs. We thus consider another property, called *strong local stability*, which is easier to check, and which provides a sufficient condition for the existence of an error-sensitive proof-labelling scheme. Given two labelled graphs (G, ℓ) and (G', ℓ'), and a subgraph H of both G and G', the labelling $\ell - \ell_H + \ell'_H$ for G is the labelling such that, for every node $v \in V(G)$, $(\ell - \ell_H + \ell'_H)(v) = \ell'_H(v)$ if $v \in V(H)$, and $(\ell - \ell_H + \ell'_H)(v) = \ell(v)$ otherwise.

Definition 3.3. *A language $\mathcal{L}$ is* strongly locally stable *if there exists a constant $\beta > 0$, such that, for every graph H, and every two labelled graphs $(G, \ell) \in \mathcal{L}$ and $(G', \ell') \in \mathcal{L}$ admitting H as a subgraph, the labelled graph $(G, \ell - \ell_H + \ell'_H)$ is at edit-distance at most $\beta\,|\partial_{G'} H + \partial_G H|$ from $\mathcal{L}$.*

The following lemma states that strong local stability is indeed a notion that is at least as strong as local stability. For convenience, we change the notation of the labellings from ℓ_i to $\ell^{(i)}$.

Lemma 3.2. *If a language $\mathcal{L}$ is strongly locally stable, then it is locally stable.*

Proof idea. The main difference between the two notions is that in the strongest definition, there are only two labellings that come into play, whereas in the classic definition, there is an arbitrary number of labellings. (Another more subtle aspect is that the definition of the border is not exactly the same.) The idea of the proof is to take an arbitrary good labelling of G, and to replace it little by little by labellings from the (G_i, ℓ_i). At each step only two labellings come into play. The end result is (G, ℓ) and its distance to the original good labelling of G, is at most the size of the union the borders of each step. This union is actually the set of border nodes for the classic definition. □

Proof. Let us consider a strongly locally stable language $\mathcal{L}$, with parameter β, and a labelled graph (G, ℓ) induced by labelled graphs $(G_i, \ell^{(i)}) \in \mathcal{L}$, $i = 1, \ldots, h$, via the subgraphs $H_1, \ldots, H_h$. Let $\rho^{(0)}$ be a labelling of G such that $(G, \rho^{(0)}) \in \mathcal{L}$. We iteratively relabel every node of H_i, $i = 1, \ldots, h$, by their corresponding labels in $\ell^{(i)}$, starting from $(G, \rho^{(0)})$. More precisely, let us first consider $(G, \rho^{(0)} - \rho^{(0)}_{H_1} + \ell^{(1)}_{H_1})$. H_1 is a subgraph of both G and G_1. Therefore, since $\mathcal{L}$ is strongly locally stable, we get

$$\textsc{dist}\left(\left(G, \rho^{(0)} - \rho^{(0)}_{H_1} + \ell^{(1)}_{H_1}\right), \mathcal{L}\right) \leq \beta\,|\partial_G H_1 \cup \partial_{G_1} H_1|.$$

Let $\rho^{(1)}$ be a labelling of G such that $(G, \rho^{(1)}) \in \mathcal{L}$ and

$$\textsc{dist}\left(\left(G, \rho^{(0)} - \rho^{(0)}_{H_1} + \ell^{(1)}_{H_1}\right), \left(G, \rho^{(1)}\right)\right) \leq \beta\,|\partial_G H_1 \cup \partial_{G_1} H_1|.$$

Let us assume that, for some $j \in \{1 \ldots h-1\}$, we have already established the existence of labellings $\rho^{(i)}$ of G, $i = 1, \ldots, j$, such that, for every $i = 1, \ldots, j$,

$$\textsc{dist}\left(\left(G, \rho^{(i-1)} - \rho^{(i-1)}_{H_i} + \ell^{(i)}_{H_i}\right), \left(G, \rho^{(i)}\right)\right) \leq \beta\,|\partial_G H_i \cup \partial_{G_i} H_i|$$

with $(G, \rho^{(i)}) \in \mathcal{L}$. Again, since $\mathcal{L}$ is locally stable, we get that

$$\textsc{dist}\Big(\big(G, \rho^{(j)} - \rho^{(j)}_{H_{j+1}} + \ell^{(j+1)}_{H_{j+1}}\big), \mathcal{L}\Big) \leq \beta\, |\partial_G H_{j+1} \cup \partial_{G_{j+1}} H_{j+1}|.$$

We set $\rho^{(j+1)}$ as a labelling of G such that $(G, \rho^{(j+1)}) \in \mathcal{L}$, and

$$\textsc{dist}\Big(\big(G, \rho^{(j)} - \rho^{(j)}_{H_{j+1}} + \ell^{(j+1)}_{H_{j+1}}\big), \big(G, \rho^{(j+1)}\big)\Big) \leq \beta\, |\partial_G H_{j+1} \cup \partial_{G_{j+1}} H_{j+1}|.$$

In this way, we construct a sequence of labellings $\rho^{(1)}, \dots, \rho^{(h)}$ such that, for every $i = 1, \dots, h$, $(G, \rho^{(i)}) \in \mathcal{L}$, and

$$\textsc{dist}\Big(\big(G, \rho^{(i-1)} - \rho^{(i-1)}_{H_i} + \ell^{(i)}_{H_i}\big), \big(G, \rho^{(i)}\big)\Big) \leq \beta\, |\partial_G H_i \cup \partial_{G_i} H_i|.$$

Now, since $\ell_{H_1} = \ell^{(1)}_{H_1}$, we get that, restricted to H_1,

$$\textsc{dist}\Big(\big(G, \ell\big), \big(G, \rho^{(0)} - \rho^{(0)}_{H_1} + \ell^{(1)}_{H_1}\big)\Big) = 0.$$

Therefore, restricted to H_1,

$$\textsc{dist}\Big(\big(G, \ell\big), \big(G, \rho^{(1)}\big)\Big) \leq \beta |\partial_G H_1 \cup \partial_{G_1} H_1|.$$

It follows that, restricted to $H_1 \cup H_2$,

$$\textsc{dist}\Big(\big(G, \ell\big), \big(G, \rho^{(1)} - \rho^{(1)}_{H_2} + \rho^{(2)}_{H_2}\big)\Big) \leq \beta |\partial_G H_1 \cup \partial_{G_1} H_1|.$$

as a consequence, we get that, restricted to $H_1 \cup H_2$,

$$\textsc{dist}\Big(\big(G, \ell\big), \big(G, \rho^{(2)}\big)\Big) \leq \beta\Big(|\partial_G H_1 \cup \partial_{G_1} H_1| + |\partial_G H_2 \cup \partial_{G_2} H_2|\Big).$$

More generally, restricted to $H_1 \cup \dots \cup H_h$,

$$\textsc{dist}\Big(\big(G, \ell\big), \big(G, \rho^{(h)}\big)\Big) \leq \beta \sum_{i=1}^{h} \Big(|\partial_G H_i \cup \partial_{G_i} H_i|\Big).$$

Since the sets $\partial_G H_i \cup \partial_{G_1} H_i$, $i = 1, \dots, h$, are disjoint, and since $V(G) = \cup_{i=1}^{h} V(H_i)$, it follows that

$$\textsc{dist}\Big(\big(G, \ell\big), \big(G, \rho^{(h)}\big)\Big) \leq \beta\, \Big|\bigcup_{i=1}^{h} \big(\partial_G H_i \cup \partial_{G_i} H_i\big)\Big|.$$

That is, $\mathcal{L}$ is locally stable, as desired. □

In fact, strong local stability is a notion strictly stronger than local stability, although they coincide on bounded-degree graphs.

Proposition 3.5. *There are languages that are locally stable but not strongly locally stable. However, all locally stable languages on bounded degree graphs are strongly locally stable.*

Proof. To show that there are languages that are locally stable but not strongly locally stable, we consider the following language $\mathcal{L}$. In $\mathcal{L}$, the labelling ℓ describes a set of edges H_ℓ, and a colouring of each node in blue or red, where H_ℓ must be made of subgraphs that are stars, and every star must be monochromatic. This language has a proof-labelling scheme. On legal instances, the prover assigns to every center u of a star a certificate $c(u) = 0$, and to every other node u of a star a certificate $c(u) = 1$. All the others are given an empty certificate. The verifier checks that a node with a certificate 1 has exactly one neighbour in H_ℓ, that this neighbour has certificate 0 and that it has the same colour. Also it checks that not two adjacent nodes can have certificate 0. This is a proof-labelling scheme for $\mathcal{L}$, which is error-sensitive. Indeed, just like for the language ACYCLIC SUBGRAPH, one can fix the labels locally, by removing the faulty edge from H_ℓ. By Theorem 3.1, this language is locally stable. However, it is not strongly locally stable. Indeed, consider a first instance (G, ℓ) that is a star on n nodes with only blue nodes, and a second instance (G, ℓ'), on the same graph, but with only red nodes. Now consider $(G, \ell - \ell_H + \ell'_H)$, where H contains the center, and has half of the nodes of G. This instance is at distance roughly $n/2$ from $\mathcal{L}$, while $\partial_{G'} H + \partial_G H$ contains just a single node, the center.

We now show that all locally stable languages on bounded degree graphs are strongly locally stable. Let $\Delta \geq 1$, and let $\mathcal{F}_\Delta$ be the family of graphs with maximum degree Δ. Let $\mathcal{L}$ be a locally stable language on graphs in $\mathcal{F}_\Delta$. Let us consider a connected graph H, and two labelled graphs $(G, \ell) \in \mathcal{L}$ and $(G', \ell') \in \mathcal{L}$, with $G \in \mathcal{F}_\Delta$, and $G' \in \mathcal{F}_\Delta$, both admitting H as a subgraph. Let $(G, \ell - \ell_H + \ell'_H)$ be the labelled graph induced by labelled graphs (G, ℓ) and (G', ℓ') via the subgraph H. We view $(G, \ell - \ell_H + \ell'_H)$ as induced by (G, ℓ) and (G', ℓ') via the subgraphs $G \setminus H$ and H. By local stability, we get that the distance from $(G, \ell - \ell_H + \ell'_H)$ to $\mathcal{L}$ is at most $\beta \,|\, (\partial_G H \cup \partial_{G'} H) \cup (\partial_G (G \setminus H) \cup \partial_G (G \setminus H)) \,|$. Now, $|\partial_G (G \setminus H)| \leq \Delta |\partial_G H|$, because each edge from the cut $(H, G \setminus H)$ must have an endpoint in H and these endpoints have at most degree Δ. As a consequence the distance from $(G, \ell - \ell_H + \ell'_H)$ to $\mathcal{L}$ is at most $\beta(\Delta + 1)|\partial_G H \cup \partial_{G'} H|$, and the strong stability follows. □

Thanks to the characterization in Theorem 3.1, and to the sufficient condition of Lemma 3.2, we immediately get error-sensitiveness for the language

$$\text{LEADER} = \big\{(G, \ell) : \forall v \in V(G),\ \ell(v) \in \{0, 1\} \\ \text{and there exists a unique } v \in V(G) \text{ for which } \ell(v) = 1\big\}.$$

Corollary 3.1. LEADER *admits an error-sensitive proof-labelling scheme.*

Also, one can show that the language ST_l of spanning trees, whenever encoded by adjacency lists, admits an error-sensitive proof-labelling scheme. This is in contrast to Proposition 3.4.

Corollary 3.2. ST_l *admits an error-sensitive proof-labelling scheme.*

Proof idea. The proof use the following natural path. In the definition of strong local stability we consider a labelling $(G, \ell - \ell_H + \ell'_H)$. This labelling corresponds to a tree, when restricted to H, or to $G \setminus H$. The hybrid labelling may contain cycles and disconnected components. Then one has to show that the number of cycles or distinct connected components is upper bounded by the number of nodes in the border, up to multiplicative constants. □

Proof. We show that ST_l is strongly locally stable. Let us consider two labelled graphs $(G, \ell) \in \text{ST}_l$ and $(G', \ell') \in \text{ST}_l$, both admitting H as a subgraph. We show that $(G, \ell - \ell_H + \ell'_H)$ is not far from $\mathcal{L}$. For this purpose, we aim at modifying the labels of few nodes so that to form a spanning tree of G. First, for every node $u \in \partial_G H \cup \partial_{G'} H$, we modify $\ell'_H(u)$ such that the label of u becomes consistent with its neighbourhood in G. That is, all edges listed in the label exist in G, and they match edges listed by the neighbours of u in G. After this modification, which impacts only $|\partial_G H \cup \partial_{G'} H|$ nodes, the resulting labelling of the nodes in G encodes a set of edges $F \subseteq E(G)$. However, F may not be a spanning tree, and it may include cycles, and may even be not connected.

Let $\widehat{G}$ be the graph obtained from G after we remove all edges in $E(H)$, and all nodes in $V(H) \setminus (\partial_G H \cup \partial_{G'} H)$. Note that $V(H) \cup V(\widehat{G}) = V(G)$ and $V(H) \cap V(\widehat{G}) = \partial_G H \cup \partial_{G'} H$. The set F is equal to the union of the edges described by ℓ on $\widehat{G}$, and of the edges described by ℓ' on H. Indeed consider an edge $e \in F$. If both endpoints of e are in $\widehat{G}$, then this edge is encoded by ℓ at its two endpoints, as the labels of these endpoints are copied from ℓ, and the modification of $\ell - \ell_H + \ell'_H$ performed at the nodes in $\partial_G H \cup \partial_{G'} H$ does not impact such nodes. If e has both endpoints in $H \setminus (\partial_G H \cup \partial_{G'} H)$ then, by the same reasoning, this edge is encoded by ℓ' at its two endpoints. If e has both endpoints in $\partial_G H \cup \partial_{G'} H$, then the modification of $\ell - \ell_H + \ell'_H$ performed at the nodes in this latter set did not affected edge e, which implies that e was originally encoded in ℓ'. Finally, if e has one endpoint in $\partial_G H \cup \partial_{G'} H$, and the other one outside $\partial_G H \cup \partial_{G'} H$, then, from by the modification of $\ell - \ell_H + \ell'_H$, the edge e was present in ℓ in at least one of its extermities.

As ℓ is the labelling of a spanning tree of G, F restricted to $\widehat{G}$ is a spanning forest of $\widehat{G}$. Similarly, as ℓ' is a spanning tree of G', F restricted to H is a spanning forest of H. Also, since $V(\widehat{G}) \cap V(H) = \partial_G H \cup \partial_{G'} H$, it follows that, in both forests, every tree contains a node of $V(\widehat{G}) \cap V(H)$. Let us denote by $n_{\widehat{G}}$, $m_{\widehat{G}}$, and $s_{\widehat{G}}$ the number of nodes, edges, and connected components of F restricted to $\widehat{G}$, respectively. Similarly, let us denote by n_H, m_H, and s_H the same parameters for H. Since the connected components of F restricted to $\widehat{G}$, and to H, are forests, we get that:

$$m_{\widehat{G}} = n_{\widehat{G}} - s_{\widehat{G}} \text{ and } m_H = n_H - s_H \tag{3.1}$$

Moreover, since each connected component contains a node of the border, we get

$$s_{\widehat{G}} \leq |V(\widehat{G}) \cap V(H)| \text{ and } s_H \leq |V(\widehat{G}) \cap V(H)|. \tag{3.2}$$

Now, let us consider the whole set F, and let us define n_F, m_F, and s_F as the number of nodes, edges, and connected components of F, respectively. By definition,

$m_F = m_{\widehat{G}} + m_H$. Thus, by Eq. (3.1), we get that

$$m_F = n_{\widehat{G}} + s_{\widehat{G}} + n_H + s_H.$$

Moreover, by definition, $n_F = n_{\widehat{G}} + n_H - |V(\widehat{G}) \cap V(H)|$. Therefore,

$$m_F = n_F + |V(\widehat{G}) \cap V(H)| + s_{\widehat{G}} + s_H.$$

We can now bound the number of edges that we need to remove from F in order to get a spanning forest (with the same number of connected components). For such a forest, it must hold that its number of edges, m, satisfies $m = n_F + s_F$. Therefore,

$$\begin{aligned} m_F - m &= (n_F + |V(\widehat{G}) \cap V(H)| + s_{\widehat{G}} + s_H) - (n_F + s_F) \\ &\leq |V(\widehat{G}) \cap V(H)| + s_{\widehat{G}} + s_H \\ &\leq 3|V(\widehat{G}) \cap V(H)|, \end{aligned}$$

where the last equality holds by Eq. (3.2). Thus, by removing at most $3|\partial_G H \cup \partial_{G'} H|$ edges from F, we get a spanning forest of G with at most $|\partial_G H \cup \partial_{G'} H|$connected components. Therefore, by adding $|\partial_G H \cup \partial_{G'} H| - 1$ edges, one can construct a spanning tree of G. So, in total, transforming F into a spanning tree required to modify at most $4|\partial_G H \cup \partial_{G'} H|$ edges. This may impact the labels of at most $8|\partial_G H \cup \partial_{G'} H|$ nodes. As the labels of the nodes in $\partial_G H \cup \partial_{G'} H$ were also modified at the very beginning of the construction, it follows that the number of node labels impacted by our spanning tree construction is at most $9|\partial_G H \cup \partial_{G'} H|$. It follows that $\textsc{st}_l$ is strongly locally stable with parameter at most 9, which implies that it admit an error-sensitive proof-labelling scheme with sensitivity parameter at least $\frac{1}{9}$. □

Also, Theorem 3.1 allows us to prove that minimum-weight spanning tree (MST) is error-sensitive (whenever the tree is encoded locally by adjacency lists). More specifically, let

$$\textsc{mst}_l = \Big\{(G, \ell) : \forall v \in V(G),\, \ell(v) \subseteq N(v) \text{ and } \Big(\bigcup_{v \in V(G)} \bigcup_{u \in \ell(v)} \{u, v\}\Big) \text{ forms a MST}\Big\}. \tag{3.3}$$

Corollary 3.3. $\textsc{mst}_l$ *admits an error-sensitive proof-labelling scheme.*

Proof idea. This proof is more involved than the ones for simple spanning trees. This is because minimum spanning trees are more constrained than spanning trees, hence, changing a part of the graph may change the minimum spanning tree much more than it changes a spanning tree. The key claim is that actually there is another type of stability of minimum spanning tree. More precisely, consider two graphs with a subgraph H in common. There exists two subsets of edges $S_{\max}$ and $S_{\min}$, with $S_{\min} \subseteq S_{\max}$ such that: (1) the MSTs of the two graphs restricted to H, call them $(M_i)_i$, are such that: $S_{\min} \subseteq M_i \subseteq S_{\max}$, and (2) the difference between $|S_{\max}|$ and $|S_{\min}|$ can basically be bounded by the number of border nodes involved. To establish this result, we analyse the runs of Kruskal algorithm on the different graphs at hand. □

Proof. We show that $\textsc{mst}_l$ is strongly locally stable. Let us consider a graph H, and two labelled graphs $(G,\ell) \in \textsc{mst}_l$ and $(G',\ell') \in \textsc{mst}_l$ admitting H as a subgraph. We show that the labelled graph $(G, \ell - \ell_H + \ell'_H)$ is not far from $\textsc{mst}_l$. Let T be the spanning tree of G defined by the set of edges defined by ℓ, and let T' be the spanning tree of G' defined by the set of edges defined by ℓ'. Let F the edge set defined by $\ell - \ell_H + \ell'_H$ on G, after the same modification of that labelling on the nodes of $\partial_G H \cup \partial_{G'} H$ as in the proof of Corollary 3.2, i.e., the labels of $\partial_G H \cup \partial_{G'} H$ are modified so that the adjacency lists of these nodes in their labels match the labels of their neighbours. Let $\widehat{G}$ be the graph defined as in the proof of Corollary 3.2, that is, $\widehat{G}$ is the graph obtained from G after removing all edges in $E(H)$, and all nodes in $V(H) \setminus (\partial_G H \cup \partial_{G'} H)$. Note that F is obtained from the union of the two forests that came form ℓ and ℓ', on $E(\widehat{G})$ and $E(H)$, respectively. Hence, every connected component of F contains a node in $\partial_G H \cup \partial_{G'} H$.

Recall that Kruskal algorithm constructs an MST by considering the edges in increasing order of their weights, and by adding the currently considered edge to the current MST if and only if this edge does not create a cycle with the previously added edges. It is known that every MST of a graph can be generated by Kruskal algorithm, by breaking ties between edges of identical weight in a way to add all edges of the desired MST. Let $\mathcal{O}$ be the ordering of the edges of G that leads to the tree T, and let $\mathcal{O}'$ be the ordering of the edges of G' that leads to the tree T'. Let $\mathcal{O}'_H$, be the same ordering as $\mathcal{O}'$ but restricted to the edges of H.

Let G_1 be the graph obtained from H by adding a new node u connected to every node of $\partial_G H + \partial_{G'} H$ by edges with weights smaller than the smallest weight in $E(G)$ and in $E(G')$. Let $\mathcal{O}_1$ be the ordering of $E(G_1)$ obtained by concatenating $\mathcal{O}'_H$ to an arbitrary ordering of the edges incident to u. Let T_1 be the MST of G_1 that Kruskal algorithm constructs in G_1 when it uses the ordering $\mathcal{O}_1$. Also let G_2 be a copy of H, let T_2 be the MST constructed by Kruskal algorithm on G_2 using $\mathcal{O}_2 = \mathcal{O}'_H$. Finally, we define the ordering $\mathcal{O}_3$ of the edges of G as the ordering such that the edges of $E(\widehat{G})$ appear in the same order as in $\mathcal{O}$, the edges of $E(H)$ appear in the same order as in $\mathcal{O}'$, and the edges of $E(T) \cap E(\widehat{G})$ have priority. Let T_3 be the spanning tree defined by Kruskal algorithm on G with $\mathcal{O}_3$. T_3 is necessarily equal to T on the edges of $\widehat{G}$ because they are MST of the same graph, and because the edges of $E(T) \cap E(\widehat{G})$ have priority in $\mathcal{O}_3$.

Claim 3.1. *The following inclusions hold.*

$$E(T_1) \cap E(H) \subseteq E(T') \cap E(H) \subseteq E(T_2) \cap E(H).$$

$$E(T_1) \cap E(H) \subseteq E(T_3) \cap E(H) \subseteq E(T_2) \cap E(H).$$

Before proving Claim 3.1, let us show how to complete the proof using that Claim. By Claim 3.1, on H, T_3 can be transformed into T' by changing only edges of $E(T_2) \setminus E(T_1)$. Moreover $E(T_2) \cap E(H)$ and $E(T_1) \cap E(H)$ are a spanning forests of H with at most $|\partial_G H \cup \partial_{G'} H|$ trees in it, because, as in the proof of Corollary 3.2, every tree contains at least a node of $\partial_G H \cup \partial_{G'} H$. We get that

$$|(E(T_2) \cap E(H)) \setminus (E(T_1) \cap E(H))| \leq |\partial_G H \cup \partial_{G'} H|.$$

Therefore, restricted to the graph H, the tree T_3 can be transformed into the tree T' by adding or removing at most $|\partial_G H \cup \partial_{G'} H|$ edges. Now, as T_3 is equal to T on $\widehat{G}$, $E(T_3)$ can be transformed into F by changing at most $|\partial_G H \cup \partial_{G'} H|$ edges. Thus F is at edit-distance at most $2|\partial_G H \cup \partial_{G'} H|$ from a minimum spanning tree of G. Since the modification we made at the very beginning to ensure the consistency of the labels affected at most $|\partial_G H \cup \partial_{G'} H|$ nodes, it follows that the edit-distance from $(G, \ell - \ell_H + \ell'_H)$ to the language is most $3|\partial_G H \cup \partial_{G'} H|$, and thus the language is strongly locally stable.

It just remains to prove Claim 3.1. We show the two sets of inclusion at once. Let M be either $E(T')$ or $E(T_3)$, and let Ω be the corresponding ordering of the edges leading to T' or T_3. Note that, by construction Ω, $\mathcal{O}_1$, and $\mathcal{O}_2$ are consistent on the edges that they have in common, i.e., on all the edges of $E(H)$. Let $\mathcal{O}_{\text{tot}}$ be an ordering that is consistent with the three orderings Ω, $\mathcal{O}_1$ and $\mathcal{O}_2$. We can run Kruskal algorithm on the three instances G, G_1 and G_2 with $\mathcal{O}_{\text{tot}}$. Let $M_1^{(i)}$, $M_2^{(i)}$ and $M^{(i)}$, be the subset of edges in $E(T_1)$, $E(T_2)$, and M, respectively, that have been added by Kruskal algorithm to the current tree before considering the ith edge in $\mathcal{O}_{\text{tot}}$. We show, by induction on i, that the three following properties hold for every $i \geq 1$:

P1: $M_1^{(i)} \cap E(H) \subseteq M^{(i)} \cap E(H) \subseteq M_2^{(i)} \cap E(H)$;

P2: if two nodes of H are linked by a path in $M_2^{(i)}$ then they are also linked by a path in $M^{(i)}$;

P3: if two nodes of H are linked by a path in $M^{(i)}$ then they are also linked by a path in $M_1^{(i)}$.

These properties are trivially true for $i = 1$, as all sets $M_1^{(1)}$, $M_2^{(1)}$ and $M^{(1)}$ are empty. Suppose that P1, P2, and P3 hold are true for $i-1$, and consider i-th edge $e = \{u, v\}$ considered by Kruskal algorithm in $\mathcal{O}_{\text{tot}}$ for T_1, T_2 and T' or T_3. We consider two cases.

Consider first the case where $e \notin E(H)$. Then e appears either only in $\mathcal{O}_1$, or only in Ω. If e appears only in $\mathcal{O}_1$, then independently of whether Kruskal algorithm takes e or not, the three properties P1, P2, and P3 hold for i. If e appears only in Ω, then, clearly, P1 and P2 hold for i. The only scenario for which P3 may become wrong for i is if e is added to M, and this addition creates a new path between two nodes x and y of H, while there are no paths between x and y in $M_1^{(i)}$. Let us show that this does not happen. Indeed, since $e \notin E(H)$, such a path must pass through the border of H, which is included in $\partial_G H \cup \partial'_G H$ (this holds for both choices for M, that is, either $E(T')$ or $E(T_3)$). In particular adding e to the set of edges taken by Kruskal algorithm so far connects two nodes of the border of H. Now, all the nodes of $\partial_G H \cup \partial'_G H$ are already connected in $M_1^{(i)}$. Indeed, the edges of $E(G_1) \setminus E(H)$ have smaller weights. Therefore, all the nodes of $\partial_G H \cup \partial'_G H$ are connected in $M_1^{(i)}$, and thus it is not possible that there is a path created by adding e in M that does not already exists in $M_1^{(i)}$.

Second, consider the case $e \in E(H)$. Then e appears in all the orderings. Let us consider two subcases depending on whether or not e is taken in M.

- If e is taken in M, then e is not closing a cycle in $M^{(i-1)}$, and thus, thanks to P2, e is not closing any cycle in $M_2^{(i-1)}$ either. Thus e is also taken in T_2, and P1 holds. P2 still holds as well since e is added to both sets. If e is taken in T_1 then P3 holds. Instead if e is not taken in T_1, then its two extremities were already linked by a path, and P3 also holds.

- If e is not taken in M, then e closes a cycle in $M^{(i-1)}$. Therefore, by P3, e also closes a cycle in $M_1^{(i-1)}$, and thus it is not taken in T_1 either, and P1 holds. P3 still holds as we have added no edges to M. If e is not taken in T_2 then P2 holds. And if e is taken in T_2, then the fact that e is not taken in M implies that the nodes were already connected, and thus again P2 holds.

This completes the proof of Claim 3.1, and thus the proof of Corollary 3.3. □

3.4 Compact error-sensitive proof-labelling schemes

The characterization of Theorem 3.1 together with Lemma 3.1 implies an upper bound of $O(n^2)$ bits on the certificate size for the design of error-sensitive proof-labelling schemes for locally stable distributed languages. In this section, we show that the certificate size can be drastically reduced in certain cases. It is known that spanning tree and minimum spanning tree have proof-labelling schemes using certificates of polylogarithmic size, $\Theta(\log n)$ bits [7, 82], and $\Theta(\log^2 n)$ bits [79], respectively. We show that the proof-labelling schemes for spanning tree and MST in [7, 79, 82] are actually error-sensitive.

Recall that Proposition 3.4 proved that spanning tree does not admit any error-sensitive proof-labelling schemes whenever the tree is encoded at each node by a pointer to its parent: ST_p does not have any error-sensitive proof-labelling scheme. However, we show that ST_l, i.e., the language of spanning trees encoded by adjacency lists, does have a very compact error-sensitive proof-labelling scheme.

Theorem 3.2. ST_l *has an error-sensitive proof-labelling scheme with certificates of size* $O(\log n)$ *bits.*

Proof. We show that the classical proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for ST_l is error-sensitive. We illustrate the proof with an example, see figure 3.2 for the graph we consider.

On instances of the language, i.e., on labelled graphs (G, ℓ) where ℓ encodes a spanning tree T of G, the prover $\mathbf{p}$ chooses an arbitrary root r of T, and then assigns to every node u a certificate $(I(u), P(u), d(u))$ where $I(u) = \mathsf{ID}(r)$, $P(u)$ is the ID of the parent of u in the tree (or $ID(u)$ if u is the root), and $d(u)$ the hop-distance in the tree from u to r. The verifier $\mathbf{v}$ at every node u first checks that:

- the adjacency lists are consistent, that is, if u is in the list of v, then v is in the list of u;

- there exists a neighbour of u with ID $P(u)$, we denote it $p(u)$;

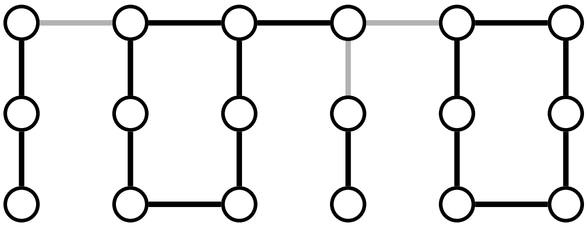


Figure 3.2: We consider the graph above. The edges of the input are black, the edges that are in the communication graph, but not in the input, are grey. The input does not describe a spanning tree, as the subgraph has two cycles, and is not even connected.

- the node u has the same root-ID $I(u)$ as all its neighbours in G;
- $d(u) \geq 0$.

Then, the verifier checks that:

- if $\mathsf{ID}(u) \neq I(u)$ then $d(p(u)) = d(u) - 1$, and for every other neighbour w in ℓ, $d(w) = d(u) + 1$ and $p(w) = u$;
- if $\mathsf{ID}(u) = I(u)$ then $P(u) = ID(u)$, $d(u) = 0$, and every neighbour w of u in ℓ satisfies $d(w) = d(u) + 1$ and $p(w) = u$.

See figures 3.3 and 3.4 for an example of certificate assignment, and the behaviour of the verifier.

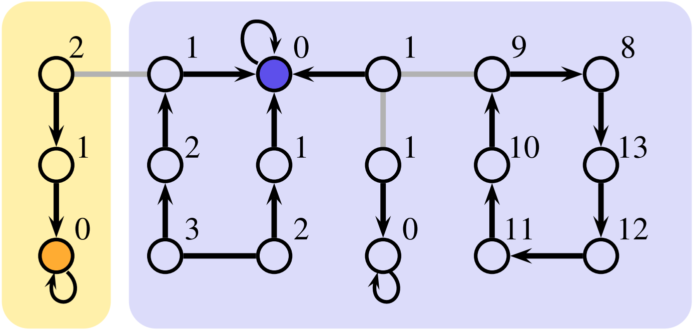


Figure 3.3: This figure illustrates the certificates provided by the prover. The numbers are the distances, and the arrows show the orientation of the pointers. The nodes in the blue rectangle have the ID of the dark blue node as their root-ID in the certificates. The ones in the yellow rectangle have the ID of the orange node.

By construction, if $(G, \ell) \in \mathrm{ST}_l$, then $\mathbf{v}$ accepts at every node. Also, it is easy to check that if $(G, \ell) \notin \mathrm{ST}_l$, then, for every certificate function c, at least one node rejects.

To establish error-sensitivity for the above proof-labelling scheme, let us assume that $\mathbf{v}$ rejects at $k \geq 1$ nodes with some certificate function c. Then, let (G', ℓ') be the labelled graph coinciding with (G, ℓ) except that all edges for which $\mathbf{v}$ rejects at

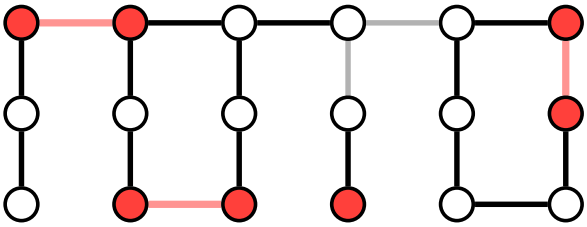


Figure 3.4: In this picture, the red nodes are the ones that reject, when the verifier is run on the graph with the certificates of the previous figure. The red edges have both endpoints rejecting. The nodes on the top left corner reject because they do not have the same root-ID. The ones on the bottom, linked by an edge, reject, because they detect that the edge linking them is in the input but is not oriented. The last node on the bottom rejects because it has distance zero and a pointer to itself, as if it were the root, but its ID is not the root-ID of the certificate. Finally, the endpoints of the edge on the right reject, because the distances are not consistent with the pointers.

both endpoints are removed both from G, and from the adjacency lists in ℓ of the endpoints of these edges. Note that modifying ℓ into ℓ' only requires to edit labels of nodes that are rejecting.

The graph G' may be disconnected. Let (C, ℓ'_C) be a connected component of (G', ℓ'). See figure 3.5.

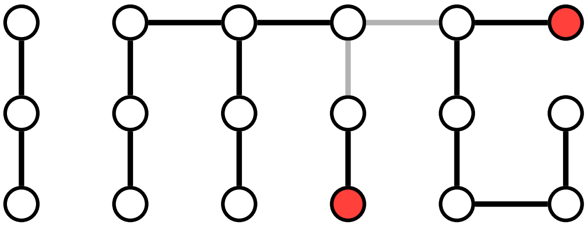


Figure 3.5: After removing the edges whose two endpoints reject, we are left with the graph of this last figure. Note that the communication graph now has two connected components. Also remark that the edges of the input that are left, form a forest. When running the verifier again on this graph, some nodes still reject, they are coloured in red. These nodes are the ones such that: (1) if we consider the orientation given by the certificates, they have the role of the root of a tree, but (2) they do not have the ID that corresponds to root-ID. By putting back the edges of the communication graph, only three edges are needed to get a proper spanning tree.

We claim that the edges of ℓ'_C form a forest in C. First note that if there is a cycle in the edges of ℓ'_C, then this cycle already existed in ℓ because we have added no edges when transforming ℓ into ℓ'. Consider such a cycle in ℓ, and the certificates given by $\mathbf{p}$. Either an edge is not oriented, that is no node uses this edge to point to its parent, or the cycle is consistently oriented and then distances are not consistent. In both cases two adjacent nodes of the cycle would reject when running $\mathbf{v}$. Then this cycle cannot be present in ℓ'_C, as at least one edge has been removed. As a consequence ℓ'_C form a forest of C. In C, if a node is connected to no other node by an edge of ℓ'_C, we will consider it as a tree of one node. With this convention, ℓ' is a spanning forest of G'.

We will now bound the number of trees in ℓ' by a function of k. The number of trees in ℓ' is equal to the sum of the number of trees in each component (C, ℓ'_C).

Let us run $\mathbf{v}$ on graph (C, ℓ'_C), and let k_C be the number of rejecting nodes. Observe that for every two nodes u and v in a component C, it holds that $I(u) = I(v)$. Indeed, otherwise, there would exist two adjacent nodes u and v in C with $I(u) \neq I(v)$, resulting in $\mathbf{v}$ rejecting at both nodes, which would yield the removal of $\{u, v\}$ from G. Consequently, at most one tree of ℓ'_C has a root whose ID corresponds to the ID given in the certificate. Then the number of trees in ℓ'_C is bounded by $k_C + 1$, and the total number of trees is bounded $\sum_C k_C + 1 = (\sum_C k_C) + |C|$.

Note that because of the design of the proof-labelling scheme, the nodes that accept when running $\mathbf{v}$ on (G, ℓ) also accept in (G', ℓ'). Then $\sum_C k_C \leq k$.

Let V_C be the set of nodes of C. It is easy to see that for all C, there exists a node of V_C that rejects when we run $\mathbf{v}$ on (G, ℓ). Indeed if there is no rejecting node, then no edge between C and the rest of the graph is removed, and then there is only one component in the graph. But then all node accept, which contradict the fact that $k \geq 1$. Then $|C| \leq k$.

So overall all ℓ' encodes a spanning forest with at most $2k$ trees. Such a labelling can thus be modified to get a spanning tree by modifying the labels of at most $4k$ nodes. That is, $(\mathbf{p}, \mathbf{v})$ is error-sensitive with parameter $\alpha \geq \frac{1}{4}$. □

Finally, we show that the compact proof-labelling scheme in [79, 82] for minimum-weight spanning tree, as specified in Eq. (3.3) of Section 3.3 is error-sensitive when the edge weights are distinct.

Theorem 3.3. MST$_l$ *admits an error-sensitive proof-labelling scheme with certificates of size* $O(\log^2 n)$ *bits.*

Hereafter, we provide first a sketch of proof for Theorem 3.3, and then the proof itself.

Sketch of proof. A classic proof-labelling scheme for MST (see, e.g., [75, 79, 82]) consists in encoding a run of Borůvka algorithm. Recall that Borůvka algorithm maintains a spanning forest whose trees are called fragments, starting with the forest in which every node forms a fragment. The algorithm proceeds in a sequence of steps. At each step, it selects the lightest outgoing edge from every fragment of the current forest, and adds all these edges to the MST, while merging the fragments linked by the selected edges. This algorithm eventually produces a single fragment, which is a MST of the whole graph, after at most a logarithmic number of steps.

At each node u, the certificate of the scheme consists of a table with a logarithmic number of fields, one for each round of Borůvka algorithm. For each step of Borůvka algorithm, the corresponding entry of the table provides a proof of correctness for the fragment including u, plus the certificate of a tree pointing to the lightest outgoing edge of the fragment. The verifier verifies the structures of the fragments, and the fact that no outgoing edges from each fragment have smaller weights than the one given in the certificate. It also checks that the different fields of the certificate are consistent (for instance, it checks that, if two adjacent nodes are in the same fragment at step i, then they are also in the same fragment at step $i+1$).

To prove that this classic scheme is error-sensitive, we perform the same decomposition as in the proof of Theorem 3.2, removing the edges that have both endpoints rejecting. We then consider each connected component C of the remaining graph, together with the subgraph S of that component described by the edges of the given labelling. In general, S is not a MST of the component C (S can even be disconnected). Nevertheless, we can still make use of the key property that the subgraph S is not arbitrarily far from a MST of the component C. Indeed, the edges of S form a forest, and these edges belong to a MST of the component. As a consequence, it is sufficient to add a few edges to S for obtaining a MST. To show that S is indeed not far from being a MST of C, we define a relaxed version of Borůvka algorithm, and show that the labelling of the nodes corresponds to a proper run of this modified version of Borůvka algorithm. We then show how to slightly modify both the run of the modified Borůvka algorithm, and the labelling of the nodes, to get a MST of the component. Finally, we prove that the collection of MSTs of the components can be transformed into a MST of the whole graph, by editing a few node labels only. □

Proof. Let G be an edge-weighted graph. For simplicity we assume that all the edge-weights are distinct, and thus the MST is unique. Recall that the sequential version of Borůvka algorithm for constructing a MST maintains a forest initially composed of n trees (called fragments), each reduced to just one node, and proceeds in phases where, at each phase, one fragment is picked, and the edge with minimum weight incident to that fragment is added to the forest, resulting in reducing the number of fragments by one, until a single fragment remains, which forms a MST. As shown in, e.g., [95], one can run a parallel version of Borůvka algorithm which proceeds in at most $\lceil \log_2 n \rceil$ rounds, where each round consists in merging fragments in parallel. Note that a merging may involve more than just two fragments during a single round, so the number of fragments may actually decrease faster than by a factor 2 at each round.

We show that the proof-labelling scheme for MST described in [79, 82] is error-sensitive. Recall that, in this proof-labelling scheme, the prover essentially encodes at each node the run of the parallel version of Borůvka algorithm. More specifically, the certificate at each node u is divided into $\lceil \log_2 n \rceil$ fields, one for each round $i = 1, \dots, \lceil \log_2 n \rceil$, plus an additional one. Each field corresponding to a round in u's certificate contains (1) a rooted tree spanning the fragment including u at round i, pointing at an arbitrary node of the fragment, whose ID we call the ID of the fragment, with its proof, (2) another rooted tree, also spanning the fragment but pointing to the endpoint of the lightest edge e outgoing the fragment, with its proof and (3) the ID of the other endpoint of the edge e, and its weight. The former spanning tree is used to ensure the connectivity of the fragment, while the latter spanning tree is used to make sure that the edge e is truly the edge of minimum weight incident to the fragment. Also a last field is added, with the spanning tree of the whole network using exactly the edges of the labelling (that should span all the network) and its proof.

The verifier checks that, for each round, the two spanning trees of the fragment are correct. It also checks that the run is consistent, that is: (1) two adjacent nodes with same fragment ID at some round have the same fragment ID and the same lightest outgoing edge for all further rounds, (2) if an edge is used to merge two fragments at some round, then its endpoints belong to the same fragment for all remaining rounds, and (3) if a spanning tree is pointing to an edge, then this edge exists and is used to merge the fragment with another fragment, (4) the final spanning tree has exactly the edges of the labelling, and correctly span the whole graph, that is all the nodes have the same root-ID for this tree. It is proved in [79, 82] that this approach results in a proof-labelling scheme for MST.

We show that the above proof-labelling scheme is error-sensitive. Let us assume that $k \geq 1$ nodes reject with some certificate function c. We perform the same decomposition as in the proof of Theorem 3.2, removing from G and ℓ the edges whose two extremities are rejecting. We obtain a labelled graph (G', ℓ'). Let C be a connected component of G'. Let us run the verifier on (C, ℓ'_C) with the same certificate function, and let k_C be the number of rejecting nodes in C. As before the number of rejecting node in the whole graph can only decrease from (G, ℓ) to (G', ℓ'), thus $\sum_C k_C \leq k$.

Consider a node that is rejecting, it can be rejecting either because the certificate of a round i is not correct (*eg* a spanning tree is broken, or condition (3) does not hold), or because the consistency conditions between the rounds (conditions (1), (2) and (4)) are not fulfilled.

We claim that, because of the decomposition step, the only cases for which a node rejects are the ones were, either it has no parent in one of the trees and is not identified as the root (it does not have the correct root-ID or distance), or it is the correct root but it does not have the edge that is announced in the certificate.

We can use the same line of argument as the proof of Theorem 3.2: if another type error exists, then there exists an edge such that both endpoints will witness the error, and then such an edge cannot exist in (C, ℓ_C) because it would have been removed when doing the decomposition.

We will now consider a relaxed version of Borůvka algorithm that we call *lazy* Borůvka. This algorithm does not produce an MST in general.

As the classical version of Borůvka, the lazy variant grows a forest of fragments. Initially, there is one fragment per node. At each round, lazy Borůvka proceeds in three steps. First it picks an arbitrary *name* for each fragment. Second, for each fragment F, it considers all edges connecting F to a fragment with different name, and either chooses the incident edge with smallest weight, or do not choose any edge, in which case we say that F is *skipping its turn.* Third it merges the fragments that are linked by edges selected during the second step. The algorithm stops if all the fragments have the same name.

Note that in general lazy Borůvka does not produce an MST and can even not terminate. But if, for each round, the names assigned to adjacent fragments are distinct, and that there is no round i, such that every fragment skip at every round after i, then lazy Borůvka eventually produces an MST.

Given a fragment F, we refer to all fragments including F during the further rounds of lazy Borůvka as its *successors*. The fragments of the previous rounds that F contains are called *predecessors*. Also, we call *cluster* a maximal set of adjacent fragments having the same name during a same round. We consider the following two properties of a run of lazy Borůvka:

1. During the run, if some fragments form a cluster, then all their successors will also be part of a same cluster, but will remain different fragments.

2. At every round of the run, at most one fragment per cluster chooses an edge.

We show that ℓ'_C corresponds to the outcome of a run of lazy Borůvka with these properties.

Claim 3.2. *The labelling ℓ'_C is the outcome of a correct run R of lazy Borůvka on C, and this run satisfies the properties 1 and 2.*

Proof of the claim. Let us show that ℓ'_C is the outcome of a correct run R of the algorithm on C. To do so we will use the certificates. Consider an execution of lazy Borůvka where fragments are as described by the certificates, and the names of the fragments are the root-ID of the corresponding fragments. As we argued before, these fragments are well-defined, that is they are trees, with correct proof, and the same root-ID at every node. Moreover these fragments are consistent from a round to the next round, because they satisfy the consistency properties checked by the verifier. The fact that the root-ID may not be the ID of the root of the tree is not a problem, as it corresponds to a name. Finally recall that if a node of C rejects when checking round i, this is because, that node has no parent in a tree encoded in the certificate, and either it does not have the appropriate root ID, or it is not incident to the appropriate edge. In both cases, there are no outgoing edges corresponding to that fragment for Round i, that we interpret as the fact that this fragment skipped its turn at this round.

Thus the different steps are valid for lazy Borůvka and correspond to ℓ'_C. We now prove that the run has property 1 and 2, and at the end we will show that the termination is also correct.

For the first property, let us assume, for the purpose of contradiction, that at some round in R two adjacent fragments F_1 and F_2 have the same name, but two successors F'_1 and F'_2 have different names. Then, when verifying the certificates, the both endpoints of an edge e connecting F_1 to F_2 would reject, as the certificates describe this run, and as the verifier checks that the rounds are consistent. There is no such edge e in C by construction of G', thus this situation does not occur. Also if the two successors F'_1 and F'_2 are identical, then at some round the certificates describe that a fragment is taking an edge to a fragment that has the same root-ID, which is impossible (as such an edge would have been removed when creating G'). These arguments generalize to cluster, by connectivity. As for the second property, suppose that at some round i two fragments of a cluster choose an edge. It means that in the certificates of this run, there were two fragments with correct spanning trees pointing to these edges. As the verifier checks that two adjacent nodes with the same root-ID have the same outgoing edge, either the outgoing edge e was the same in the certificates of the two fragments, and then at least one of them will take no edge because it will detect that it does not have the edge e, or this edge was different for the two fragments and then all the edges between these fragments would have both endpoints rejecting and then they would not be adjacent as these edges would have been removed. These arguments generalize to cluster, by connectivity.

To conclude, the termination of this run is also correct with respect to the specification of lazy Borůvka. This is because of two facts. First in the certificate, the last field describes a tree that has the same root-ID on every node, and the verifier checks this, thus it holds after the decomposition step. Thereafter, the run stops at a round where all the fragments have the same name. Second, suppose there was a round i before the last round described by the certificate at which all the fragments had the same name. Then because of property 1, at round i the fragments were exactly the same as in the last round, and every node has skipped for all the next rounds. Thus we can consider the round i is the last round, it still holds that the run corresponds to ℓ_C, and has property 1 and 2. This completes the proof of Claim 3.2. $\diamond$

In general, when it stops, lazy Borůvka can produce a forest which is arbitrarily far from being an MST. Nevertheless, we show that as the run R satisfies the properties 1 and 2, the forest produced is at distance at most $O(k_C)$ from an MST of C, where k_C is the number of rejecting node in C. To do so we will modify the run R, by applying iteratively an operation on the run, adding edges to ℓ'_C. We do so until we get to a run where at every round, not two adjacent fragments have the same name, that is a run that builds an MST.

We now describe the operation that we can apply to a run and the labelling associated with the run. Consider the first round for which there is a cluster with more than one fragment, and let K be such a cluster. There are only a few cases to consider.

- Case 1: none of the fragments in K is choosing an edge although there are fragments with names different from the one of K, adjacent to K. In this case, we assign new distinct names for this round to all of the fragments in K, making sure that these names are not already used at that round by other fragments (that is we use fresh names).

- Case 2: one of the fragments of K is choosing an edge, which has minimum weight among all edges that connect that fragment to the other fragments of C, including the fragments of K. In this case, we replace the names of the other fragments of K by fresh names.

- Case 3: a fragment F of the cluster K is choosing an edge e, although the lightest edge outgoing from F is an edge e' that connects it to a fragment F' of K. In this case, we add a round between round $i-1$ and round i where all fragments of C are given distinct names, and every fragment is skipping its turn except F, which is choosing the edge e'. Also we add this edge e' to the labelling.

- Case 4: round i is the last round. In this case, we have only one cluster K containing all the remaining fragments. We consider the lightest edge e connecting two fragments in K, and add a round between round $i-1$ and round i where all fragments of C are given distinct names, and every fragment is skipping its turn except F, which is choosing the edge e. Also we add this edge e to the labelling.

Claim 3.3. *If we have a correct run of lazy Borůvka that satisfies property 1 and 2 and that corresponds to the current labelling, then the operation preserves these properties.*

Proof of the claim. Suppose we have a correct run of lazy Borůvka that satisfies property 1 and 2 and that corresponds to the current labelling. Consider the four cases of the operation.

- Case 1. The run is still correct after the renaming because: the fragments of K were skipping, thus the modification of the names does not affect their behaviour, and the behaviour of the other fragments is still the same because we have chosen fresh names (in particular if at round i a fragment $F \notin K$ chooses an edge to a fragment $F' \in K$, then this action is still valid as F and F' still have different names). The outcome and the property 2 are still correct as we have not changed the fact that the nodes are skipping, nor the labelling. Finally, property 1 holds because we have considered the first round with a cluster of more than one fragment, so the predecessors of the fragments of K had different names at the previous rounds.

- Case 2. The same line of reasoning as in the previous case holds: the behaviour is unchanged, the change of name does not affect the correctness of the actions of neither the fragments of K nor the ones outside K, and the property 1 holds because we consider the first round.

- Case 3. Consider first the round that we have added. The fragment F chooses the lightest edge to a fragment that has a different name, because we have chosen different names for all the fragments. Then this round is correct for lazy Borůvka algorithm. Now we have to check that the next rounds are correct. By merging two fragments, we may have created several kinds of problems. First the name of this fragment could be badly defined as the names of the successors of these fragments can be different. But this cannot be the case because property 1 holds in the run before the modification. Second, this merged fragment could take two edges at the same round, one taken by the successor of F before the operation, and one taken by the successor of F' before the operation. This also cannot happen, because of property 2. Finally the behaviour of the other fragments is unchanged has they only consider the names of their adjacent fragments, and that we have not changed these names. Therefore we still have a correct run. We have added the new edge in the labelling thus the run still describes the labelling at hand. The property 1 and 2 still hold for the round we have added, and also hold for the next rounds, as we have just merged two fragments of the same cluster, thus the names are unchanged, and if two fragments of a cluster are now choosing an edge at the same round then it means that this also happens in the run before the operation, which a contradiction.

- Case 4. The same kind of reasoning as for the previous case holds.

This case analysis completes the proof of the claim. $\Diamond$

Because of claim 3.2, the labelling ℓ'_C is the outcome of a correct run R of lazy Borůvka on C, and this run satisfies the properties 1 and 2, therefore we can apply the operation on it. We claim that we can iterate this operation, and get a run R in which there are no clusters with more than just one fragment, after a finite number of iterations. To prove this claim let us first prove that after an iteration for which we have used case 3 or case 4, the number of fragments in the final cluster has decreased by one. Consider the two fragments that we have merged during the operation. In the run before the operation, these two fragments had successors that were never merged, because of property 1. Thus the successors were distinct fragments at the end. Now that we have merged them, they form only one fragment at the end. As the behaviour of the other fragments is not affected by the change, the number of fragment at the end has decreased by one. Let us now prove that for the cases 1 and 2, the sum, over the rounds, of the number of clusters with more than one fragment has strictly decreased. This is easy to see, as we have scattered such a cluster in both cases, without creating new ones. Also for cases 1 and 2, the number of fragments in the final cluster remains unchanged. Therefore at every step, either the number of fragment in the final cluster has decreased by one, or it remains unchanged and the sum over the runs of the number of clusters with more than one fragment has strictly decreased. As these two quantities must be non-negative, the operation can be repeated only finite amount of time. Finally after all these operations, the run is such that at every round not two adjacent fragments have the same name, thus (the modified) ℓ'_C is a spanning tree of C.

We have added exactly one edge everytime we have decreased by one the number of fragments in the final cluster. Thus the number of edges added is the number of fragments in the final cluster in the original run R, minus one. This number is at most k_C. Indeed, at most one fragment contains no rejecting nodes because only one fragment can have the node whose ID was used as the root-ID in the certificates, and all the roots of the other fragments will reject, and there are k_C rejecting nodes. Therefore the distance (in the number of modified edges) between the original ℓ'_C and the modified ℓ'_C that is a correct spanning tree of C, is at most k_C. As the same reasoning holds for every connected component, if we define the spanning tree of a disconnected graph as the union of the spanning trees of its connected component, then the modified $\ell' = \cup_C \ell' C$ is the spanning tree of G', and it is at distance at most $\sum_C k_C \leq k$ from the original $\ell' = \cup_C \ell'_C$ (in the number of modified edges).

We now compare (the modified) ℓ' with the spanning tree of the original graph G.

We claim that the set of edges described by ℓ' can be transformed into a spanning tree of G by adding or removing at most $2k$ edges. Remember that from G to G' we have only removed the edges that were between two of the rejecting nodes. Let us call S this set of edges. We go backwards and remove edges from G to go to G' keeping track of the spanning tree. Among the edges of S at most $k-1$ can be part of the MST of G, because otherwise there would be a cycle as there are only k rejecting nodes. Removing the other edges from G does not change the MST. Let G^1 be G without these edges, and let us also remove them from S. Now let us consider one of the remaining edges of S, that we call $e = (u, v)$. Let G^2 be the graph G^1 without this edge e. If removing e disconnects the graph, then the spanning tree of G^2 if the same as the one of G^1, without e. If it does not, then we define e' as the edge of smallest weight in the cut between the nodes that are closer to u in the tree, and the ones that are closer to v in the tree. Let us check that the new spanning tree is minimum by checking the cycle property: if for every cycle in the graph, the heaviest edge is not part of the spanning tree then the spanning tree is minimum. The only cycles we have to consider are the ones that contain e'. Suppose that the edge e' is the heaviest of a cycle in G^2. This cycle must cross the cut with another edge, and this edge must have a smaller weight, otherwise e' would not be the heaviest, but this contradicts the definition of e', thus by adding e' we have a spanning tree of G^2. We can iterate this construction until there is no more edges in S. At the end $G^k = G'$ and we know that the spanning tree of G' is (the modified) ℓ'. We have added or removed at most $2k$ edges.

To conclude, in the first step from (G, ℓ) to (G', ℓ'), we have edited only the labels of the rejecting nodes, thus k labels. Then we got from each ℓ'_C to the final ℓ'_C by adding at most k_C edges, thus in the whole graph we have modified at most $2k$ labels. And in the last step we have added or removed at most $2k$ edges, thus modified at most $4k$ labels. Thus in total we have edited at most $7k$ labels. Thus the distance is at most linear in the number of rejecting nodes and the proof-labelling scheme is error-sensitive. □

Chapter 4

Uniformity

In this chapter, we study the impact of using global proofs instead of local proofs. In the usual proof-labelling schemes, every node is given its own proof, whereas here we will consider that every node can access the same global proof. This is equivalent to the constraint that the scheme should be *uniform*, that is, that all the certificates should be equal. The comparison between the two models provides insights on the locality of the certification.

This chapter is surveyed in Subsection 1.3.2 of the introduction. It is based on [39], which is joint work with Juho Hirvonen. Actually, a section of [39] has been moved to another chapter. Namely, a section devoted to the similarity between hierarchies defined in communication complexity and distributed decision has been moved to Chapter 6, which is chapter focused on hierarchies.

4.1 Introduction

Motivation and specific related work. The first proof-labelling schemes were designed in the context of self-stabilizing algorithms, where a distributed algorithm would, in addition to the output, keep some information to verify that the state of the network is not corrupted. Similar scenarios exist for global proofs. For example, one may consider a network where the machines compute in a distributed fashion, but an external operator with a view of the whole network can once in a while broadcast a piece of information, such as the name of a leader. As one expects this type of update to be costly, the focus is on minimizing the size of such broadcast information.

From a more theoretical perspective, global proofs are a natural alternative form of non-determinism. Moreover, in proof-labelling schemes a part of the certificate is often global. For example, the name of a leader is given to all nodes. Global proofs can be used to study how much of such redundant information a local proof must have. Finally, one may consider that global proofs are the most natural equivalent of classical non-determinism: only the algorithm is distributed and we ask what is the cost of distributing the proof.

The idea of a prover for computation in a network, or in a system with several computational units, appears outside of distributed computing, and usually with a global proof. In property testing, models where a prover provides a certificate to the machine that queries the graph have been considered [69, 70, 88]. In two-party communication complexity, non-determinism can come as a global proof that both players can access [8].

Specific definitions. Usual proof-labelling schemes use *(purely) local proof*, that is, the prover provides every node with its own certificate. In this chapter, we denote the size of each individual proof in an optimal such scheme by $s_\ell(n)$. We introduce *(purely) global proofs*, where the prover provides a single certificate, and every node can access it. Its minimum size is denoted by $s_g(n)$. Finally, in *mixed proofs*, the prover provides a global proof and local proofs. The size considered, denoted by $s_m(n)$, is the sum of the size of the global proof, and the size of the concatenation of the local proofs in an optimal scheme.

4.2 The price of locality

In this section, we study the size of local, mixed and global proofs for different problems, and the price of locality that follows.

4.2.1 Proof sizes

We first prove some general inequalities between the sizes of the different proof sizes. We then define and discuss the novel notion of *price of locality.*

Theorem 4.1. *For any language, the optimal proof sizes respect the following inequalities.*

$$s_\ell(n) \leq s_m(n) \leq s_g(n) \tag{4.1}$$

$$s_m(n) \leq n \cdot s_\ell(n) \tag{4.2}$$

$$s_g(n) \leq n \cdot s_\ell(n) + O(n \log n). \tag{4.3}$$

Proof. The first line of inequalities mainly follows from the definitions. Suppose one is given a mixed certificate for a language, with local certificates of size $f(n)$ each, and a global certificate of size $g(n)$. The size of this mixed certificate is $s_m(n) = n \cdot f(n) + g(n)$. Then one can create a local proof of size $f(n) + g(n)$, by giving to every node its local part concatenated with the global part. Thus $s_\ell(n) \leq s_m(n)$. The inequality $s_m(n) \leq s_g(n)$ holds because the mixed proof is a generalization of the global proof. Similarly, if there exists local certificates of size $s_\ell(n)$, then one can use them in the mixed model. The size measured in the mixed model will then be $n \cdot s_\ell(n)$. Finally, given local certificates, one can craft a global certificate. The global certificate consists in a list of couples, each couple being an ID and the local certificate of the node with this ID. The size is in $n \cdot s_\ell(n) + O(n \log n)$ because identifiers are on $O(\log n)$ bits. □

4.2.2 Price of locality

We define the *Price of Locality* for distributed proofs, by analogy with the *Price of Anarchy* in algorithmic game theory [84, 92]. Note that this is not the same as the price of locality that appears in the title of [85]. The price of locality (PoL) of a language is defined as the ratio between the size of the concatenation of the purely local certificates, divided by the size of the mixed certificate. That is:

$$PoL(n) = \frac{n \cdot s_\ell(n)}{s_m(n)}.$$

It may come as a surprise that we use mixed proofs instead of global proofs for this definition. There are several reasons for this. First, the inequalities above insure that with this definition the ratio is between 1 and n, whereas with global proofs the price could be smaller than 1, thus not a price per se. We study this possibility in section 4.4. More fundamentally, mixed proofs are a better way to measure how much it costs to fully distribute a proof, as they are a proper generalization of the local proofs, which is not the case of global proofs. Second, our upper bounds use purely global proofs, and our lower bounds (except in section 4.4) consider mixed proofs, thus we get the strongest results on both sides.

Another point of view for this part of the dissertation would be to define the *uniformity of a language*. It would simply be the inverse of the price of locality. It happens that the price is more handy, and that is why we use this instead of the uniformity. Yet the topic is really uniformity: how much does the landscape changes if only uniform schemes are allowed? Which portion of the classic schemes is uniform?

Remark 4.1. *Note that we assume that the local proofs given to the nodes are of the same size, and thus the concatenation is exactly n times larger then the size of one local certificate. The interesting question of whether the average proof size could be asymptotically better, if proofs of different sizes were allowed, is outside of the scope of this chapter.*

4.2.3 High price of locality

In this section, we prove that it can be very costly to distribute the proof. This is easy and is a warm-up for the rest of the chapter. A scheme uses *uniform local proofs*, if the local proofs given to the nodes of the network are all equal. It is simple to change such proof system into a global proof: just take the uniform local proof and make it global. The verifier has the same behaviour and the scheme is correct. This implies the following theorem.

Theorem 4.2. *For languages where an optimal proof-labelling scheme uses uniform local proofs, the price of locality is $\theta(n)$.*

This theorem applies to the language Symmetry, the set of graphs that do not admit a non-trivial automorphism, which has an optimal scheme with $O(n^2)$ uniform local proofs ([67], and Section 1.2 in the introduction chapter). We now consider the language At-most-one-selected (Amos), that has been defined and used in [54]. In this problem, the nodes are given binary inputs, and the *yes*-instances are the ones such that at most one node has input 1. We prove that this language meets the hypothesis of the previous theorem.

Theorem 4.3. *The language* Amos *has an optimal proof-labelling scheme with uniform proofs of size* $O(\log n)$.

Proof. We describe the scheme. The prover strategy on *yes*-instances is the following. If there is exactly one selected node, the prover provides the ID of the node as uniform certificate, otherwise it provides an empty label. The verification algorithm is, for every node v: if v is selected and the certificate is not its ID, then reject, otherwise accept. It is easy to check that this scheme is correct. First, if no node is selected, all nodes accept, for all certificate. Second, if one nodes is selected, then the prover provides its ID as a certificate, and thus the selected node accepts, and all the other nodes too. Finally, if two or more nodes are selected, at most one of them has its ID written in the global certificate, because the IDs are distinct ; thus at least one node is rejecting.

In [67], the authors prove that verifying that exactly one node is selected (Leader Election) requires $\Omega(\log n)$ local certificate. The proof basically shows that without this amount of proof, an instance with two leaders would be accepted. This reasoning holds for Amos, and we can derive a $\Omega(\log n)$ lower bound for local certificates as well. □

Corollary 4.1. *The price of locality for* Amos *is in* $\Theta(n)$.

4.2.4 Intermediate price of locality

In this subsection, we show that the language Minimum Spanning Tree (MST) has price of locality $\Theta(\log n)$. It is an intermediate price, between n (the previous subsection), and constant (the next section). Remember that the language MST is the set of weighted graphs in which a subset of the edges are selected, and form a minimum spanning tree of the graph. The edge weights are assumed to be polynomial in n, and for simplicity we assume that the edge weights are distinct.

In [79], the authors show that there exist local proofs of size $O(\log^2 n)$ for Mst, and that this bound is tight. We show a simple global proof that has size $O(n \log n)$. As a mixed proof for the simpler language Spanning Tree requires $\Omega(n \log n)$ (see Section 4.3), this bound is also tight.

Theorem 4.4. *The global proof size for* Minimum Spanning Tree *is in* $O(n \log n)$.

Proof. We describe the scheme. On a *yes*-instance the prover provides a list of the selected edges with their weights. This global certificate has size $O(n \log n)$, because the edge weights and the identifiers can be written on $O(\log n)$ bits. Then every node first checks that the certificate is correct regardless of the graph. That is, every node checks that:

- The certificate is a well-formed edge list. Let L be this list.

- The list L describes an acyclic graph. That is that there is no set of nodes $w_1, w_2, ..., w_k$ such that $(w_1, w_2), (w_2, w_3), ..., (w_{k-1}, w_k)$, and (w_k, w_1) appear in the list.
- The list L describes a connected graph. That is for any pair of nodes present in the list, there exists a path in the list that connects them.

Then every node v of the graph checks locally that:

- The L is consistent with the selected edges that are adjacent to it.
- It has an adjacent selected edge.
- For every $e = (v, w)$ in the graph but not in the list, and every edge e' on the path from v to w in L, the weight of e' is smaller than the weight of e.

We now prove the correctness of the scheme. The first part of the verification insures that the set of edges described by L form an acyclic connected graph. The two first checks of the second part insure that it contains the selected edges and that it is spanning the graph. As it is a spanning tree, it must then be exactly the set of selected edges. Finally, remember that the so-called *cycle property* states that a spanning tree verifying the last item of the previous algorithm is a minimum spanning tree [26]. □

Corollary 4.2. *The price of locality for* Mst *is in* $\Theta(\log n)$*.*

4.3 Locality for free and reversing decision

In this section, we show that for some languages there exists local proofs of size $O(\log n)$ and that any mixed proof has size $O(n \log n)$. It follows that in this case, the price of locality is constant, that is the locality of the proofs comes for free.

The language we consider, called At least one selected (Alos), consists of all labelled graphs such that at least one node has a non-zero input label. We say that a node with a non-zero input label is *selected*. Proving that at least one node has some special property (being the root, having some input, being part of some special subgraph) is an important subroutine in many schemes.

On a more fundamental perspective, reversing decision basically deals with proving that some node is rejecting, which falls into the scope of the Alos. It has long been known that $O(\log n)$ local proof is sufficient for reversing decision, and the current section shows that not only this is optimal, but also one cannot gain by using global proofs.

Theorem 4.5. *A mixed proof for the language* Alos *requires* $\Omega(n \log n)$ *bits.*

The theorem is equivalent to state that the language requires either $\Omega(\log n)$ bits per local proof or $\Omega(n \log n)$ bits of global proofs. For a sketch of the proof, see Subsection 1.3.2 in the introduction.

Proof of Theorem 4.5. Consider a mixed scheme with local certificates of size $f(n)$ and a global certificate of size $g(n)$. Let r be the verification radius of the scheme.

Blocks. The lower bound instances are consistently oriented cycles of length at most $n = (b+1)(2r+1)$, for some integer b. Cycles are constructed from blocks of $2r+1$ nodes: the ith block is a path $B_i = (v_j, v_{j+1}, \ldots, v_{j+2r})$, where $j = i(2r+1) + 1$, oriented consistently from v_j to v_{j+2r}. Each node v_j is labeled with the unique identifier j.

Constructing instances from blocks. Let $\pi\colon [b] \to [b]$ be a permutation on the set of the first b blocks. Each permutation defines a cycle C_π where we take the blocks in the order given by π, and finally take the $(b+1)$th block. Each pair of consecutive blocks in π is connected by an edge, and B_{b+1} is connected to $B_{\pi^{-1}(1)}$.

Finally, the center node $v_{b(2r+1)+r+1}$ of B_{b+1} is labelled with a non-zero label, making the instance a *yes*-instance. All other nodes are labelled with the zero-label. Denote this family of permuted *yes*-instances by $\mathcal{C} = \{C_\pi\}_\pi$.

Labeled blocks. The prover assigns a local proof of $f(n)$ bits to each node. Thus, there are $2^{f(n)(2r+1)}$ different labeled versions of each block. We call these *labelled blocks*. Denote by $B_{i,\ell}$ the block B_i labelled according to ℓ. We call B_i the *type* of $B_{i,\ell}$

Consider two labelled blocks, $B_{i,\ell}$ and $B_{j,\ell'}$, in this order, linked by an edge. We say that labelled blocks are accepting from $B_{i,\ell}$ to $B_{j,\ell'}$ with global certificate L if, when we run the verifier on the nodes that are at distance at most r from an endpoint of the connecting edge, all these nodes accept. We denote this by $B_{i,\ell} \to_L B_{j,\ell'}$.

For each choice L of the global certificate, this edge relation defines a graph $G_{\mathcal{B},L}$ on the set of labelled blocks. A path in $G_{\mathcal{B},L}$ corresponds to a labelled path fragment in which all nodes at least r steps away from the path's endpoints accept. Finally, an accepting cycle is a cycle in $G_{\mathcal{B},L}$ such that all nodes accept.

Bounding the overlap of certificates. For each $C_\pi \in \mathcal{C}$, there must exist an accepting assignment of certificates to the nodes. Let L denote the global part of this accepting certificate. Such a C_π corresponds to a directed cycle in $G_{\mathcal{B},L}$. Note that in this cycle the last edge can be omitted as it would always link the last block to the first block. Then C_π corresponds to a directed path $P(C_\pi, L)$ of length b in $G_{\mathcal{B},L}$. Denote the set of labelled blocks on this path by $S(C_\pi, L)$.

Let $\mathcal{C}_L$ denote the set of instances such that there exists an accepting local certification given the global certificate L. Every *yes*-instance has an accepting certification, so there must exist L^* with

$$|\mathcal{C}_{L^*}| \geq |\mathcal{C}|/2^{g(n)}.$$

Now consider any two instances C_π and $C_{\pi'}$ in $\mathcal{C}_{L^*}$. We drop the specification of the global certificate from the notation. Assume that C_π and $C_{\pi'}$ use the same set of blocks, that is $S(C_\pi, L^*) = S(C_{\pi'}, L^*)$. Also assume without loss of generality, that π is the identity permutation. Now in $P(C_{\pi'})$ there must exist a *back edge* with respect to π, that is, an edge between labelled blocks B and B', of types $B_{\pi'^{-1}(i)}$ and $B_{\pi'^{-1}(i+1)}$ respectively, such that $\pi'^{-1}(i) > \pi'^{-1}(i+1)$. This is because we assumed that the instances consist of the same blocks, but are different. Therefore at some point an edge of $C_{\pi'}$ must go backwards in the order of π. We also have that $B, B' \neq B_{b+1}$ as if there is no back edge before reaching B_{b+1}, we must have $C_\pi = C_{\pi'}$.

This implies that there is an accepting cycle formed by taking first the path from B to B' along $P(C_\pi)$ and then an edge from B' to B. This cycle does not contain a selected node. It follows that there is a *no*-instance of size at least $2(2r+1)$ and a certification that causes the verifier to accept the instance. Therefore we have the following lemma.

Lemma 4.1. *For all pairs of instances C_π, $C_{\pi'}$ with the same accepting global certificate L, we have that $S(C_\pi, L) \neq S(C_{\pi'}, L)$.*

Remark 4.2. *Note that the contradicting instances can be of size $2(2r+1)$ but the identifiers can be of size n and the certificates of size $f(n)$. Therefore the lower bound only holds for uniform verifiers that do not get any guarantees except that 1) the identifiers come from the set $[n+c]$, for some constant c, and 2) the certificates are of size at least $f(n)$.*

Alternatively it is possible to consider ALOS *on possibly disconnected instances so that every connected component must have at least one node selected. In this case the proof will fool even a non-uniform prover.*

Counting argument. By Lemma 4.1, each pair of permutations π, π' in $\mathcal{C}_{L^*}$ must induce a different set of labelled blocks that form the accepting certifications of instances C_π and $C_{\pi'}$. The number of different permutations in $\mathcal{C}_{L^*}$ is at least $b!/2^{g(n)}$. On the other hand, the number of different sets of labelled blocks, selecting a block of each type, is $2^{f(n)(2r+1)b}$. As shown in Lemma 4.1, to have a legal certification, we must have that $2^{f(n)(2r+1)b+g(n)} \geq b!$.

Using Stirling's approximation we get that $f(n)(2r+1)b + g(n) \geq b\log_2 b - (\log_2 e)b + O(\ln b)$. Since $b = \Theta(n)$ and $r = O(1)$, this implies that either $f(n) = \Omega(\log n)$ or $g(n) = \Omega(n \log n)$. Thus the mixed proof has size $\Omega(n \log n)$ □

Corollary 4.3. *Inverting decision with a single additional level of nondeterminism requires local certificates of size $\Omega(\log n)$ or global certificates of size $\Omega(n \log n)$.*

Proof. Consider the language NONE SELECTED, that is, the language of labelled graphs such that all nodes have the zero label. This language is locally decidable without nondeterminism, that is, NONE SELECTED $\in$ LD [51] or Λ_0 in the notation of Section 6.5. The language ALOS is its complement. Finally, by Theorem 4.5, deciding ALOS, that is, reversing the decision of NONE SELECTED requires local certificates with $\Omega(\log n)$ bits or global certificates with $\Omega(n \log n)$ bits. □

The proof can be adapted to several other problems, namely leader election, spanning tree and the set of odd-cycles, giving a lower bound for mixed proof systems.

Corollary 4.4. *Any mixed proof system for* LEADER ELECTION *requires local certificates of size $\Omega(\log n)$ or global certificates of size $\Omega(n \log n)$.*

Proof of Corollary 4.4. Consider the proof of Theorem 4.5. The family $\mathcal{C}$ of *yes*-instances for ALOS is also a family of *yes*-instances for LEADER ELECTION. Since

Leader election $\subsetneq$ Alos, the proof of Theorem 4.5 produces *no*-instances of Leader election that the verifier accepts. □

Corollary 4.5. *Any mixed proof system for* spanning tree *requires local certificates of size* $\Omega(\log n)$ *or global certificates of size* $\Omega(n \log n)$*.*

Proof sketch. Consider two types of instances: the cycles where all the edges are selected, and the the cycles where all edges but one are selected. The first instances are not in the language, the second are. We can rephrase this restricted problem as: there is at least one non-selected edge. Then the same type of proof works. □

Corollary 4.6. *Any mixed proof system for* odd-cycle *requires local certificates of size* $\Omega(\log n)$ *or global certificates of size* $\Omega(n \log n)$*.*

Proof sketch. The proof of Corollary 4.6 consists in a refinement of the proof for Alos. We can build on an odd number of blocks, each block being of odd length itself. Then we can give a colour to each block so that half of the blocks are black and half are white. Finally we can force the paths to alternate between white and black blocks. The cycles obtained will then be of even length, and thus be *no*-instances. The number of possible paths is reduced, but only by term of the form 2^b, which is negligible compared with the $b!$ term. The calculation then still gives the $\Omega(n \log n)$ lower bound. □

A consequence of these corollaries is that all the $\Omega(\log n)$ lower bounds obtained in [67] for local certificates can be lifted to $\Omega(n \log n)$ mixed proofs with our technique. However for the problem Amos we studied in the previous section, our technique does not work, which is consistent with the fact that an $\Omega(n \log n)$ lower bound would contradict the $O(\log n)$ upper bound we show. As already said, the technique of [67] works for Amos, and provides the $\Omega(\log n)$ bound for local proofs. The reason our technique fails is because we show that if the certificates are too short then one can shorten the cycles that are *yes*-instances, which is not useful for Amos, as a 'subinstance' of this problem is still in the language: one can only remove selected nodes. The authors of [67] show that one can glue different *yes*-instances together and get a configuration that is still accepted by the nodes, and for Amos this basically means one can glue different instance with one node selected, and then get an instance with more than one node selected, and this instance is still accepted, which rises a contradiction. Note that because of this duality, the proof technique of [67] could not help to get lower bound for Alos, even when looking only at local proofs.

It is also worth noting that the intersection of the languages Amos and Alos, is Leader Election. For this language, it has long been known that a PLS has size $\Theta(\log n)$, and is formed by the certificates of a spanning forest, along with the ID of the leader given to all the nodes. The results of the current and previous sections show that this decomposition is somehow mandatory: one basically needs a global part of size $\Theta(\log n)$, and a local part of size $\Theta(\log n)$.

4.4 Beyond free locality

The language Bipartite is the set of bipartite graphs. Local proofs of constant size exist for this language: the prover can just describe a 2-colouring of the graph by giving a bit to each node, and every node can check that its neighbours are given a colour different from its own. We conjecture that for this language, even when restricting the topology to cycles, optimal purely global proofs are larger than the sum of the optimal local proof sizes. More precisely this sum is $\Theta(n)$, and we conjecture that purely global proofs take $\Theta(n \log n)$ bits.

Conjecture 4.1. *For* Bipartite*, purely global proofs have size* $\Theta(n \log n)$*.*

We are not able to prove the lower bound of the conjecture, but we can prove weaker inequalities. For this problem, the range of the identifiers is important, and that is why we consider the maximum identifier to be a parameter M, that we do not bound by a polynomial any more.

Theorem 4.6. *For* Bipartite*, there exist two constants α and β such that, for identifiers bounded by M:*

$$\alpha \max\{n, \log\log M\} \leq s_g(n) \leq \beta \min\{M, n \log M\}.$$

Note that if $M = n$ then we get a tight $\Theta(n)$ bound. The $\Omega(n)$ lower bound holds for any ID range, but the $\log\log M$ bound shows that this cannot be tight for every ID range: we can get arbitrarily large lower bound if we allow arbitrarily large identifiers.

Proof idea. The $O(M)$ upper bound corresponds to the following scheme. The prover provides a global proof that is a table, indicating for each ID, whether the node of with this ID is black, white or not in the graph. The verifier can check locally the correctness of this colouring.

Now for the lower bound proof, we consider cycles. The key step is to prove that, although a priori a proof of bipartiteness is not required to explicitly give a 2-colouring to the nodes, it actually always does. That is, for every scheme, the nodes can extract a colour from the proof, and this colour provides a proper 2-colouring of the *yes*-instances. To prove this we use the block machinery of the proof of Theorem 4.5, to define a special directed graph. We can prove that it is bipartite, by studying its cycles and strongly connected components. Then the nodes can use this graph to infer their colours from the proof. Once we have this property, the $\Omega(n)$ lower bound basically follows from the fact that encoding the colour of the n nodes of the cycle takes at least $\Omega(n)$ bits. The $\Omega(\log\log M)$ lower bounds comes from the fact that if there are many possible IDs, and few certificates, then there must exist two identifiers (or actually blocks of identifiers) such that, for every possible global certificate, they are assigned the same colour. And this is not possible as there are even length cycles where they are adjacent. □

Proof. We start with the upper bounds. The $O(n \log M)$ upper bound comes from the certificate made by concatenating the couples (ID, local proof) for every node, as in Theorem 4.1. For the $O(M)$ upper bound, the prover strategy is to provide a vector with M cells, where cell i will contain a bit indicating the colour of the node with ID i. In both cases the nodes will get their own colours and the colours of their neighbours from the certificate, and they can check locally the consistency of the colouring.

We now prove the lower bounds for the restricted case of cycles. Note that bipartiteness on cycles boils down to distinguishing between odd and even length cycles. A priori, in a scheme for this language, the prover is not forced to explicitly provide a colouring to the nodes. We show that a proof always implies a colouring. More precisely, a node can always extract from the proof its colour and the colours of its neighbours, and then check the consistency of the colouring. As in Section 4.3, we will use blocks of nodes to build a large a number of instances. The blocks are paths of $2r+1$ nodes. The i-th block, noted b_i has consecutive IDs from $i(2r+1)+1$ up to $(i+1)(2r+1)$. Every block is oriented in the direction of increasing IDs. A *block-based cycle* is a cycle made by concatenating blocks, with a consistent orientation.

Lemma 4.2. *For every global proof c, there exists a colouring function $f_c : [M] \mapsto \{0,1\}$, such that for every block-based cycle H that is accepting with certificate c, f_c defines a proper colouring of H.*

Proof (Lemma 4.2). First, note that as the blocks have odd length, a block-based cycle has even length if and only if it is composed of an even number of blocks. Then, for block-based cycles, replacing virtually each block by a vertex, and trying to 2-colour the resulting cycle is equivalent to 2-colour the nodes of the original instance.

Fix a certificate c. Consider the directed graph G_c, whose nodes are the blocks $(b_i)_i$. There is an oriented edge (b_i, b_j) if and only if there exists a block-based cycle for which c is an accepting certificate, and where the block b_i is followed by the block b_j.

Claim 4.1. *The graph G_c contains no directed odd cycle.*

Suppose the graph G_c contains a directed odd cycle. Consider the corresponding block-based cycle C. Because it has odd length, it is a *no*-instance. Consider a node v of this instance that is rejecting with certificate c. Without loss of generality, assume it is in the first half of its block b_i (that is, its ID is between $i(2r+1)+1$ and $i(2r+1)+r+1$). Let b_h be the block preceding b_i in C. The node v can only see (parts of) of b_h and b_i, because its radius is r. As (b_h, b_i) belongs to G_c, there exists a *yes*-instance C', in which every node accepts with proof c, and in which b_i follows b_h. This is a contradiction, because with certificate c, v is accepting in C', and rejecting in C, although it has the exact same view in both instances. Thus the graph G_c contains no directed odd cycle.

Claim 4.2. *Every connected component of the graph G_c is strongly connected.*

Consider the following way of building G_c: take an arbitrary ordering of the cycles that accept with c, and add them (i.e. add their edges) to G_c, one by one. We show the strong connectivity of the connected components by induction. The property holds for the empty graph. Suppose every connected component is strongly connected until some step, and that we add a new cycle. As a directed cycle is strongly connected, merging it with one or several strongly connected components, keeps the strong connectivity.

It is known that a strongly connected digraph with no odd length directed cycles can be 2-coloured (see e.g. Theorem 1.8.1 in [13]). Thus, from Claim 4.1 and Claim 4.2, we get that G_c has a 2-colouring. This 2-colouring induces a 2-colouring on all the block-based cycles accepting with c, thus it defines the function f_c of the lemma. □

Now fix a size n and consider the following table. The columns are indexed by the blocks, thus there are $M/(2r+1)$ of them. The rows are indexed by all the possible certificates, that is all the strings on $s_g(n)$ bits. The cell that corresponds to block b and certificate c contains the colour given by f_c to the center node of b. We will now give two simple properties of this table that will imply the two lower bounds.

Let a *balanced binary vector* be a vector of bits with the same number of zeros and ones. Let the *complement* of a binary vector be the same binary vector where ones and zeros have been complemented.

Lemma 4.3. *For every balanced binary vector p of length n, there exists a row of the table such that the vector made by the n first cells is equal to either p or its complement.*

Proof (Lemma 4.3). Consider a balanced binary vector p. Consider a cycle H made by concatenating the n first blocks, in an ordering such that colouring block i with the i^{th} bit of p, defines a proper colouring of the cycle. Note that, as p is balanced, such a cycle must exist. This cycle H has even length, thus it belongs to the language and there exists an accepting certificate c. The first n cells of the row of c must describe a proper colouring of H, and there are only two such colourings: p and its complement. □

As for every balanced vector of length n there exists a row that it matches (or its complement matches) on the n first cells, and that a row can only correspond to one such vector (up to complement), the table must have at least $2^n/2$ rows. This means that there are at least $2^n/2$ different certificates, thus the certificate size is lower bounded by n, up to multiplicative constants.

Lemma 4.4. *Two columns of the table cannot be equal.*

Proof (Lemma 4.4). Suppose columns i and j are equal. Consider an even-length block-based cycle C, where the blocks i is linked to the block j. Such a cycle always exists. For every certificate c, the same colour is given to both blocks i and j in f_c, because the columns are equal. Thus no certificate provides a proper colouring of C, which is a contradiction because C belongs to the language. □

As there are M different columns, there is at least order of $\log(M)$ certificates. Then the length of a certificate is in $\Omega(\log\log(M))$. This finishes the proof of Theorem 4.6. □

Chapter 5

Redundancy

This chapter deals with proof-labelling scheme having non-constant view radius. This is first motivated by the quest for more compact schemes. Second it is a tool to study redundancy, that is local correlation between certificates. It is based on the submitted paper [42], which is joint work with Pierrre Fraigniaud, Juho Hirvonen, Ami Paz and Mor Perry. It corresponds to Subsection 1.3.3 in the introduction of the thesis.

5.1 Introduction

Motivation and specific related work. Several attempts have been made to make proof-labelling schemes more efficient. For instance, it was shown in [14] that randomization helps a lot in term of *communication* costs, typically by hashing the certificates, but this might actually come at the price of dramatically increasing the certificate size. Sophisticated deterministic and efficient solutions have also been provided for reducing the size of the certificates, but they are targeting specific structures only, such as MST [83]. Another direction for reducing the size of the certificates consists of relaxing the decision mechanism, by allowing each node to output more than just a single bit (accept of reject) [4, 5]. For instance, certifying cycle-freeness simply requires certificates of $O(1)$ bits with just 2-bit output, while certifying cycle-freeness requires certificates of $\Omega(\log n)$ bits with 1-bit output [82]. However, this relaxation assumes the existence of a centralized entity gathering the outputs from the nodes, and there are still network predicates that require certificates of $\widetilde{\Omega}(n^2)$ bits even under this relaxation. Another notable approach is using approximation [24], which reduces, e.g., the certificate size for certifying the diameter of the graph from $\Omega(n)$ down to $O(\log n)$, but at the cost of only determining if the given value is up to two times the real diameter.

In this chapter, we aim at designing deterministic and generic ways for reducing the certificate size of proof-labelling schemes. This is achieved by following the guidelines of [90], that is, trading time for space by exploiting the inherent redundancy in distributed proofs. In [90], the author allow the verification procedure to take t rounds (t can be a function of n) in order to reduce the certificate size. For instance, it was shown that, for the so-called universal scheme, the size of the certificates can be reduced from $\Theta(n^2)$ to $\Theta(n^2/t)$ bits when increasing verification time from 1 round to

t rounds. The results in [83] were another source of inspiration, as it is shown that, by allowing $O(\log^2 n)$ rounds of communication, one can verify MST using certificates of $O(\log n)$ bits. In fact, [83] even describe an entire (non-silent) self-stabilizing algorithm for MST construction based on this mechanism for verifying MST.

Background on communication complexity. In this chapter, we use results from the well studied field of communication complexity [86, 104]. In the set-disjointness (DISJ) problem on k bits, each of two players, Alice and Bob, is given a k-bit string, denoted S_A and S_B, respectively, as input. They aim at deciding whether $S_A \cap S_B = \emptyset$, i.e., whether there does not exist $i \in \{1, \dots, k\}$ such that $S_A[i] = S_B[i] = 1$. The communication complexity of a given protocol solving DISJ is the number of bits Alice and Bob must communicate, in the worst case, when using this protocol. The communication complexity of DISJ is the minimum communication complexity of a protocol solving it.

In *nondeterministic communication complexity*, each of the players receives, in addition to its input, a binary string as a non reliable hint (these hints may depend on both input strings).[1] For example, a good hint for $\overline{\text{DISJ}}$, i.e., deciding whether there exists $i \in \{1, \dots, k\}$ such that $S_A[i] = S_B[i] = 1$, is the index i itself. Indeed, if one of the two players receives i as a hint, he or she can send it to the other player, and they both check that $S_A[i] = 1$ and $S_B[i] = 1$.

The communication complexity of a nondeterministic protocol for DISJ is the sum of the number of bits the players exchange and the number of nondeterministic proof bits provided to the players in the worst case. The nondeterministic communication complexity of DISJ is the minimum, among all nondeterministic protocols for DISJ, of the communication complexity of that protocol. The nondeterministic communication complexity of DISJ is known to be $\Omega(k)$, as a consequence of, e.g., Example 1.23 and Definition 2.3 in [86].

Specific notations and model definition. In this chapter, following the original text [42], we will consider predicates, that is properties that the configuration can satisfy or not satisfy. As said in Chapter 2, using predicates instead of languages is just a change of terminology, the underlying notion is the same.

The *radius* of a proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ is defined as the maximum number of rounds of the verifier $\mathbf{v}$ in the LOCAL model [95], over all identity-assignments to all the instances in $\mathcal{G}$, and all arbitrary certificates. It is denoted by $\mathsf{radius}(\mathbf{p}, \mathbf{v})$. We will compare proof-labelling schemes of radius 1, denoted 1-PLS, with proof-labelling scheme of radius $t \geq 1$ is abbreviated into t-PLS. The value t will be given to the nodes by an oracle at the beginning of the decision scheme (remember that the nodes do not have access to n in general).

The minimum certificate size of a t-PLS for the predicate $\mathcal{P}$ on n-node labelled graphs is denoted by $\mathsf{size\text{-}pls}(\mathcal{P}, t)$, that is,

$$\mathsf{size\text{-}pls}(\mathcal{P}, t) = \min_{\mathsf{radius}(\mathbf{p},\mathbf{v}) \leq t} \mathsf{size}(\mathbf{p}, \mathbf{v}).$$

[1] Another definition of non-determinism for communication complexity, with gloabl proofs will be used in Chapter 6.

With this notation, $\mathsf{size\text{-}pls}(\mathcal{P}) = \mathsf{size\text{-}pls}(\mathcal{P}, 1)$.

Definition 5.1. *Let $\mathcal{I} \subseteq \mathbb{N}^+$, and let $f : \mathcal{I} \to \mathbb{N}^+$. Let $\mathcal{P}$ be a boolean predicate on labelled graphs. A set $(\mathbf{p}_t, \mathbf{v}_t)_{t\in\mathcal{I}}$ of proof-labelling schemes for $\mathcal{P}$, with respective radius $t \geq 1$,* scales *with* scaling factor *f on $\mathcal{I}$ if $\mathsf{size}(\mathbf{p}_t, \mathbf{v}_t) = O\big(\frac{1}{f(t)} \cdot \mathsf{size\text{-}pls}(\mathcal{P})\big)$ bits for every $t \in \mathcal{I}$. Also, $(\mathbf{p}_t, \mathbf{v}_t)_{t\in\mathcal{I}}$* weakly scales *with scaling factor f on $\mathcal{I}$ if $\mathsf{size}(\mathbf{p}_t, \mathbf{v}_t) = \widetilde{O}\big(\frac{1}{f(t)} \cdot \mathsf{size\text{-}pls}(\mathcal{P})\big)$ bits for every $t \in \mathcal{I}$.*

In the following, somewhat abusing terminology, we shall say that a proof-labelling scheme (weakly) scales while, formally, it should be a set of proof-labelling schemes that scales. A definition of the *redundancy* of a language can be defined as the scaling factor for the optimal t-proof-labelling scheme. Indeed if this factor is high, then it means that one can gain a lot by having a large radius, and in turn this means that the certificates of close nodes are very correlated.

Remark. At first glance, it may seem that no proof-labelling schemes can scale more than linearly, i.e., one may be tempted to claim that for every predicate $\mathcal{P}$ we have $\mathsf{size\text{-}pls}(\mathcal{P}, t) = \Omega\left(\frac{1}{t} \cdot \mathsf{size\text{-}pls}(\mathcal{P})\right)$. The rational for such a claim is that, given a proof-labelling scheme $(\mathbf{p}_t, \mathbf{v}_t)$ for $\mathcal{P}$, with radius t and $\mathsf{size\text{-}pls}(\mathcal{P}, t)$, one can construct a proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for $\mathcal{P}$ with radius 1 as follows: the certificate of every node v is the collection of certificates assigned by $\mathbf{p}_t$ to the nodes in the ball of radius t centered at v; the verifier $\mathbf{v}$ then simulates the execution of $\mathbf{v}_t$ on these certificates. In paths or cycles, the certificates resulting from this construction are of size $O(t \cdot \mathsf{size\text{-}pls}(\mathcal{P}, t))$, from which it follows that no proof-labelling scheme can scale more than linearly. There are several flaws in this reasoning, which make it actually erroneous. First, it might be the case that degree-2 graphs are not the worst case graphs for the predicate $\mathcal{P}$; that is, the fact that $(\mathbf{p}, \mathbf{v})$ induces certificates of size $O(t)$ times the certificate size of $(\mathbf{p}_t, \mathbf{v}_t)$ in such graphs may be uncorrelated to the size of the certificates of these proof-labelling schemes in worst case instances. Second, in t rounds of verification every node learns not only the certificates of its t-neighbourhood, but also its structure, which may contain valuable information for the verification; this idea stands out when the lower bounds for $\mathsf{size\text{-}pls}(\mathcal{P})$ are established using labelled graphs of constant diameter, in which case there is no room for studying how proof-labelling schemes can scale. The take away message is that establishing lower bounds of the type $\mathsf{size\text{-}pls}(\mathcal{P}, t) = \Omega(\frac{1}{t} \cdot \mathsf{size\text{-}pls}(\mathcal{P}))$ for t within some non-trivial interval requires specific proofs, which often depend on the given predicate $\mathcal{P}$.

5.2 All proof-labelling schemes scale linearly in trees

This section is entirely dedicated to the proof of one of our main results, stating that *every* predicate on labeled trees has a proof that scales linearly. Further in the section, we also show how to extend this result to cycles and to grids, and, more generally, to multi-dimensional grids and toruses.

5.2.1 Linear scaling for trees

Theorem 5.1. *Let $\mathcal{P}$ be a predicate on labelled trees, and let us assume that there exists a (distance-1) proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for $\mathcal{P}$, with $\mathsf{size}(\mathbf{p}, \mathbf{v}) = k$. Then there exists a proof-labelling scheme for $\mathcal{P}$ that scales linearly, that is,* $\mathsf{size\text{-}pls}(\mathcal{P}, t) = O\left(\frac{k}{t}\right)$.

We give a more detailed sketch of the proof than in Subsection 5 of the introduction.

Proof sketch. The proof is based on a decomposition of trees that can be checked locally. It consists in marking some nodes of the tree, so that the unmarked nodes form clusters, that have diameter roughly t, and such that two nodes from different clusters are separated by at least two marked nodes. We illustrate this marking on the simpler example of a path with nodes $(v_i)_i$ in the natural order. On such a path we mark the node v_i, if i is equal to 0 or 1 modulo t. Then we have p paths of unmarked nodes, of length $t-2$, and between every two consecutive paths, there are two marked nodes. The verifier will check locally this decomposition.

Consider the following t-PLS. The prover first decides what are the certificates used in the 1-PLS, and then give their certificates only to marked nodes. For each cluster of unmarked nodes, a marked node close to it will be chosen as its leader. Because the cluster has small diameter, the leader can see it entirely within its t-view, and also see all the marked nodes that are at distance 1 and 2 from it. The verifier on a leader node will try every possible certificate assignment to the unmarked nodes of the corresponding cluster and run the 1-round verifier on them and on their neighbours. If there exists a certificate assignment that is accepted by all these nodes, then, on this leader node the verifier accepts. On non-leader nodes, the verifier always accept. Because the clusters are separated by double layers of marked nodes, the simulations performed by the leaders are independent. Therefore if all leaders accept, then we can put all the accepting certificate assignments together and craft an accepting assignment for the 1-PLS on the whole graph. Conversely if such an assignment exists, then the leaders will all accept.

This scheme has certificates of size $O\left(\frac{1}{t} \cdot \mathsf{size}(\mathbf{p}, \mathbf{v})\right)$ on average because the decomposition ensures that there are not too many marked nodes. We can use a variation of the spreading technique of [90] to have all the certificates of this size. □

The rest of this subsection is dedicated to the detailed proof of Theorem 5.1. So, let $\mathcal{P}$ be a predicate on labelled trees, and let $(\mathbf{p}, \mathbf{v})$ be a proof-labelling scheme for $\mathcal{P}$ with $\mathsf{size}(\mathbf{p}, \mathbf{v}) = k$. First, note that we can restrict attention to trees with diameter $> t$. Indeed, predicates on labelled trees with diameter $\leq t$ are easy to verify since every node can gather the input of the entire tree in t rounds. More precisely, if we have a scheme that works for trees with diameter $> t$, then we can trivially design a scheme that applies to all trees, by adding a single bit to the certificates, indicating whether the tree is of diameter at most t or not.

The setting of the certificates in our scaling scheme is based on a specific decomposition of given tree T.

Tree decomposition. Let T be a tree of diameter $> t$, and let

$$h = \lfloor t/2 \rfloor.$$

For assigning the certificates, the tree T is rooted at some node r. A node u such that

$$\textsc{dist}_T(r,u) \equiv 0 \pmod{h},$$

and u possesses a subtree of depth at least $h-1$ is called a *border* node. Similarly, a node u such that

$$\textsc{dist}_T(r,u) \equiv -1 \pmod{h},$$

and u possesses a subtree of depth at least $h-1$ is called a *extra-border* node. A node that is a border or an extra-border node is called a *special* node. All other nodes are *standard* nodes. For every border node v, we define the *domain* of v as the set of nodes in the subtree rooted at v but not in subtrees rooted at border nodes that are descendants of v.

Lemma 5.1.

1. *The domains form a partition of the nodes in the tree T.*
2. *Every domain forms a tree rooted at a border node, with depth in the range $[h-1, 2h-1]$.*
3. *Two adjacent nodes of T are in different domains if and only if they are both special.*

Proof. Let us first prove the first item. On the one hand, every node belongs to a domain. This is because every node has at least one border ancestor, since the root is a border node. Indeed, the root r has depth 0, and the diameter of T is at least $t+1$, which implies that r necessarily possesses a subtree of depth at least $h-1$. On the other hand, every node u belongs to a unique domain. This is simply because the closest border ancestor of u is uniquely defined.

To establish second item, we consider the domain of a border node u. Note that, for any node v in the domain of u, v is a descendent of u in T, and all the nodes in the shortest path between u and v are also in the domain of u. Thus the domain of u is indeed a tree rooted at u. The depth of a domain is at least $h-1$. Indeed, if the subtree rooted at u has depth $< h-1$, then, by definition, u is not special. It remains to show that the depth of a domain is at most $2h-1$. Let us assume for the purpose of contradiction that the depth d of the domain of u satisfies $d > 2h-1$. Then there exists a path of length at least $2h$ starting at u, and going downward the tree for reaching a leaf v of this domain. Then let us consider the node u' of that path which is at distance h from u. Node u' is not a border node, since otherwise v would not be in domain of u but in the domain of u'. However, node u' is a border node as its depth is 0 modulo k, and it has a subtree of depth at least $d-h > h-1$. This contradiction completes the proof of item 2.

Finally, for establishing the third item, let us consider an edge $\{u,v\}$ in the tree T, and let us assume, w.l.o.g., that u is the parent of v in T. By construction, there can be three cases only. If none of the two nodes u and v is a border node, then both belong to the same domain, as they have the same closest border ancestor. If u is a border node, and v is a standard node, then v is in the domain of u. Finally, if v is a border node, then necessarily u is an extra-border node, in which case u and v do not belong to the same domain since v is in its own domain, while u cannot belong to the domain of v. Therefore, Item 3 holds in these three cases. □

Verifying the decomposition. The certificates of the distance-t proof-labelling scheme will contains a 2-bit field indicating to each node whether it is a root, border, extra-border, or standard node. Let us show that this part of the certificate can be verified in t rounds. The prover orients the edges of the tree towards the root r. It is well-known that such an orientation can be given to the edges of a tree by assigning to each node its distance to the root, modulo 3. These distances can obviously be checked locally, in just one round. So, in the remaining of the proof, we assume that the nodes are given this orientation upward the tree. The following lemma proves that the decomposition into border, extra-border, and standard nodes can be checked in t rounds.

Lemma 5.2. *Given a set of nodes marked as border, extra-border, or standard in an oriented tree, there is a verification protocol that checks whether that marking corresponds to a tree decomposition such as the one described above, in $2h < t$ rounds.*

Proof. The checking procedure proceeds as follows. The root r checks that it is a border node. Every border node checks that its subtree truncated at depth $2h$ fits with the decomposition. That is, it checks that: (1) no nodes in its subtree are border nodes except nodes at depth h and $2h$, (2) no nodes in its subtree are extra-border nodes except nodes at depth $h-1$ and $2h-1$, and (3) the nodes in its subtree that are special at depth $h-1$ and h do have a subtree of depth at least $h-1$. By construction, this procedure accepts any marking which is correct with respect to the decomposition rule.

Conversely, let us suppose that the algorithm accepts a marking of the nodes. We prove that this marking is necessarily correct. We proceed by induction on the depth of the nodes. At the root, the verifier checks that r is special as a border node, and it checks that the domain of r is correctly marked. In particular, it checks that the nodes of depth h that are not in its domain are properly marked as border nodes. So, the base of the induction holds. Now, assume that, for $\alpha \geq 0$, all the domains whose border nodes stand at depth at most αh are properly marked. The fact that the border nodes at depth αh accept implies that all the nodes at depth $(\alpha+1)h$ that are not in the domain of the border nodes at depth αh are properly marked. These nodes verify their own domains, as well as all the domains down to depths $(\alpha+1)h$, are all correct. Since none of these nodes reject, it follows that all the domains whose border nodes stand at depth at most $(\alpha+1)h$ are properly marked. This completes the induction step, and hence the proof of the lemma. □

We are now ready to describe the distance-t proof-labelling scheme. From the previous discussions, we can assume that the nodes are correctly marked as root, border, extra-border, and standard, with a consistent orientation of the edges towards the root.

The distance-t proof-labelling scheme. We are considering the arbitrarily given predicate $\mathcal{P}$ on labelled trees, with its proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ using certificates of size k bits. Before reducing the size of the certificates to $O(k/t)$ by communicating at distance t, we describe a proof-labelling scheme at distance t which still uses large certificates, of size $O(k)$, but stored at a few nodes only, with all other nodes storing no certificates.

Lemma 5.3. *There exists a distance t proof-labelling scheme for $\mathcal{P}$, in which the prover assigns certificates to special nodes only, and these certificates have size $O(k)$.*

Proof. On legally labelled trees, the prover provides every special node (i.e., every border or extra-border node) with the same certificate as the one provided by $\mathbf{p}$. All other nodes are provided with no certificates.

On arbitrary labelled trees, the verifier is active at border nodes only, and all non-border nodes systematically accept (in zero round). At a border node v, the verifier first gathers all information at distance $2h$. This includes all the labels of the nodes in its domain, and of the nodes that are neighbours to a node in its domain. Then v checks whether there exists an assignment of k-bit certificates to the standard nodes in its domain that results in $\mathbf{v}$ accepting at every node in its domain. If this is the case, then v accepts, else it rejects. There is a subtle point worth to be mentioned here. The value of k may actually depend on n, which is not necessarily known to the nodes. Nevertheless, this can be easily fixed as follows. The t-PLS prover is required to provide all nodes with certificates of the same size (the fact that all certificates have identical size can trivially be checked in just one round). T hen k is simply inferred from the certificate size in the t-PLS, by multiplying this size by t, whose value is, as specified in Section 5.1, given as input to each node.

Note that, as every border node v has a complete view of its whole domain, and of the nodes at distance 1 from its domain, v considers all the nodes that are used by $\mathbf{v}$ executed at the nodes of its domain. Also note that the execution of $\mathbf{v}$ at nodes in the domain of v concerns only nodes that are either in the domain of v, or are special. This follows from the third item in Lemma 5.1. Thus no two border nodes will simulate the assignment of certificates to the same node.

We now prove that, in an oriented marked tree, this scheme is correct.

– Assume first that the labelled tree satisfies the predicate $\mathcal{P}$. Giving to the special nodes the certificates as assigned by $\mathbf{p}$, all the border nodes will be able to find a proper assignment of the certificates for the standard nodes in their domain so that $\mathbf{v}$ accepts at all these nodes, since, as the labelled tree satisfies the predicate $\mathcal{P}$, there must exists at least one. This leads every node to accept.

– Suppose now that every border node accepts. It follows that, for every border node, there is an assignment of certificates to the nodes in its domain such that $\mathbf{v}$ accepts these certificates at every node. The union of these partial assignments of certificates defines a certificate assignment to the whole tree that is well-defined according to the first item of Lemma 5.1. A every node, $\mathbf{v}$ accepts since it has the same view as in the simulation performed by the border nodes in their respective domains. Thus $\mathbf{v}$ accepts at every node of T, and therefore it follows that the labelled tree satisfies $\mathcal{P}$. □

Lemma 5.3 basically states that there is a distance-t proof-labelling scheme in which the prover can give certificates to special nodes only. We now show how to spread out the certificates of the border and extra-border nodes to obtain smaller certificates. The following lemma is the main tool for doing so. As this lemma is also used further in the chapter, we provide a generalized version of its statement, and we later show how to adapt it to the setting of the current proof.

We say that a local algorithm $\mathcal{A}$ *recovers* an assignment of certificates provided by some prover $\mathbf{q}$ from an assignment of certificates provided by another prover $\mathbf{q}'$ if, given the certificates assigned by $\mathbf{q}'$ as input to the nodes, $\mathcal{A}$ allows every node to output its certificate such as assigned by $\mathbf{q}$. We define a *special* prover as a prover which assigns certificates only to the special nodes, all other nodes being given empty certificates.

Lemma 5.4. *There exists a local algorithm $\mathcal{A}$ satisfying the following. For every $s \geq 1$, for every oriented marked tree T of depth at least s, and for every assignment of b-bit certificates provided by some special prover $\mathbf{q}$ to the nodes of T, there exists assignment of $O(b/s)$-bit certificates provided by a prover $\mathbf{q}'$ to the nodes of T such that $\mathcal{A}$ recovers $\mathbf{q}$ from $\mathbf{q}'$ is s rounds.*

Proof. We first describe the prover $\mathbf{q}'$. For each border node v, let us partition the certificate $\mathbf{q}(v)$ assigned to node v by the special prover $\mathbf{q}$ into s parts of size at most $\lceil b/s \rceil$. Then one picks an arbitrary path starting from v, of length $s-1$, going downward the tree. Note that such a path exists since v is a border node. For every $i \in \{0, \ldots, s-1\}$, the ith part of the certificate $\mathbf{q}(v)$ is assigned to the ith node of that path as its certificate in $\mathbf{q}'$. As such, every node is given at most one part of the initial certificates, as the paths starting at each of the border nodes are non intersecting. To recover the original certificates, for every border node v, the algorithm $\mathcal{A}$ simply inspects the tree at distance $s-1$ downward, for gathering all the parts of the initial certificate $\mathbf{q}(v)$ of v. Then v concatenates these parts, and v outputs the resulting certificate. All other nodes output a certificate formed by the empty string. □

We have now all the ingredients to prove Theorem 5.1.

Proof of Theorem 5.1. In the distance-t proof-labelling scheme, the prover chooses a root and an orientation of the tree T, and provides every node with a counter modulo 3 in its certificate allowing the nodes to check the consistency of the orientation. Then the prover constructs a tree decomposition of the rooted tree, and provides every node with its type (root, border, extra-border, or standard) in its certificates. Applying Lemmas 5.3 and 5.4, the prover spreads the certificates assigned to the special nodes by $\mathbf{p}$. Every node will get at most two parts, because only the paths associated to a border node and to its parent (an extra-border node) can intersect. Overall, the certificates have size $O(k/h) = O(k/t)$. The verifier checks the orientation and the marking, then recovers the certificates of the special nodes, as in Lemma 5.4, and performs the simulation as in Lemma 5.3. This verification can be done with a view of radius $t \leq 2h$, yielding the desired distance-t proof labelling scheme. □

5.2.2 Linear scaling in cycles and grids

For the proof techniques of Theorem 5.1 to apply to other graphs, we need to compute a partition of the nodes into the two categories, special and standard, satisfying three main properties. First, the partition should split the graph into regions formed by standard nodes, separated by special nodes. Second, each region should have a diameter small enough for allowing special nodes at the border of the region to simulate the standard nodes in that region, as in Lemma 5.3. Third, the regions should have a diameter large enough to allow efficient spreading of certificates assigned to special nodes over the standard nodes, as in Lemma 5.4. For any graph family in which one can define such a decomposition, an analogue of Theorem 5.1 holds. We show that this is the case for cycles and grids.

Corollary 5.1. *Let $\mathcal{P}$ be a predicate on labelled cycles, and let us assume that there exists a (distance-1) proof-labelling scheme $(\mathbf{p}, \mathbf{v})$ for $\mathcal{P}$ with $\mathsf{size}(\mathbf{p}, \mathbf{v}) = k$. Then there exists a proof-labelling scheme for $\mathcal{P}$ that scales linearly, that is, $\mathsf{size\text{-}pls}(\mathcal{P}, t) = O\left(\frac{k}{t}\right)$. The same holds for predicates on 2-dimensional labelled grids.*

Proof. We just explain how the proof of Theorem 5.1 can be adapted to apply for cycles and grids.

For every labelled cycle, the prover picks an arbitrary node r, which will play the same role as the root chosen in the proof of Theorem 5.1, and an orientation of the cycle pointing toward r. The chosen node is called *leader.* Let $h = \lfloor t/2 \rfloor$. The prover marks as border node every node u such that $\textsc{dist}(r, u) \equiv 0 \pmod h$, where the distance is taken in the oriented cycle. Similarly, the prover marks as extra-border node every node u such that $\textsc{dist}(r, u) \equiv -1 \pmod h$. Note that the border and extra-border nodes that are the furthest away from the leader by the orientation may actually be close to the leader in the undirected cycle. As this may cause difficulties for spreading the certificates, the prover does not mark them, and keep these nodes standard. The domain of a border node is defined in a way similar to trees. The leader, the orientation, and the marking can be checked in a way similar to the proof of Lemma 5.2. In particular, observe that that the size of each domain is at most t. The marking separates the graph into independent domains that can be simulated in parallel as in Lemma 5.3. The diameter of each domain is at least $t/2$ which allows to do the spreading as in Lemma 5.4, resulting in certificates of size $O(k/t)$.

For every labelled grid, the prover provides the edges with north-south and east-west orientations, using two counters modulo 3. In a grid $p \times q$ with $n = pq$ nodes, this orientation induces a coordinate system with edges directed from $(0, 0)$, defined as the south-west corner, to (p, q), defined as the north-east corner. The leader is the node at position $(0, 0)$. Let $h = \lfloor t/4 \rfloor$. The partition of the nodes is as follows. Every node with coordinate (x, y) where both x and y are 0 modulo h are the border nodes. Non-border nodes with coordinate (x, y) where x or y equals 0 or -1 modulo

h are extra-border nodes. Now, as for cycles, we slightly modify that decomposition for avoiding domains with too small diameter. Specifically, north-most border and extra-border nodes, and the east-most border and extra-border nodes are turned back to standard nodes. The domain of a border node is composed of all nodes with larger x-coordinate and larger y-coordinates for which there are no closer border node in the oriented grid. Using the same technique as in the proof of Lemma 5.2, this partition of the nodes can be checked locally. The simulation as performed in the proof of Lemma 5.3 can be performed similarly using that decomposition, because the domains have diameter at most t, and are well separated by special nodes. Finally, the spreading of the certificates as in Lemma 5.4 can be done in the following way. For every special node (x, y) where x equals 0 or -1 modulo h, the certificate is spread over the $h-1$ nodes to the east. For every special node (x, y) where y equals 0 or -1 modulo h, the certificate is spread over the $h-1$ nodes to the north. Note that there is always enough space to the east or to the north of the special nodes as we have removed the special nodes that could be too close to the east and north borders. Also note that some node have their certificates spread in two directions, but this does not cause problem as it just increases the size of the certificates by a constant factor. □

By the same techniques, Corollary 5.1 can be generalized to toroidal 2-dimensional labelled grids, as well as to d-dimensional labelled grids and toruses, for every $d \geq 2$.

5.3 Universal scaling of uniform schemes

It is known [90] that, for every predicate $\mathcal{P}$ on labeled graphs with $\mathsf{size\text{-}pls}(\mathcal{P}) = \widetilde{\Omega}(n^2)$, there is a proof-labelling scheme that scales linearly on the interval $[1, D]$ in graphs of diameter D. We show that, in fact, the scaling factor can be much larger. First, recall that a graph $G = (V, E)$ has *growth* $b = b(t)$ if, for every $v \in V$, and every $t \in [1, D]$, we have $|B_G(v, t)| \geq b(t)$. We say that a proof-labelling scheme is *uniform* if the same certificate is assigned to all nodes by the prover.

Theorem 5.2. *Let $\mathcal{P}$ be a predicate on labeled graphs, and let us assume that there exists a uniform 1-PLS $(\mathbf{p}, \mathbf{v})$ for $\mathcal{P}$ with $\mathsf{size}(\mathbf{p}, \mathbf{v}) = k$. There is a proof-labelling scheme for $\mathcal{P}$ that weakly scales with scaling factor $b(t)$ on graphs of growth $b(t)$. More specifically, let $G = (V, E)$ be a graph, let $t_0 = \min\{t \geq 1 \mid b(t) \geq \log n\}$, and $t_1 = \max\{t \geq 1 \mid k \geq b(t)\}$. Then, in G, for every $t \in [t_0, t_1]$, $\mathsf{size\text{-}pls}(\mathcal{P}, t) = \widetilde{O}\left(\frac{k}{b(t)}\right)$.*

Proof. Let $s = (s_1, \ldots, s_k)$, where $s_i \in \{0, 1\}$ for every $i = 1, \ldots, k$, be the k-bit certificate assigned to every node of a graph $G = (V, E)$. Let $t \geq 1$ be such that $k \geq b(t) \geq c \log n$ for a constant c large enough. For every node $v \in V$, we set the certificate of v, denoted $s^{(v)}$, as follows: for every $i = 1, \ldots, k$, v stores the pair (i, s_i) in $s^{(v)}$ with probability $\frac{c \log n}{b(t)}$. Recall the following Chernoff bounds: Suppose $Z_1, \ldots, Z_m$ are independent random variables taking values in $\{0, 1\}$, and let $Z = \sum_{i=1}^{m} Z_i$. For every $0 \leq \delta \leq 1$, we have $\Pr[Z \leq (1-\delta)\mathbb{E}Z] \leq e^{-\frac{1}{2}\delta^2 \mathbb{E}Z}$, and $\Pr[Z \geq (1+\delta)\mathbb{E}Z] \leq e^{-\frac{1}{3}\delta^2 \mathbb{E}Z}$.

- On the one hand, for every $v \in V$, let X_v be the random variable equal to the number of pairs stored in $s^{(v)}$. By Chernoff bounds, we have $\Pr[X_v \geq$

$\frac{2c\,k\log n}{b(t)}] \leq e^{\frac{c\,k\log n}{3\,b(t)}} = n^{-\frac{c\,k}{3\,b(t)}}$. Therefore, by union bound, the probability that a node v stores more than $\frac{2c\,k\log n}{b(t)}$ pairs (i, s_i) is at most $n^{1-\frac{c\,k}{3\,b(t)}}$, which is less than $\frac{1}{2}$ for c large enough.

- On the other hand, for every $v \in V$, and every $i = 1, \ldots, k$, let $Y_{v,i}$ be the number of occurrences of the pair (i, s_i) in the ball of radius t centered at v. By Chernoff bound, we have $\Pr[Y_{v,i} \leq \frac{1}{2}c\log n] \leq e^{-\frac{c\log n}{8}} = n^{-c/8}$. Therefore, by union bound, the probability that there exists a node $v \in V$, and an index $i \in \{1, \ldots, k\}$ such that none of the nodes in the ball of radius t centered at v store the pair (i, s_i) is at most $kn^{1-c/8}$, which is less than $\frac{1}{2}$ for c large enough.

It follows that, for c large enough, the probability that no nodes store more than $\widetilde{O}(k/b(t))$ pairs (i, s_i), and every pair (i, s_i) is stored in at least one node of each ball of radius t, is positive. Therefore, there is a way for a prover to distribute the pairs (i, s_i), $i = 1, \ldots, k$, to the nodes such that (1) no nodes store more than $\widetilde{O}(k/b(t))$ bits, and (2) every pair (i, s_i) appears at least once in every t-neighbourhood of each node. At each node v, the verification procedure first collects all pairs (i, s_i) in the t-neighbourhood of v, in order to recover s, and then runs the verifier of the original (distance-1) proof-labelling scheme. Finally, we emphasize that we only use probabilistic arguments as a way to prove the existence of certificate assignment, but the resulting proof-labelling scheme is deterministic and its correctness is not probabilistic. □

Theorem 5.2 finds direct application to the universal proof-labelling scheme [67] using $O(n^2+kn)$ bits in n-node graphs labeled with k-bit labels. Indeed, the certificate of each node consists of the $n \times n$ adjacency matrix of the graph, the array of n entries, each one equal to the k-bit label at the corresponding node, and the array of n entries listing the identities of the n nodes. It was proved in [90] that the universal proof-labelling scheme can be scaled by a factor t. Theorem 5.2 significantly improves that result, by showing that the universal proof-labelling scheme can actually be scaled by a factor $b(t)$, which can be exponential in t.

Corollary 5.2. *For every predicate $\mathcal{P}$ on labeled graphs, there is a proof-labelling scheme for $\mathcal{P}$ as follows. For every graph G with growth $b(t)$, let $t_0 = \min\{t \geq 1 \mid b(t) \geq \log n\}$. Then, for every $t \geq t_0$ we have* $\textsf{size-pls}(\mathcal{P}, t) = \widetilde{O}\left(\frac{n^2+kn}{b(t)}\right)$.

Theorem 5.2 also tells us that, if a proof-labelling scheme is involving certificates in which the same k-bit sub-certificate is assigned to every node, then the size of this common sub-certificate can be drastically reduced by using a t-round verification procedure. This is particularly interesting when the size of the common sub-certificate is large compared to the size of the rest of the certificates. An example of such a scheme is in essence the one described in Corollary 2.2 of [80]. Given a parameter $k \in \Omega(\log n)$, let $\textsc{iso}_k$ be the predicate on graph stating that there exist two vertex-disjoint isomorphic induced subgraphs of size k in the given graph.

Corollary 5.3. *For every $k \in \left[1, \frac{n}{2}\right]$, we have* $\textsf{size-pls}(\textsc{iso}_k) = \Theta(k^2)$ *bits, and, for every $t > 1$,* $\textsf{size-pls}(\textsc{iso}_k, t) = \widetilde{O}(k^2/b(t))$.

Proof. We first sketch a 1-PLS. Every node is given as certificate the $k \times k$ adjacency matrix of the two isomorphic subgraphs, along with the corresponding IDs of the nodes in the two subgraphs. The certificates also provides each node with the ID of an arbitrary node in each subgraphs, that we call the leaders. In addition, the nodes are given certificates that correspond to two spanning trees rooted at the two leaders. The verification procedure works as follows. Every node first checks that the spanning trees structures are correct. Then the roots of the spanning trees check that they are marked as leader. Finally every node whose ID appear in one of the two adjacency matrices checks that its actual neighbourhood corresponds to what it should be according to the given adjacency matrices. By construction, this is a valid 1-PLS, using certificates on $O(k^2 + \log n)$ bits. A simple adaptation of the proof of Theorem 6.1 of [67] enables to prove that $\Omega(k^2)$ bits are needed. The result regarding t-PLS is a direct application of Theorem 5.2 to the part of the certificate that is common to all nodes. □

5.4 Certifying distance-related predicates

For any labeled (weighted) graph (G, x), the predicate $\textsc{diam}$ on (G, x) states whether, for every $v \in V(G)$, $x(v)$ is equal to the (weighted) diameter of G.

Theorem 5.3. *There is a proof-labelling scheme for* $\textsc{diam}$ *that scales linearly between* $[c \log n, n/\log n]$*, for some constant c. More specifically, there exists* $c > 0$*, such that, for every* $t \in [c \log n, n/\log n]$*,* $\textsf{size-pls}(\textsc{diam}, t) = \widetilde{O}\left(\frac{n}{t}\right)$*. Moreover, no proof-labelling schemes for* $\textsc{diam}$ *can scale more than linearly on the interval* $[1, n/\log n]$*, that is, for every* $t \in [1, n/\log n]$*,* $\textsf{size-pls}(\textsc{diam}, t) = \widetilde{\Omega}\left(\frac{n}{t}\right)$*.*

The theorem follows from the next lemmas.

Lemma 5.5. *There exists a constant c, such that for every* $t \in [c \log n, n]$*,* $\textsf{size-pls}(\textsc{diam}, t) = O\left(\frac{n \log^2 n}{t}\right)$*.*

Proof. A proof-labelling scheme for diameter with optimal certificate size $\Theta(n \log n)$ bits has been designed in [24]. We simply use this scheme for certifying that, for every node v, the diameter of the graph is at least $x(v)$. Indeed, [24] uses only $O(\log n)$-bit certificates to certify the existence of a pair of nodes at mutual distance at least $x(v)$ in the graph. The rest of the proof is dedicated to certifying that no pairs of nodes are at distance more than $x(v)$ in the graph, i.e., "diameter $\leq x(v)$". Namely, we show how the scheme in [24] scales with the radius of verification. For this purpose, let us briefly recall this scheme. Each node v of a graph $G = (V, E)$ is provided with a certificate D_v consisting of a table with n entries storing the ID of every node in G, and the distance to these nodes. (Every certificate is therefore on $O(n \log n)$ bits). Somewhat abusing notations, let us denote by $D_v(u)$ the distance to node u, as stored in table D_v. The verification proceeds by, first, having each node checking that it stores the same set of IDs as the ones stored by its neighbours, and that its own ID appears in its table. Second, each node checks that the distances in its certificate vary as expected. That is, each node v checks that: (1) $D_v(v) = 0$, (2) for every node u and every neighbour v', $D_v(u) - w(\{v, v'\}) \leq D_{v'}(u) \leq D_v(u) + w(\{v, v'\})$, and (3) there exists a neighbour v' such that $D_{v'}(u) = D_v(u) - w(\{v, v'\})$. Finally, every node v checks that $D_v(u) \leq x(v)$ for every node u. This verification process is correct, as shown in [24].

Now, let $t \geq c \log n$ for a constant $c > 0$ large enough, and let us construct a proof-labelling scheme for "diameter $\leq x(v)$", with radius t. The idea is that each node v does not store all entries in the table D_v but only a fraction t of these entries. The issue is to select which entries to keep, and which entries to drop. For our scheme to work, we need to guarantee that, if the distance to node u is not stored in D_v, then there is a node v' on a shortest path from v to u, at distance at most t from v, that stores $\text{dist}(v', u)$ in its table $D_{v'}$. To achieve this, for every node $u \neq v$, each node v keeps $\text{dist}(v, u)$ in its table D_v with probability $\frac{c \log n}{t}$. (Node v systematically keeps $D_v(v) = 0$ in its table). From this setting, we derive the following two properties.

1. For every pair of nodes (u, v), let us denote by $P_{v,u}$ the path of length t formed by the t first nodes on a shortest path from v to u, and let $X_{v,u}$ denote the sum of t Bernoulli random variables with parameter $\frac{c \log n}{t}$. By the use of the same Chernoff bound as in the proof of Theorem 5.2, we have $\Pr[X_{v,u} \leq \frac{1}{2} c \log n] \leq e^{-\frac{c}{8} \log n} = n^{-c/8} < \frac{1}{2n^2}$ for c large enough. Therefore, by union bound, the probability that there exists a pair of nodes (u, v) such that no nodes of $P_{v,u}$ store the distance to node u is less than $\frac{1}{2}$.

2. For every node v, let Y_v be the number of nodes for which v keeps the distance these nodes. Again, by Chernoff bound, $\Pr[Y_v \geq \frac{2cn \log n}{t}] \leq e^{-\frac{c\, n \log n}{3\, t}} \leq e^{-\frac{c \log n}{3}} = n^{-c/3} < \frac{1}{2n}$ for c large enough. Therefore, by union bound, the probability that there exists a node v that stores the distances to more than $\frac{2cn \log n}{t}$ nodes is less than $\frac{1}{2}$.

Let $\mathcal{E}_{v,u}$ be the event "at least one node of $P_{v,u}$ stores its distance to node u", and let $\mathcal{E}'_v$ be the event "node v stores no more than $\frac{2cn \ln n}{t}$ distances to other nodes". We derive from the above that

$$\Pr[\forall (u, v) \in V \times V, \mathcal{E}_{v,u} \wedge \mathcal{E}'_v] > 0.$$

It follows that there exists an assignment of entries to be kept in each table $D_v, v \in V$, such that each resulting partial table is of size $O\left(\frac{n \log^2 n}{t}\right)$ bits, and, for every two nodes u and v, at least one node at distance at most t, on a shortest path from v to u, stores its distance to node u.

It remains to show that these sparse certificates can be verified in t rounds. Let $B(v, t)$ be the ball of radius t around v. Each node v verifies that, first, for every node $v' \in B(v, t)$ such that both v and v' stores the distance to a same node u, we have $D_{v'}(u) - \text{dist}(v, v') \leq D_v(u) \leq D_{v'}(u) + \text{dist}(v, v')$, and, second, for every node u such that v stores its distance to u, there exists a node $v' \in B(v, t)$ such that $D_v(u) = D_{v'}(u) + \text{dist}(v, v')$. Third, using the distances collected in $B(v, t)$, node v constructs the table D'_v where $D'_v(u) = D_v(u)$ if u is stored in D_v, and $D'_v(u) =$

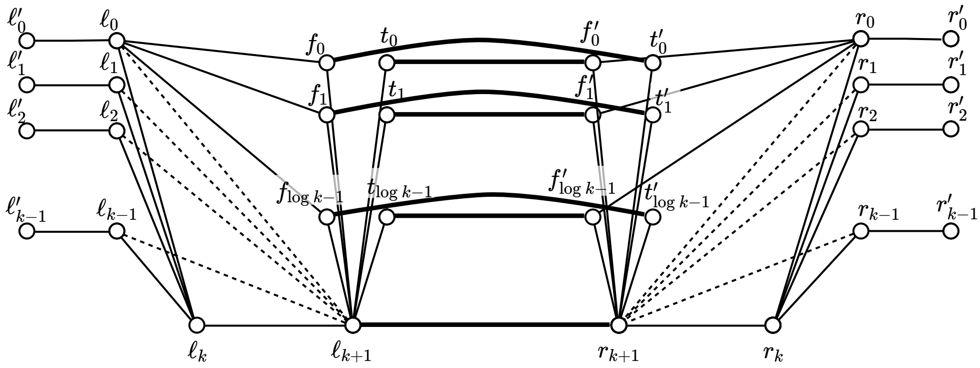


Figure 5.1: The lower bound graph construction. Thin lines represent P-paths, thick lines represent $(2t+1)$-paths, and the dashed lines represent edges who's existence depend on the input. The paths connecting ℓ_i and r_i to their binary representations are omitted, except for those of ℓ_0 and r_0.

$\min_{v' \in B(v,t)}(D_{v'}(u) + \mathrm{dist}(v, v'))$ otherwise. Finally, node v checks that $D'_v(u) \leq x(v)$ for every node u. If all these tests are passed, then v accepts, otherwise it rejects.

By the setting of the partial tables $D_v, v \in V$, in a legal instance, we get that $D'_v(u) = \mathrm{dist}(v, u)$ for every node u, and therefore all nodes accepts. Instead, if there exists $(u, v) \in V \times V$ such that information about u are stored in D_v, but $D_v(u) \neq \mathrm{dist}(u, v)$, then let us consider such a pair (u, v) where $D_v(u)$ is minimum. For v to accept, there must exist some node $v' \in B(v, t)$ such that $D_{v'}(u) = D_v(u) - \mathrm{dist}(v, v')$. By the choice of the pair (u, v), $D_{v'}(u) = \mathrm{dist}(u, v')$, and thus $D_v(u) = \mathrm{dist}(u, v)$, a contradiction. Therefore v' cannot exist, and thus v rejects. It follows that this scheme is a correct t-PLS for diameter, using $O\left(\frac{n \log^2 n}{t}\right)$-bit certificates. □

Lemma 5.6. *For every $t \in [1, n/\log n]$, $\mathsf{size\text{-}pls}(\textsc{diam}, t) = \Omega\left(\frac{n}{t \log n}\right)$.*

Peleg and Rubinovich [96] proved distributed lower bounds by a reduction to communication complexity; Abboud et al [1] gave a lower bound on the approximability of the diameter, using a simpler reduction; both results are in the CONGEST model. We adapt the graph construction from the latter and the simulation construction of the former in order to prove lower bounds on the label size in the t-PLS model. Interestingly, in our model there are no bandwidth restrictions, which makes it more similar to the LOCAL model than to the CONGEST model, yet our lower bounds are based on a CONGEST lower bound.

We now describe the construction of the lower bound graph (see Figure 5.1). Let $k = \Theta(n)$ be a parameter whose exact value will follow from the graph construction. Alice and Bob use the graph in order to decide DISJ on k-bit strings. Let $P \geq 1$ be a constant, and let t be the parameter of the t-PLS, which may or may not be constant. The graph consists of the following sets of nodes: $L = \{\ell_0, \dots, \ell_{k-1}\}$, $L' = \{\ell'_0, \dots, \ell'_{k-1}\}$, $T = \{t_0, \dots, t_{\log k-1}\}$, $F = \{f_0, \dots, f_{\log k-1}\}$, and ℓ_k and ℓ_{k+1}, which will be simulated by Alice, and similarly $R = \{r_0, \dots, r_{k-1}\}$, $R' = \{r'_0, \dots, r'_{k-1}\}$, $T' =$

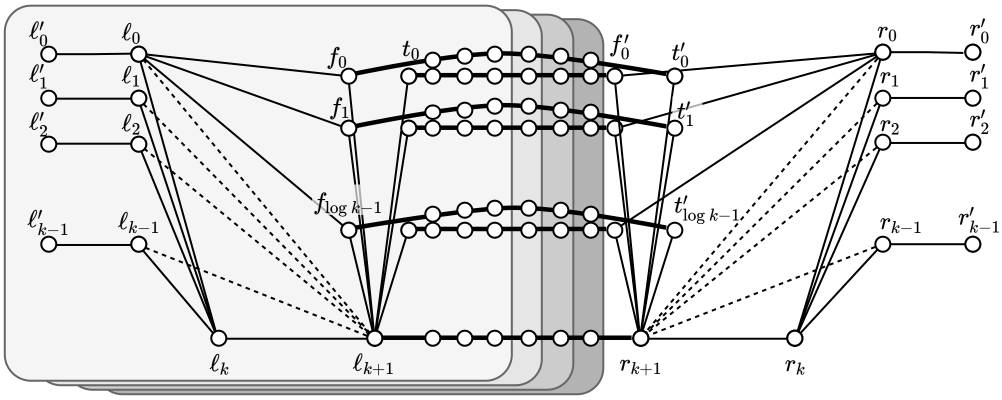


Figure 5.2: The lower bound graph construction for $t = 3$, and the sets of nodes simulated by Alice in the three rounds of verification. Alice eventually knows the outputs of all the nodes in the light-shaded set.

$\left\{t'_0, \ldots, t'_{\log k-1}\right\}$, $F' = \left\{f'_0, \ldots, f'_{\log k-1}\right\}$, and r_k and r_{k+1}, which will be simulated by Bob.

The nodes are connected by paths, where the paths consist of additional, distinct nodes. For each $0 \leq i \leq k-1$, connect with P-paths (i.e., paths of P edges and $P-1$ new nodes) the following pairs of nodes: (ℓ_i, ℓ'_i), (ℓ_i, ℓ_k), (ℓ_k, ℓ_{k+1}), (r_i, r'_i), (r_i, r_k), and (r_k, r_{k+1}). Add such paths also between ℓ_{k+1} and all $t_h \in T$ and $f_h \in F$, and between r_{k+1} and all $t'_h \in T'$ and $f'_h \in F'$. Connect by a P-path each $\ell_i \in L$ with the nodes representing its binary encoding, that is, connect ℓ_i to each t_h that satisfies $i[h] = 1$, and to each f_h that satisfies $i[h] = 0$, where $i[h]$ is bit h of the binary encoding of i. Add similar paths between each $r_i \in R$ and its encoding by nodes t'_h and f'_h. In addition, for each $0 \leq h \leq \log k - 1$, add a $(2t+1)$-path from t_h to f'_h and from f_h to t'_h, and a similar path from ℓ_{k+1} to r_{k+1}.

Assume Alice and Bob want to solve the DISJ problem for two k-bit strings S_A and S_B using a non-deterministic protocol. They build the graph described above, and add the following edges: (ℓ_i, ℓ_{k+1}) whenever $S_A[i] = 0$, and (r_i, r_{k+1}) whenever $S_B[i] = 0$. The next claim is at the heart of our proof.

Claim 5.1. *If S_A and S_B are disjoint then $D = 4P + 2t + 2$, and otherwise $D \geq 6P + 2t + 1$.*

Proof. A more detailed analysis can be found in [1]. Each node in L, T, F or on a path connecting two such nodes is at distance at most $2P$ from ℓ_{k+1}, and each node in R, T', F' or in a path connecting two such nodes is at distance at most $2P$ from r_{k+1}. Since ℓ_{k+1} and r_{k+1} are connected by a $(2t+1)$-path, the distance between every two nodes in L, R, T, F, T', F' or on a path between them is at most $4P + 2t + 1$.

Consider two nodes ℓ'_i and r'_j with indices $i \neq j$, an index h such that $i[h] \neq j[h]$, and assume w.l.o.g that $h[i] = 1$. In this case, there is a simple path connecting the following nodes: $\ell'_i, \ell_i, t_j, f'_h, r_j, r'_j$, and its length is $4P + 2t + 1$. For two nodes ℓ'_i and r'_i of the same index, a simple case analysis shows that the distance between these nodes is at most $6P + 2t + 1$.

To see the claim, note that if the inputs are disjoint then for every $0 \leq i \leq k-1$, at least one of the edges (ℓ_i, ℓ_{k+1}) or (r_i, r_{k+1}) exists, and thus there is a $(4P+2t+2)$-path between ℓ'_i and r'_i, through ℓ_{k+1} or through r_{k+1}, e.g. $\ell'_i, \ell_{k+1}, r_{k+1}, r_k, r_i, r'_i$. The other distances in the graph are not larger, and the distance between ℓ'_i and r'_i is indeed $4P+2t+2$ whenever $S_A[i] \neq 0$ or $S_B[i] \neq 0$ [2] so the diameter is indeed $4P+2t+2$. On the other hand, if the sets are not disjoint, there is an index i such that $S_A[i] = S_B[i] = 1$, and the distance between ℓ'_i and r'_i is $6P+2t+1$ so the diameter is at least $6P+2t+1$. □

With this claim in hand, we are ready to prove Lemma 5.6.

Proof of Lemma 5.6. Fix $t \in [1, n/\log n]$, and let S_A and S_B be two input strings for the DISJ problem on k bits. We show how Alice and Bob can solve DISJ on S_A and S_B in a nondeterministic manner, using the graph described above and a t-PLS for DIAM $= 4P+2t+2$.

Alice and Bob simulate the verifier on the labeled graph. The nodes simulated by Alice, denoted A, are $L \cup L' \cup T \cup F \cup \{\ell_k, \ell_{k+1}\}$ and all the paths between them, and by Bob, denoted B, are $R \cup R' \cup T' \cup F' \cup \{r_k, r_{k+1}\}$ and the paths between them. For each pair of nodes $(a,b) \in A \times B$ that are connected by a $(2t+1)$-path, let P_{ab} be this path, and $\{P_{ab}(i)\}$, $i = 0, \ldots, 2t+1$ be its nodes in consecutive order, where $P_{ab}(0) = a$ and $P_{ab}(2t+1) = b$. Let C be the set of all $(2t+1)$-path nodes, i.e. $C = V \setminus (A \cup B)$. The nodes in C are simulated by both players, in a decremental way described below.

Alice interprets her nondeterministic string as the certificates given to the nodes in $A \cup C$, and she sends the certificates of C to Bob. Bob interprets his nondeterministic string as the certificates of B, and gets the certificates of C from Alice. They simulate the verifier execution for t rounds, where, in round $r = 1, \ldots, t$, Alice simulates the nodes of A and all nodes $P_{ab}(i)$ with $(a,b) \in A \times B$ and $i \leq 2t+1-r$, while Bob simulates the nodes of B and all nodes $P_{ab}(i)$ with $i \geq r$.

Note that this simulation is possible without further communication. The initial state of nodes in A is determined by S_A, the initial state of the nodes $P_{ab}(i)$ with $i \leq 2t$ is independent of the inputs, and the certificates of both node sets are encoded in the nondeterministic string of Alice. In each round of verification, all nodes whose states may depend on the input of Bob or on his nondeterministic string are omitted from Alice's simulation, and so she can continue the simulation without communication with Bob. Similar arguments apply to the nodes simulated by Bob. Finally, each node is simulated for t rounds by at least one of the players. Thus, if the verifier rejects, that is, at least one node rejects, then at least one of the players knows about this rejection.

Using this simulation, Alice and Bob can determine whether DISJ on (S_A, S_B) is true as, from Claim 5.1, we know that if it is true then DIAM $= 4P+2t+2$, and the verifier of the PLS accepts, while otherwise it rejects. The nondeterministic communication complexity of the true case of DISJ on k-bit strings is $\Omega(k) = \Omega(n)$,

[2] In the case where both inputs are the 0 strings, the diameter is only $4P+2t+1$; we can exclude this unique case by forbidding this input to the DISJ problem without changing its asymptotic complexity

so Alice and Bob must communicate this amount of bits. From the graph definition, $|C| = \Theta(t \log n)$ which implies $\mathsf{size\text{-}pls}(\textsc{diam}, t) = \Omega\left(\frac{n}{t \log n}\right)$, as desired. □

Let β be a non-negative integer. For any labeled graph (G, x), β-ADDITIVE SPANNER is the predicate on (G, x) that states whether, for every $v \in V(G)$, $x(v) \subseteq \{\mathrm{ID}(w), w \in N(v)\}$ for every $v \in V(G)$, and whether the collection of edges $E_H = \{\{v, w\}, v \in V(G), w \in x(v)\}$ forms a β-additive spanner of G, i.e., a subgraph H of G such that, for every two nodes s, t, we have $\textsc{dist}_H(s,t) \leq \textsc{dist}_G(s,t) + \beta$. Similarly, the predicate (α, β)-SPANNER on (G, x) states whether H is such that, for every two nodes s, t, $\textsc{dist}_H(s,t) \leq \alpha\ \textsc{dist}_G(s,t) + \beta$.

Theorem 5.4. *There is a proof-labelling scheme for additive-spanner that weakly scales linearly on the interval $[1, n/\log n]$. More precisely, for every pair of integers $\alpha \geq 1$ and $\beta \geq 0$, there is a one-round proof-labelling scheme for (α, β)-SPANNER of size $O(n)$, and a t-round proof-labelling scheme of size $\widetilde{O}(\frac{n}{t})$. Moreover, this is optimal for additive spanners with a constant β and $t \in [1, n/\log n]$, that is,* $\mathsf{size\text{-}pls}(\beta\text{-}\textsc{additive spanner}, t) = \widetilde{\Theta}(\frac{n}{t})$ *for β and t as above.*

Proof. The upper bound proof is similar to the one of Lemma 5.5 for diameter. Specifically, instead of using a unique table D_v storing distances in the graph, every node v stores two tables D_v and $\widehat{D}_v$, where $\widehat{D}_v$ stores the distances in the spanner. To scale, each node keeps only a fraction $\frac{c \log n}{t}$ of the entries in each table, for some constant $c > 0$. For c large enough, this is sufficient for every node to recover its distance to every other node in both the graph and the spanner, by using the same arguments as in the proof of Lemma 5.5. The verification again proceeds as in the proof of Lemma 5.5, excepted that, instead of checking whether $D_v(u) \leq x(v)$ for every node u, every node v checks that, for every node u, $\widehat{D}_v(u) \leq \alpha D_v(u) + \beta$ or $\widehat{D}_v(u) \leq D_v(u) + \beta$, depending on whether one is dealing with general spanners or additive spanners, respectively. Correctness directly follows from the same arguments as in the proof of Lemma 5.5.

To achieve the lower bound, we use the same construction as in the proof of Lemma 5.6. The original graph is the graph described above for the case where both S_A and S_B are the all-0 strings, i.e. all the potential edges are present in the graph. To decide DISJ, Alice and Bob build a spanner for this graph in the same manner as in the proof of Lemma 5.6, i.e. keep all the nodes and paths, and keep the edges that correspond to zeros in their input strings. This construction guarantees that each graph edge that is not in the spanner can be replaced by a $2P$-path in it. If the inputs are disjoint, then at most one edge in each shortest path is not in the spanner, so the stretch is at most $2P - 1$. On the other hand, if the inputs are not disjoint then there is a pair (ℓ'_i, r'_i) that stretches from $4P + 2t + 1$ in the original graph to $6P + 2t + 1$ in the spanner.

We pick P such that the spanner is a legal (α, β)-spanner if and only if the inputs are disjoint, that is, $4P{+}2t{+}2 \leq \alpha(4P{+}2t{+}1){+}\beta$, while $6P{+}2t{+}1 > \alpha(4P{+}2t{+}1){+}\beta$. For additive spanners, i.e. $\alpha = 1$, the first condition always holds; to guarantee the second, we choose $P > \beta/2$. The lower bound is thus $\Omega\left(\frac{n}{\beta t}\right)$, or $\Omega\left(\frac{n}{t}\right)$ for a constant β. □

5.5 Distributed proofs for spanning trees

In this section, we study two specific problems which are classical in the domain of proof-labelling schemes: the verification of a spanning tree, and of a minimum-weight spanning tree. The predicates ST and MST are the sets of labeled graphs where some edges are marked and these edges form a spanning tree, and a minimum spanning tree, respectively. We present proof-labelling schemes for them that scale linearly in t. Note that ST and MST are problems on general labeled graphs and not on trees, i.e., the results in this section improve upon Section 5.3 (for these specific problems), and are incomparable with the results of Section 5.2.

Formally, let $\mathcal{F}$ be the family of all connected undirected, weighted, labeled graphs (G, x). For simplicity, we assume all edge weights are distinct, and thus the minimum spanning tree is also unique. Each label $x(v)$ contains a (possibly empty) subset of edges adjacent to v, which is consistent with the neighbours of v, and we denote the collection of edges represented in x by T_x. In the ST (respectively, MST) problem, the goal is to decide for every labeled graph $(G, x) \in \mathcal{F}$ whether T_x is a spanning tree of G (respectively, whether T_x is a spanning tree of G with the sum of all its edge-weights minimal among all spanning trees of G). For these problems we have the following results.

Theorem 5.5. *For every $t \in O(\log n)$, we have that* $\textsf{size-pls}(\mathrm{ST}, t) = O\left(\frac{\log n}{t}\right)$.

Ostrovsky et al. [90] (Theorem 8) designed a t-distance proof-labelling scheme for acyclicity, with $(\log n/t)$-bit certificates, for $t \leq \min\{\log n, D\}$. Here, D is the diameter of the graph, which is at least the largest depth of a tree in it. In the scheme, each tree is oriented outwards from an arbitrarily chosen root, and after running the verification process, each node knows who is his parent in its tree, and the root of each tree knows it is the root. This scheme plays an essential role in the proof of Theorem 5.5.

Proof of Theorem 5.5. To prove that a marked subgraph T_x is a spanning tree, we need to verify it has the following properties: (1) spanning the graph, (2) acyclic, (3) connected. We choose an arbitrary node as the root of T_x.

The certificate of a node v is composed of three parts: a bit indicating whether the tree is shallow as in the proof of Theorem 5.1, $O(\log n/t)$-bits by the scheme of Ostrovsky et al. [90] (Theorem 8) for acyclicity, and $O(\log n/t)$-bits that are a part of the $\textsf{ID}$ of the root, as in the proof of Theorem 5.1.

In the verification process, the nodes first verify they all have the same first bit. If the tree is shallow, all nodes but the root accept, and the root collects the whole structure of the graph and of T_x and verifies it is a spanning tree. Otherwise, each node verifies that at least one of its edges is marked to be in T_x, making sure T_x is spanning all the graph. The nodes then run the verification process from [90] (Theorem 8), while ignoring edges not in T_x, to make sure the graph is acyclic. Finally, they all run the reconstruction process from Theorem 5.2 to find the root $\textsf{ID}$, and the root of T_x (as defined by the acyclicity scheme) verifies this root $\textsf{ID}$ is indeed its own $\textsf{ID}$. This guarantees T_x is a connected forest, i.e. a tree, as desired. □

Theorem 5.6. *For every $t \in O(\log n)$, we have that* $\textsf{size-pls}(\textsc{mst}, t) = O\left(\frac{\log^2 n}{t}\right)$.

The upper bound matches the lower bound of Korman et al. [83] (Corollary 3) for the specific case $t = \Theta(\log n)$. Note that our theorem holds only for $t \in O(\log n)$, meaning that we can get from proofs of size $O(\log^2 n)$ to proofs of size $O(\log n)$, but not to a constant.

Our upper bound is based on a famous 1-round PLS for MST [79, 83], which in turn builds upon the algorithm of Gallager, Humblet, and Spira (GHS) [63] for a distributed construction of MST. The idea behind this scheme is, given a labeled graph (G, x), to verify that T_x is consistent with an execution of the GHS algorithm in G.

The GHS algorithm maintains a spanning forest that is a subgraph of the minimum spanning tree, i.e., the trees of the forest are fragments of the desired minimum spanning tree. The algorithm starts with a spanning forest consisting of all nodes and no edges. At each phase each of the fragments adds the minimum edge going out of it, thus merging several fragments into one. After $O(\log n)$ iterations, all the fragments are merged into a single component, which is the desired minimum-weight spanning tree. We show that each phase can be verified with $O(\log n/t)$ bits, giving a total complexity of $O(\log^2 n/t)$ bits.

Proof. Let (G, x) be a labeled graph such that T_x is a minimum-weight spanning tree. If t is greater than the diameter D of G, every node can see the entire labeled graph in the verification process, and we are done; we henceforth assume $t \leq D$. The certificates consist of four parts.

First, we choose a root and orient the edges of T_x towards it. We give each node its distance from the root modulo 3, which allows it to obtain the ID of its parent and the edge pointing to it in one round. Second, we assign the certificate described above for ST (Theorem 5.5), which certifies that T_x is indeed a spanning tree. This takes $O(\log n/t)$ bits.

The third part of the certificate tells each node the phase in which the edge connecting it to its parent is added to the tree in the GHS algorithm, and which of the edge's endpoints added it to the tree. Note that after one round of verification, each node knows for every incident edge, at which phase it is added to the spanning tree, and by which of its endpoints. This part uses $O(\log\log n)$ bits.

The fourth part of the certificate consists of $O(\log^2 n/t)$ bits, $O(\log n/t)$ for each phase of the GHS algorithm. To define the part of a certificate of every phase, fix a phase, a fragment F in the beginning of this phase, and let $e = (u, v)$ be the minimum-weight edge going out of F, where $u \in F$ and $v \notin F$. Our goal is that the nodes of F verify together that e is the minimum-weight outgoing edge of F, and that no other edge was added by F in this phase. To this end, we first orient the edges of F towards u, i.e. set u as the root of F. If the depth of F is less than t, then in $t - 1$ rounds the root u can see all of F and check that (u, v) is the lightest outgoing edge. All other nodes just have to verify that no other edge is added by the nodes of F in this phase. Otherwise, if the depth of F is at least t, by Theorem 5.2, the information about $\mathsf{ID}(u)$ and $w(e)$ can be spread on F such that in t rounds it can be collected by all nodes of F. With this information known to all the nodes of F, the root can locally verify that it is named as the node that adds the edge and that it has the named edge with the right weight. The other nodes of F can locally verify that they do not have incident edges with a smaller weight, and that no other edge is added by F. This part takes $O(\log n/t)$ bits per iteration, which sums to a total of $O(\log^2 n/t)$ bits.

Overall, our scheme verifies that T_x is a spanning tree, and that it is consistent with every phase of the GHS algorithm. Therefore, the scheme accepts (G, x) if and only if T_x is a minimum spanning tree. □

Chapter 6

Impact of interactivity

In this chapter, we build a hierarchy of classes, on the top of deterministic decision and proof-labelling schemes. This is the distributed decision analogue of the polynomial hierarchy in centralized computing. While the three first chapters of this thesis were focused on diverse aspects of proof-labelling schemes (with theorems about specific languages, proof sizes etc.), this chapter is more structural, with a focus on complexity classes.

This chapter is based on the paper [41], which is joint work with Pierre Fraigniaud and Juho Hirvonen. It corresponds to Subsection 1.3.4 in the introduction of the thesis.

6.1 Introduction

Motivation and related work. As said in the introduction, it is possible to do an analogy between (centralized) theory of complexity and distributed decision: the class P corresponds to the set of languages that can be decided deterministically, and the class NP can correspond to languages that can be recognized by proof-labelling schemes with proofs of size $O(\log n)$. Considering $O(\log n)$ as the natural threshold was an idea supported in [67], and we give further arguments in that direction at the end of the introduction. This chapter is about building a hierarchy of classes of distributed decision, and studying the problems inside these classes.

One of the lines of research that motivated us to define a local hierarchy is the study of *distributed graph automata*. In particular, [97] recently proved that an analogue of the polynomial hierarchy, where sequential polynomial-time computation is replaced by distributed local computation, turns out to coincide with the logic called MSO. However, while this result is important for our understanding of the computational power of finite automata, the model does not quite fit with the standard model of distributed computing aiming at capturing the power of large-scale computer networks (see, e.g., [95]). Indeed, on the one hand, the model in [97] is somewhat weaker than desired, by assuming a finite-state automata at each node instead of a Turing machine, and by assuming anonymous computation instead of the presence of unique node identities. On the other hand, the very same model is also stronger than the standard model, by assuming a decision-making mechanism based on an arbitrary mapping from the collection of all node states to $\{\text{true}, \text{false}\}$. Instead, the classical distributed decision mechanism we have defined is based on the logical conjunction of the individual decisions.

Another hierarchy is defined in [54], where the authors investigated the local hierarchy in which the certificates must not depend on the identity-assignment to the nodes. Under such *identity-oblivious* certificates, there are distributed languages outside Σ_1. However, all languages are in the probabilistic version of Σ_1, that is, in Σ_1 where the correctness of the verification is only stochastically guaranteed with constant probability. Moreover, the local hierarchy collapses to Π_2 [10]. In [52], it is proved that Σ_1 is exactly captured by the set of distributed languages that are closed under lift. (A configuration (G', x') is a t-lift of a configuration (G, x) if there is an input-preserving mapping from $V(G')$ to $V(G)$ which preserves the t-neighbourhood of the nodes in these graphs). Interestingly, in the same framework as [54] but where the decision function is a global interpretation of the all the individual outputs, instead of the logical conjunction of individual boolean outputs, [4, 5] proved that the local hierarchy collapses to Σ_1. Also, in the same framework as [54], but where the certificates may depend on the identity assignment, all distributed languages are in Σ_1 (see [82]).

Specific definitions. We define an infinite hierarchy $\{(\Sigma_k)_{k\geq 0}, (\Pi_k)_{k\geq 0}\}$ of classes. Informally, each class can be defined by a game between two players, called the *prover* and the *disprover*. Both players are given a language $\mathcal{L}$ and an instance (G, x). In Σ_k (resp., Π_k), with $k > 0$, the prover (resp., disprover) goes first, and assigns an $O(\log n)$-bit certificate to each node. Then, the players alternate, assigning $O(\log n)$-bit certificates to each node in turn, until k certificatess $c_1, c_2, \dots, c_k$ are assigned. The fact of changing the speaker is called an *alternation.* A language $\mathcal{L}$ is in the corresponding class if there is a local algorithm A, and a prover-disprover pair such that, given (G, x), for every set of labels that the disprover assigns, the prover can always assign labels such that A accepts if and only if $(G, x) \in L$. That is, if $(G, x) \notin L$, then the disprover can always force some node to reject, whatever the prover does. Such a combination of a local algorithm A and a prover-disprover pair is called a *decision protocol* for L in the corresponding class. More formally:

Definition 6.1. *Let $\Sigma_0 = \Pi_0 = \text{LD}$, and, for $k > 0$, let Σ_k be the set of languages L for which there exists $\alpha \geq 0$, and a local algorithm A such that*

$$(G, x) \in L \iff \exists c_1, \forall c_2, \dots, \mathrm{Q}\, c_k, A(G, x, c_1, c_2, \dots, c_k) = 1,$$

where Q is the existential (resp., universal) quantifier if k is odd (resp., even), and every label $c_i \in \{0,1\}^*$ is of size at most $\alpha \log n$ in n-node graphs. The class Π_k is defined similarly, except that the acceptance condition is:

$$(G, x) \in L \iff \forall c_1, \exists c_2, \dots, \mathrm{Q}\, c_k, A(G, x, c_1, c_2, \dots, c_k) = 1.$$

Note that the class of problems that can be solved deterministically is $\Sigma_0 = \Pi_0$ (also known as LD), and that the class of problems that have a proof-labelling scheme with proofs of size $O(\log n)$ is the level Σ_1.

For both Σ_k and Π_k, as in proof-labelling scheme before, the equivalence should hold *for every identity-assignment* to the nodes with identities in $[1, N]$, where N is a fixed function polynomial in n. Indeed, the membership of an instance (G, x) to a language is independent of the identities given to the nodes. On the other hand, the labels given by the prover, and by the disprover may well depend on the actual identities of the nodes in the graph where the decision algorithm A is run.

In the remaining, a protocol is said to be *interactive* if there are several provers, and non-interactive if there is only one prover. The study of *interactivity* is the study of the impact of changing from the non-interactive to the interactive setting.

The last section of this chapter deals with a hierarchy in communication complexity, that we will defined at the beginning of the section.

About the logarithmic certificate size. In this chapter we have assumed that every certificate is on $O(\log n)$ bits at each node in an n-node graph. Indeed, this assumption naturally extends the class LogLCP in [66], and labels on $O(\log n)$ bits fits with the classical $\mathsf{CONGEST}$ model for distributed computation [95]. In this paragraph, we briefly discuss the impact of considering smaller and larger labels.

Regarding smaller labels, remember that in Chapter 4 we proved that deciding whether there is at least one leader in the network requires a global certificate on $\Omega(n \log n)$ bits. As we have shown, reversing a distributed decision corresponds to certifying that at least one node rejects. It follows that local certificates on $\Omega(\log n)$ bits are required for reversing a distributed decision. This is an additional motivation for considering certificates on at least $\Omega(\log n)$ bits for the hierarchy, as otherwise establishing connections between the classes and their complement would be cumbersome.

Regarding larger labels, a natural idea consists in considering labels on $O(\text{polylog}\, n)$ bits. Indeed, with such label size, one could combine $O(\text{polylog } n)$ labels into one label, and still remain in the same class. However, this ability to combine a large number of labels does not appear to be crucial in distributed decision, and we did not identify scenarios for which this ability would help. In fact, the only non-artificial problem that would be impacted by labels on $O(\text{polylog } n)$ bits instead of labels on $O(\log n)$ bits is MST. Indeed, MST is in Σ_1 only if one allows certificates as large as $\Omega(\log^2 n)$ bits, while it stands in Π_2 with labels on $O(\log n)$ bits. Moreover, most of our separation results can be extended to labels on $O(\text{polylog } sn)$ bits. In particular, it is worth noticing that the existence of a language L outside LH holds as long as the labels are on $o(n)$ bits.

6.2 Structural results

6.2.1 The odd-even collapsing, and the Λ_k-hierarchy

Interestingly, the ending universal quantifier in both Σ_{2k} and Π_{2k+1} does not help. The class Π_1 turns out to be just slightly stronger than $\textsc{ld}$. Specifically, we prove the following result.

Theorem 6.1. *For every $k \geq 1$, $\Sigma_{2k} = \Sigma_{2k-1}$ and $\Pi_{2k+1} = \Pi_{2k}$. Moreover,* $\textsc{ld} \subseteq \Pi_1 \subseteq \textsc{ld}^{\#\text{node}}$*, where* $\textsc{ld}^{\#\text{node}}$ *is* $\textsc{ld}$ *enhanced with access to an oracle providing each node with the number of nodes in the graph.*

Proof. We first show that an existential quantification on labels of size $O(\log n)$ bit is sufficient to provide the nodes with the exact size of the graph.

Claim 6.1. *Let $L = \{(G,x) :$ for every $v \in V(G)$, $x(v) = |V(G)|\}$. We have $L \in \Sigma_1$.*

To establish the claim, we use a certified spanning tree, as described in the proof-labelling scheme toolbox (Section 1.2). More specifically, on a legal instance, let T be any spanning tree of G, and root T at an arbitrary node r. At node v, let us set certificate $c(v) = (\text{ID}(r), p(v), s(v))$ where $p(v)$ is the identity of the parent of v in T, and $s(v)$ is the size of the subtree of T rooted at v. The verification proceeds as follows: each node v checks that it agrees on $\text{ID}(r)$ and $x(v)$ with all its neighbours in the graph, and that $s(v) = 1 + \sum_{w \in p^{-1}(v)} s(w)$ where $p^{-1}(v)$ denotes the set of v's children, i.e., all neighbours w of v such that $p(w) = v$. In addition, the root r checks that $s(r) = x(r)$. If all tests are passed, then the node accepts, otherwise it rejects. It follows that this algorithm accepts if and only if $x(v) = n$ for every node v. This completes the proof of the claim.

We now show how to use Claim 6.1 for eliminating the last universal quantifier in the case of Σ_{2k}, for $k > 0$. Let $L \in \Sigma_{2k}$, and let A be a t-round local algorithm such that:

$$(G,x) \in L \iff \exists c_1, \forall c_2, \ldots, \exists c_{2k-1}, \forall c_{2k}, A(G,x,c_1,c_2,\ldots,c_{2k}) = 1.$$

Recall that all certificates c_i, $i = 1, \ldots, 2k$, are of size at most $\alpha \log n$ for some $\alpha \geq 0$. We construct an algorithm A' that simulates A for a protocol that does not need the last universal quantifier on c_{2k}. On a legal instance, the first certificate assignment c'_1 consists of some correct c_1 for A, with an additional label that encodes a spanning tree x' (rooted at an arbitrary node), and the value of the number of nodes in G. Regarding the remaining labellings, for each c_{2i-1} assigned by the disprover, the prover assigns c_{2i} as in the protocol for A, ignoring the bits padded to c_1 for creating c'_1. After the labellings have been assigned, each node v running A' gathers its radius-t neighbourhood $B_G(v,t)$. Then, it virtually assigns every possible combination of $(\alpha \log n)$-bit labellings $c_{2k}(u)$ to each node $u \in B_G(v,t)$, and simulates A at v to check whether it accepts or rejects with this labelling. If every simulation accepts, then A' accepts at v, else it rejects. Since every nodes generate all possible c_{2k} labellings in its neighbourhood, we get that

$$(G,x) \in L \iff \exists c'_1, \forall c_2, \ldots, \exists c_{2k-1}, A(G,c_1,c_2,\ldots,c_{2k-1}) \text{ accepts},$$

which places L in Σ_{2k-1}.

The proof of $\Pi_{2k+1} = \Pi_{2k}$ is similar by using the first existential quantifier (which appears in second position) to certify the number of nodes in the graph.

The proof of $\Pi_1 \subseteq \textsc{ld}^{\#\text{node}}$ is also similar, since, in $\textsc{ld}^{\#\text{node}}$, the nodes can directly use the value of the number of nodes directly provided by the oracle #node to test all possible $(\alpha \log n)$-bit labellings c_1 at every node. □

A consequence of Theorem 6.1 is that only of the classes Σ_k for odd k, and Π_k for even k, are worth investigating, once one puts aside the particular case of Π_1. Therefore, we define the classes $(\Lambda_k)_{k\geq 0}$ as follows:

Definition 6.2. *For any $k \geq 0$, let* $\Lambda_k = \begin{cases} \Sigma_k & \textit{if } k \textit{ is odd;} \\ \Pi_k & \textit{otherwise.} \end{cases}$

In particular, $\Lambda_0 = \Pi_0 = \textsc{ld}$. By definition, we get $\Lambda_k \subseteq \Lambda_{k+1}$ for every $k \geq 0$, as the distributed Λ_{k+1} protocol can simply ignore the first label to decide a language in Λ_k.

Definition 6.3. *The* local hierarchy *is defined as* $\mathsf{LH} = \cup_{k\geq 0}\Lambda_k$.

Note that, as for all hierarchies with similar flavor, a language L is in LH if and only if there exists $k \geq 0$ such that $L \in \Lambda_k$. That is, the number of alternations is upper bounded by the same k, for all possible instances (G, x).

6.2.2 Complementary classes

We define the complement classes co-Λ_k, for $k \geq 0$, as

$$\text{co-}\Lambda_k = \{L : \bar{L} \in \Lambda_k\}.$$

Note that, due to the asymmetric nature of distributed decision (unanimity is required for global acceptance, while a single rejection is sufficient for global rejection), simply reversing the individual decision of an algorithm deciding L is generally not appropriate for deciding $\bar{L}$. Nevertheless, we show that an additional existential quantifier is sufficient to reverse any decision, implying the following theorem.

Theorem 6.2. *For every $k \geq 0$,* co-$\Lambda_k \subseteq \Lambda_{k+1}$.

Proof. The proof uses a spanning tree certificate to reverse the decision, in a way similar to the proof that the complement of $\textsc{ld}$ is contained in logLCP (i.e., according to our terminology, co-$\Lambda_0 \subseteq \Lambda_1$) due to Göös and Suomela [66]. Let $L \in \Lambda_k$, and let A be a t-round local algorithm deciding $L \in \Lambda_k$ using labels on at most $\alpha \log n$ bits. We construct an algorithm A' which simulates A, but uses an additional label c_{k+1} to reverse the decisions made by A.

Let us assume that k is even (as it will appear clear later, the proof is essentially the same for k odd). We have that

$$(G, x) \in L \iff \forall c_1, \exists c_2, \ldots, \exists c_k, A(G, x, c_1, c_2, \ldots, c_k) = 1,$$

with all labels c_i's of size at most $\alpha \log n$ for some constant $alpha \geq 0$. In Algorithm A', the prover and the disprover essentially switch their roles. From the above, we have

$$(G, x) \notin L \iff \exists c_1, \forall c_2, \ldots, \forall c_k, \exists v \in G, A(G, v, x, c_1, c_2, \ldots, c_k) = 0.$$

The prover for A' always follows the disprover for A, and can always pick labellings $c_1, c_3, \ldots, c_{k-1}$ such that there is a rejecting node if and only if $(G, x) \notin L$. In the protocol for A', the prover sets c_{k+1} to be a spanning tree rooted at one such rejecting node v. Every other node $u \neq v$ simply checks that c_{k+1} constitutes a proper encoding of a spanning tree, and rejects if not. If all nodes $u \neq v$ accept, then c_{k+1} is indeed a proper spanning tree, and it only remains to check that v rejects in A. To this end, the node v designated as the root of the spanning tree encoded by c_{k+1} gathers all labellings in its radius-t neighbourhood, and computes $A(G, x, v, c_1, c_2, \dots, c_k)$. If A rejects at v, we set A' to accept at v, and, otherwise, we set A' to reject at v.

As discussed in Section 1.2, the spanning tree can be encoded using $O(\log n)$ bits. All labellings $c_1, c_2, \dots, c_k$ have size at most $alpha \log n$, therefore all labels of A' are of size at most $\alpha' \log n$ for some $\alpha' \geq c$. The protocol is correct, as a rejecting node exists in A if and only if $(G, x) \notin L$, and A' correctly accepts in this case. If $(G, x) \in L$, then we have that, for every choice the prover can make, the disprover can always choose its labellings so that A accepts. Thus, if the spanning tree c_{k+1} is correct, the root of that tree will indeed detect that it is an accepting node in A, and so reject in A'. □

Corollary 6.1. *For every $k \geq 0$, co-$\Lambda_k \subseteq$ co-Λ_{k+1}, and $\Lambda_k \subseteq$ co-Λ_{k+1}.*

Proof. If $L \in$ co-Λ_k, then, by definition, $\bar{L} \in \Lambda_k$, and thus also $\bar{L} \in \Lambda_{k+1}$, which implies that $L \in$ co-Λ_{k+1}. If $L \in \Lambda_k$, then $\bar{L} \in$ co-Λ_k, and thus, by Theorem 6.2, we get that $\bar{L} \in \Lambda_{k+1}$, which implies that $L \in$ co-Λ_{k+1}. □

The following theorem shows that, for every $k \geq 0$, and every language L in $\Lambda_k \cap$ co-Λ_k, there is an algorithm deciding L such that an instance $(G, x) \in L$ is accepted at all nodes, and an instance $(G, x) \notin L$ is rejected at *all* nodes as well.

Theorem 6.3. *Let $k \geq 1$, and let $L \in \Lambda_k \cap$ co-Λ_k. Then there exists a local algorithm A such that, for every instance (G, x), and for every $v \in V(G)$,*

$$(G, x) \in L \iff \begin{cases} \forall c_1, \exists c_2, \forall c_3, \dots, \exists c_k, A(G, v, x, c_1, \dots, c_k) = 1 & \text{if } k \text{ is even} \\ \exists c_1, \forall c_2, \dots, \exists c_k, A(G, v, x, c_1, \dots, c_k) = 1 & \text{otherwise} \end{cases}$$

Proof. Assume first that k is even. Since $L \in \Lambda_k \cap$ co-Λ_k, there exist two local algorithms B and B' such that

$$(G, x) \in L \iff \forall c_1, \exists c_2, \dots, \exists c_k, \forall v, B(G, v, x, \bar{c}) = 1, \tag{6.1}$$

and

$$(G, x) \notin L \iff \forall c_1, \exists c_2, \dots, \exists c_k, \forall v, B'(G, v, x, \bar{c}) = 1. \tag{6.2}$$

Now we construct the following protocol for deciding L in an unanimous manner. If $(G, x) \in L$, then the prover assigns the labels as in Eq. (6.1). Instead, if $(G, x) \notin L$, then the prover assigns the labels as in Eq. (6.2). In addition, the first bit of c_k tells which algorithm the nodes should use, with 0 for B, and 1 for B'.

Now, the decision algorithm A proceeds as follows. For each node v, if v and all neighbours of v have the same flag B or B', then v simulates B (with the first bit of c_k ignored), and it outputs $B(G, v, x, c_1, \dots, c_k)$. Conversely, if v and all the neighbour of v have the same flag B', then v simulates B', and it outputs $B'(G, v, x, c_1, \dots, c_k)$.

Finally, if the neighbourhood of v contains both flags B and B', then v outputs the value of its own flag (i.e., 0 or 1). In other words, any c_k-labelling with non-consistent flag is rejecting, though not unanimously so.

It remains to check that A behaves correctly when the flag in c_k is consistent. By construction, from Eq. (6.1) and Eq. (6.2), and from the way A uses B and B', we get that the left to right implication in the statement of the theorem is satisfied: there is always a way to label the graph so that *yes*-instances are accepted everywhere, and *no*-instances are rejected everywhere.

For the other direction, let $f(v)$ denote the flag-bit of the label c_k at node v. We consider the two cases of whether $(G,x) \in L$ or not. First, assume that $(G,x) \in L$, but $f(v) = B'$ for all nodes v. Since every node is simulating B', and is reversing its decision, it follows from Eq. (6.2) that

$$(G,x) \in L \iff \exists c_1, \forall c_2, \dots, \forall c_k, \exists v, B'(G,v,x,c_1,c_2,\dots,c_k) = 0.$$

That is, if $(G,x) \in L$, then the disprover can force *some* node to accept, even if all nodes are consistently running the wrong algorithm, B'.

Conversely, assume that $(G,x) \notin L$, but $f(v) = B$ for all nodes v. Similarly to the previous case, since every node is simulating B, it follows from Eq. (6.1) that

$$(G,x) \notin L \iff \exists c_1, \forall c_2, \dots, \forall c_k, \exists v, A(G,v,x,c_1,c_2,\dots,c_k) = 0,$$

That is, if $(G,x) \notin L$, then the disprover can force *some* node to reject.

The case for k odd is similar. □

In Theorem 6.4 in the next section, we shall see several example of languages in $\Lambda_1 \cap$ co-Λ_1, in relation with classical optimization problems on graphs. By Theorem 6.3, all of these languages can be decided unanimously.

6.3 Positive results

In this section, we precisely identify the position of some relevant problems for distributed computing in the local hierarchy LH.

6.3.1 Optimization problems

Given an optimization problem π on graphs (e.g., finding a minimum dominating set), one defines two distinct distributed languages:

- the language $\textsc{adm}_\pi$ is composed of all configurations (G,x) such that x encodes an admissible solution for π in graph G.

- the language $\textsc{opt}_\pi$ is composed of all configurations (G,x) such that x encodes an optimal solution for π in graph G.

The minimum-weight spanning tree (MST) problem, which is one of the most studied problem in the context of network computing [79, 82, 83], is a typical example of optimization problems that we aim at considering in this section. However, many other problems, such as maximum independent set, max-cut, etc., are also of our interest.

We show that, for any optimization problem π, if deciding whether a candidate solution for π is admissible is "easy", and if the objective function for π has some natural additive form, then $\textsc{opt}_\pi \in \text{co-}\Lambda_1$, and thus $\textsc{opt}_\pi \in \Lambda_2$.

Theorem 6.4. *Let π be an optimization problem on graphs. If the following two properties are satisfied:* (a) $\textsc{adm}_\pi \in \Lambda_1 \cap \text{co-}\Lambda_1$*, and* (b) *the value to the objective function for π is the sum, over all nodes, of an individual value at each node which can be computed locally and encoded on $O(\log n)$ bits, then* $\textsc{opt}_\pi \in \text{co-}\Lambda_1$.

Proof. Let us first prove the following fact. Suppose that every node u of a graph $G = (V, E)$ is given a value x_u on $O(\log n)$ bits, and a value s_u also on $O(\log n)$ bits.

Claim 6.2. *Checking whether $s_u = \sum_{v \in V} x_v$ for every node u can be achieved by a Λ_1-algorithm.*

We describe the algorithm, with, once again, certificates based on a spanning tree. Given a (rooted) spanning tree T, every node u is given the certificate for T, along with the weight $\sum_{v \in V(T_u)} x_v$ of its subtree T_u. The node u checks that (1) the spanning tree certificates are locally correct, (2) its value s_u is equal to s_v for each neighbour v, and (3) the given weight of its subtree is the sum of the weights of the subtrees rooted at its children, plus x_u. The root r also checks that the weight of the entire tree is equal to the given value s_r. If one of these properties does not hold at some node, then this node rejects. It follows that every node accepts if and only if $s_u = \sum_{v \in V} x_v$ for every node u.

Note that Claim 6.2, and the *gathering technique* used to establish that claim can be extended to functions different from the sum, such as min or max.

Let π be an optimization problem on graphs satisfying the conditions of the theorem. To prove $\textsc{opt}_\pi \in \text{co-}\Lambda_1$, we show that $\overline{\textsc{opt}_\pi} \in \Lambda_1$. We describe what certificates are assigned to the nodes by the prover, given an instance $(G, x) \in \overline{\textsc{opt}_\pi}$. Note that such instance may satisfy either x is not admissible in G, or x is admissible but not optimal.

- If $(G, x) \notin \textsc{adm}_\pi$, then the prover flags each node with $\bot$, and assigns certificates for proving that $(G, x) \notin \textsc{adm}_\pi$, which is possible thanks to Condition (a).
- Otherwise, i.e., $(G, x) \in \textsc{adm}_\pi \setminus \textsc{opt}_\pi$, the prover flags each node with $\top$, and assigns certificates for proving that $(G, x) \in \textsc{adm}_\pi$ and $(G, x') \in \textsc{adm}_\pi$, where x' is an arbitrary optimal solution. The latter two sets of certificates can be assigned thanks to Condition (a). Finally, the prover assigns certificates using the gathering technique used to establish Claim 6.2 for certifying the values of the objective function for both x and x'.

The nodes then check that all these certificates are consistent, and, in the case with flag $\top$, that indeed the objective function for x' is better than the one for x. If any of these conditions does not hold at some node, then this node rejects. As a consequence, all the nodes accept if and only if $(G,x) \notin \textsc{opt}_\pi$. Thus $\overline{\textsc{opt}_\pi} \in \Lambda_1$. □

Let us give concrete examples of problems satisfying hypotheses (a) and (b). In fact, most classical optimization problems are satisfying these hypotheses, and all the ones typically investigated in the framework of local computing (cf. the survey [103]) do satisfy (a) and (b).

Corollary 6.2. *Let π be one of the following optimization problems: maximum independent set, minimum dominating set, maximum matching, max-cut, or min-cut. Then* $\textsc{opt}_\pi \in$ co-Λ_1.

Proof. In view of Theorem 6.4, it is sufficient to show that Conditions (a) and (b) are satisfied by each problem in this list. Each of the problems maximum independent set, minimum dominating set, and maximum matching has an easy encoding: a bit that has value 1 if the node is in the set, and zero otherwise. These problems satisfies $\textsc{adm}_\pi \in \Lambda_0$, because each node can check that the local condition specifying admissible solutions holds. It follows that Condition (a) is satisfied for all these three optimization problems. The two cut problems are even easier. The input is a bit that describes which part of the cut the node belongs to, and every input is admissible as every partition of the nodes defines a cut.

Regarding Condition (b), the objective function of maximum independent set, as well as of minimum dominating set, is just the sum of a 0-1 function at each node (0 if not in the set, and 1 otherwise). For maximum matching, as well as for both cut problems, the objective function can be defined as the sum, over all nodes, of half the number of edges adjacent to the node that are involved in the solution. □

The following other corollary of Theorem 6.4 deals with two specific optimization problems, namely travelling salesman and MST. The former illustrates a significant difference between the local hierarchy defined from distributed graph automata in [97], and the one in this chapter. Indeed, we show that travelling salesman is at the second level of our hierarchy, while it does not even belong to the graph automata hierarchy (as Hamiltonian cycle is not in MSO).

Let TRAVELLING SALESMAN be the distributed language formed of all configurations (G,x) where G is a weighted graph, and x is an Hamiltonian cycle C in G of minimum weight (i.e., at node u, $x(u)$ is the pair of edges incident to u in C).

Similarly, let MST be the distributed language formed of all configurations (G,x) where G is a weighted graph, and x is a MST T in G (i.e., at node u, $x(u)$ is the parent of u in T). Note that the case of MST is also particularly interesting. Indeed, MST is known to require labels on $\Theta(\log^2 n)$ bits to be certified [79, 83], and thus MST is not in Λ_1. Note also that, for MST, it is possible to trade locality for the size of the certificates, as it was established in [83] that one can use logarithmic certificates to certify MST in a logarithmic number of rounds.

An intersting consequence of Theorem 6.4 is the following.

Corollary 6.3. MST $\in$ co-Λ_1 *and* TRAVELLING SALESMAN $\in$ co-Λ_1 *for weighted graphs with weights bounded by a polynomial in n.*

Proof. As for Corollary 6.2, it is sufficient to prove that both languages satisfy the two conditions of Theorem 6.4. Condition (b) is satisfied for both as their objective function can be defined as the sum, over all nodes, of half the sum of the weights of the edges incident to the node that are involved in the solution (which can be stored on $O(\log n)$ bits as long as all weights have values polynomial in n).

To prove Condition (a), we need to show that checking whether a collection C of edges is an Hamiltonian cycle (resp., is a spanning tree) is in $\Lambda_1 \cap$ co-Λ_1. We already noticed earlier in the chapter that SPANNING TREE $\in \Lambda_1$. To prove that HAMILTONIAN CYCLE $\in \Lambda_1$, we describe a protocol for that language. Given an Hamiltonian cycle C, the prover elects an arbitrary node r of C as a root, picks a spanning tree T rooted at r, and orients C to form a 1-factor (each node has out-degree 1 in C). The certificate at node u is the identity of r, the distance from u to r in C traversed according to the chosen orientation, and the certificate for T. The verification algorithm at node u checks the tree T (including whether u agrees with all its neighbours on the identity of r). Node u also checks that one of its neighbours in C is one hop closer to r, while the other neighbour in C is one hop farther away from r. Node r checks that one of its neighbours in C is at distance 1 from it in C, while the other it at some distance > 1. Also, a node with distance 0 in C checks that it is the root of T. If all these tests are passed, then node u accepts, otherwise it rejects. The check of T guaranties that there is a unique node r. The check of the hop distance along C guaranties that all nodes are on the same cycle. If both checks are satisfies then C is a unique cycle, covering all nodes, and therefore C is an Hamiltonian cycle.

Now, it remains to prove that SPANNING TREE $\in$ co-Λ_1 and HAMILTONIAN CYCLE $\in$ co-Λ_1. Let F be a collection of edges that is not forming a spanning tree of G. The following certificates are assigned to the nodes. If F is not spanning all nodes, then let r be a node not spanned by F, and let T be a spanning tree rooted at r. The certificate of every node u is a pair $(f(u), c(u))$ where the flag $f(u) = 0$, and $c(u)$ is the certificate for T. If F is spanning all nodes, but contains a cycle C, then let r be a node of C, let us orient the edges of C in a consistent manner, and let T be a spanning tree rooted at r. The certificate at node u is a pair $(f(u), c(u))$ where $f(u) = 1$, and $c(u)$ is a certificate for T. In addition, if u belongs to C, then u received as part of its certificate its distance to r in the oriented cycle C. Finally, If F is spanning forest, then let $F = \{T_1, \dots, T_k\}$ be the trees in F, with $k \geq 2$, and root each one at an arbitrary node r_i, $i = 1, \dots, k$. Let T (resp., T') be a spanning tree rooted at r_1 (resp., r_2). For every $i \in \{1, \dots, k\}$, the certificate at node u of T_i is a 4-tuple $(f(u), \text{index}(u), c(u), c'(u))$ where $f(u) = 2$, $\text{index}(u) = i$, and $c(u)$ (resp., $c'(u)$) is the certificate for T (resp., T').

The verification procedure is as follows. All nodes checks that they have the same flag f. A node detecting that flags differ rejects. A node with flag 0 checks the tree certificates, and the root of the tree checks that it is not spanned by F. A node with flag 1 checks the tree certificates, and the root r of the tree checks that it belongs to C (i.e., was given distance 0 on the cycle). The root r also checks that it has one neighbours in C at distance 1, and another neighbour at distance > 1. All other nodes on C check consistency of the distance counter. Finally, a node with flag 2 checks its tree certificates. The root of T checks that it is of index 1, while the root of T' checks that it is of index 2. Moreover, every node checks that its incident edges in F have extremities with same index. In the three cases, if all tests are passed, the node accepts, otherwise it rejects.

By construction, if F is not a spanning tree, then all nodes accepts. Instead, if F is a spanning tree, then certificates with different flags cannot yield all nodes to accept since two adjacent nodes with different flags both reject. A flag 0 cannot yield all nodes to accept because the non spanned node does not exist. Similarly, a flag 1 cannot yield all nodes to accept because the cycle does does not exist, and a flag 2 cannot yield all nodes to accept because there are no two different connected components, and hence no two trees T and T' rooted at nodes with different indexes.

The proof of HAMILTONIAN CYCLE $\in$ co-Λ_1 proceeds similarly. Therefore, both Conditions (a) and (b) are satisfied for both MST and TRAVELLING SALESMAN, and the result follows by Theorem 6.4. □

6.3.2 Non-trivial automorphism

Recall that $\phi : V(G) \to V(G)$ is an automorphism of G if and only if ϕ is a bijection, and, for every two nodes u and v, we have: $\{u, v\} \in E(G) \iff \{\phi(u), \phi(v)\} \in E(G)$. The graph automorphism problem is the problem of testing whether a given graph has a nontrivial automorphism (i.e., an automorphism different from the identity). Let NONTRIVIAL AUTOMORPHISM be the distributed language composed of the (connected) graphs that admit such an automorphism. It is known that this language is maximally hard for locally checkable proofs, in the sense that it requires proofs with size $\Omega(n^2)$ bits [66]. Nevertheless, we prove that this language remains relatively low in the local hierarchy.

Theorem 6.5. NONTRIVIAL AUTOMORPHISM $\in \Lambda_3$.

Proof. The first label c_1 at node u is an integer that is supposed to be the identity of the image of u by a nontrivial automorphism. Let us denote by $\phi : V(G) \to V(G)$ the mapping induced by c_1. We are left with proving that deciding whether a given ϕ is a nontrivial automorphism of G is in Λ_2. Thanks to Theorem 6.2, it is sufficient to prove that this decision can be made in co-Λ_1. Thus let us prove that checking that (G, ϕ) is *not* a nontrivial automorphism is in Λ_1. If ϕ is the identity, then the certificate can just encode a flag with this information, and each node u checks that $\phi(u)$ is equal to its own ID. So assume now that ϕ is distinct from the identity, but is not an automorphism. To certify this, the prover assigns to each node a set of at most four spanning tree certificates, that "broadcast" to all nodes the identity of at most four nodes witnessing that ϕ is not an automorphism. Specifically, if $\phi(u) = \phi(v)$ with $u \neq v$, then the certificates are for three spanning trees, respectively rooted at u, v, and $\phi(u)$, and if $\{u, v\} \in E(G)$ is mapped to $\{\phi(u), \phi(v)\} \notin E(G)$, or $\{u, v\} \notin E(G)$ is mapped to $\{\phi(u), \phi(v)\} \in E(G)$, then the certificates are for four spanning trees, respectively rooted at $u, v, \phi(u)$, and $\phi(v)$. Checking such certificates can be done locally, and thus checking that (G, ϕ) is *not* a nontrivial automorphism is in Λ_1, from which it follows that NONTRIVIAL AUTOMORPHISM $\in \Lambda_3$. □

6.3.3 Problems from the polynomial hierarchy

As the local hierarchy LH is inspired by the polynomial hierarchy, it is natural to ask about the existence of connections between their respective levels. In this section, we show that some connections can indeed be established, for central problems in the polynomial hierarchy. For instance, let $k \geq 0$, and let us consider all (connected) graphs $G = (V, E)$ such that there exists $X \subseteq V$, $|X| \geq k$, such that, for every $S \subseteq X$, there is a cycle C in G containing all vertices in S, but none in $X \setminus S$. Such graphs have Cycle-VC-dimension, $\mathrm{VC}_{cycle}(G)$, at least k. Deciding whether, given G and k, we have $\mathrm{VC}_{cycle}(G) \geq k$ is Σ_3^P-complete [99, 100]. Let CYCLE-VC-DIMENSION be the distributed language composed of all configurations (G, k) such that all nodes of G have the same input k, and $\mathrm{VC}_{cycle}(G) \geq k$.

Theorem 6.6. CYCLE-VC-DIMENSION $\in \Lambda_3$.

Proof. The existence of the set X can be certified setting a flag at each node in X, together with a tree T_X spanning X for proving that $|X| \geq k$. Given $S \subseteq X$, the cycle C can be certified in the same way as the Hamiltonian cycle in the proof of Corollary 6.3. □

Recall that QBF-k-SAT is the problem of whether a formula of the type

$$\exists y_1, \forall y_2, ..., Q y_k \Phi(y_1, ..., y_k),$$

can be satisfied, where the y_i's are sets of literals on (distinct) boolean variables, Φ is formula from propositional logic (we can assume, w.l.o.g., that Φ is in conjunctive normal form), and Q is the universal quantifier if k is even, and the existential quantifier otherwise. The literals in y_i are said to be at the ith level. This problem is complete for the kth level Σ_k^P of PH. It can be rephrased equivalently into a graph problem, by defining the distributed language QBF-SAT$_k$ formed of all configurations (G, x) with $V(G) = C \cup L$, where C is for clauses, and L is for literals, and there is edge between the positive and negative literals of a same variable, as well as an edge between each clause and all the literals appearing in the clause. More precisely, the input $x(u)$ of a node u in G can be of the form (ℓ, i, s) where ℓ stands for "literal", $i \in \{1, \ldots, k\}$ is the level of that literal, and $s \in \{+, -\}$ indicates whether the literal is positive or negative, or of the form (c) where c stands for "clause". There is an edge between (ℓ, i, s) and $(\ell, i, \bar{s})$ for all literals, and there is an edge between each clause node (i.e., labelled (c)) and all the nodes (ℓ, i, s) such that the corresponding literal appears in that clause. We set $(G, x) \in$ QBF-SAT$_k$ if and only if the corresponding formula is in QBF-k-SAT.

Theorem 6.7. QBF-SAT$_k$ $\in \Lambda_k$.

Proof. For a configuration in QBF-SAT$_k$, the certificates are given to the nodes in the natural way, assigning their values to the literals at odd levels. Each literal node can locally check that its value is the opposite of the one given to its negation, and each clause node can locally check that it is linked to at least one literal that has value true. □

6.3.4 Connections to descriptive complexity

The notion of locality is an important subject when considering expressibility of different logics on graphs, as illustrated by the locality result of Schwentick and Barthelmann [102] for first-order logic. It is then a natural question to ask which are the logics that express properties in LH. A first answer was given by Göös and Suomela [66] who proved that all properties expressible in existential-MSO are in logLCP. One can then expect that LH contains MSO, and it is indeed the case. An easy way to prove this fact is to use the recent result of Reiter [97].

Theorem 6.8. $\mathsf{MSO} \in \mathsf{LH}$.

Proof. As we pointed out earlier in the text, distributed graph automata (DGA) are based on a combination of hypotheses, some weaker than the LOCAL model, and other ones stronger. On the one hand, all computation and communication steps in DGA can be simulated in the LOCAL model. On the other hand, the decision mechanism in DGA is stronger than the one typically used in local decision, as far as distributed network computing is concerned. Specifically, in our framework, the nodes output true or false (i.e., 1 or 0), and the instance is accepted if and only if all the outputs are true. That is, the decision mechanism is simply the conjunction of all the outputs, whereas, in DGA, the outputs belong to an arbitrary finite set S, and the decision mechanism is an arbitrary function f from the set $\mathcal{O}$ of the outputs to $\{$accept, reject$\}$. Note that $\mathcal{O}$ is a *set*, and therefore a same output at two different nodes appears only once in $\mathcal{O}$.

Let us consider a language L at level k of the DGA hierarchy. We show that this language is at level at most $k+1$ of LH. Indeed we can run exactly the same protocol as in DGA, but the decision decision mechanism. Nevertheless, the decision mechanism can be simulated with an additional existential quantifier certifying a spanning tree T that is used to gather all the outputs of the nodes produced by the DGA protocol, in a way similar to the one in Claim 6.1: each node checks that its set of outputs is the union of the output sets of its children in T. The root of T stores the entire set $\mathcal{O}$ (which can be done using $O(|S| \log |S|) = O(1)$ bits), computes $f(\mathcal{O})$, and accepts or rejects accordingly. □

6.4 Separation results

From the previous results in this section, we get that the local hierarchy $\mathsf{LH} = \cup_{k \geq 0} \Lambda_k$ has a typical "crossing ladder" as depicted on Figure 6.1.

In addition, we can show that some of the inclusions are strict:

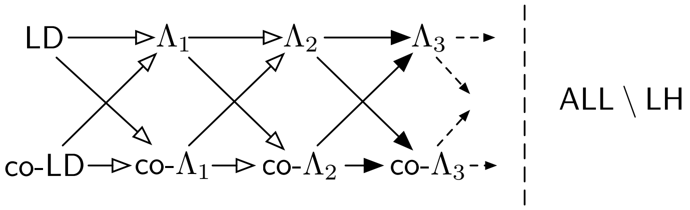


Figure 6.1: *Structure of the local hierarchy. Arrows indicate inclusions, while hollow-headed arrows indicate strict inclusions.*

- Between level 0 and level 1. It is known for long that LD is strictly included in Λ_1 (for instance, 2-COLOURABILITY $\in \Lambda_1 \setminus$ LD). Also, $\Lambda_0 \cup \text{co-}\Lambda_0$ is strictly included in co-Λ_1. Indeed, for instance, NON-3-COLOURABILITY $\in \text{co-}\Lambda_1 \setminus (\Lambda_0 \cup \text{co-}\Lambda_0)$. Therefore, all inclusions between LD and CO-LD and the classes at the first level are strict.

- Between level 1 and level 2. It is known [66] that NON-3-COLOURABILITY $\notin \Lambda_1$, implying that 3-COLOURABILITY $\notin$ co-Λ_1. On the other hand, both languages are in Λ_2, by application of Theorem 6.2. As a consequence, both are also in co-Λ_2. Therefore, all inclusions between the classes at the first and second levels are strict.

For $k \geq 2$, separating the classes at the kth level from the classes at the next level appears to be not straightforward. In particular, all classical counting arguments used to separate the first three levels (i.e., levels 0, 1, and 2) fail. On the other hand, we show that if $\Lambda_k = \Lambda_{k+1}$ for some k, then LH collapses to the kth level.

Theorem 6.9. *If there exists $k \geq 0$ such that $\Lambda_k = \Lambda_{k+1}$, then $\Lambda_i = \Lambda_k$ for all $i > k$, that is,* LH *collapses at the kth level.*

Proof. Let us assume for the purpose of contradiction, that there exists $k \geq 0$ such that $\Lambda_k = \Lambda_{k+1} \neq \Lambda_{k+2}$. Let $L \in \Lambda_{k+2} \setminus \Lambda_{k+1}$. Let us assume that k is odd (as it will appear clear later, the proof for k even is similar). Since $L \in \Lambda_{k+2}$, there exists a local algorithm A such that

$$(G,x) \in L \iff \exists c_1, \forall c_2, \ldots, \exists c_{k+2}, A(G,x,c_1,c_2,\ldots c_{k+2}) = 1.$$

We then define the language $\tilde{L}$ as

$$(G,(x,c)) \in \tilde{L} \iff \forall c_1, \exists c_2, \ldots, \exists c_{k+1} A(G,x,c,c_1,\ldots,c_{k+1}) = 1.$$

By this definition, we get $\tilde{L} \in \Lambda_{k+1}$. Now, since $\Lambda_k = \Lambda_{k+1}$, we get that there exists a local algorithm B such that

$$(G,(x,c)) \in \tilde{L} \iff \exists c_1, \forall c_2, \ldots, \exists c_k, B(G,x,c,c_1,c_2,\ldots,c_k) = 1.$$

On the other hand, by definition, $(G,x) \in L$ if and only if there exists c such that $(G,(x,c)) \in \tilde{L}$. Using Algorithm B, the latter is equivalent to

$$\exists c, \exists c_1, \forall c_2, \ldots, \exists c_k, B(G,(x,c),c_1,c_2,\ldots,c_k) = 1.$$

Now, the two existential quantifiers on c and c_1 can be combined to apply to a single label, from which we get a protocol establishing $L \in \Lambda_k$, a contradiction. The proof for k even is similar. □

Finally, we show that there are languages outside LH. In fact, this result holds even if we restrict ourselves to languages with inputs 0 or 1 on oriented paths, i.e., with identity-assignment where nodes are given consecutive IDs from 1 to n. The result follows from the fact that there are "only" $2^{2^{O(\log n)}}$ different local algorithms for such n-node instances at any fixed level of LH, while there are 2^{2^n} different languages on such instances.

Theorem 6.10. *There exists a language on 0/1-labelled oriented paths that is outside* LH.

Proof. The proof is in two steps. First, using a counting argument, we show that, for any fixed set of parameters, that is, for every level k, every constant α controlling the label size $\alpha \log n$, and every running time t, there is language that cannot be recognized by a protocol with such parameters. Then we combine these languages for various sets of parameters, for building a (Turing-computable) language that cannot be recognized by any protocol of the hierarchy.

Claim 6.3. *Let k, α, and t be non negative integers. There exists an integer n, and a language $L = L(n, k, c, t)$ on 0/1-labelled oriented paths that cannot be recognized by a protocol for Λ_k running in t rounds, and using labels of size at most $\alpha \log n$ bits.*

To establish the claim, notice that an algorithm is simply a mapping from all possible balls (including identifiers, inputs and labels) to binary outputs (accept or reject). On 0/1 inputs, and IDs in $[1, n]$, The number of algorithms for Λ_k running in t rounds using labels of size at most $\alpha \log n$ bits is at most $2^{2^{\beta \log(n)}}$, where $\beta = \beta(k, \alpha, t)$ depends only on k, α, and t. On the other hand, the number of languages on words of size n is exactly 2^{2^n}. Let n be such that $2^{2^{\beta(k,\alpha,t) \log n}} < 2^{2^n}$. By the pigeon-hole principle, there exists a languages that cannot be decided by any algorithm for Λ_k running in t rounds using labels of size at most $\alpha \log n$ bits. This completes the proof of the claim.

Let m be a nonnegative integer, and let $S(n, m)$ be the set of languages on oriented paths with n nodes that cannot be recognized by a protocol with $k = c = t = m$. By Claim 6.3, for every m there exists n such that $S(n, m)$ is not empty. We strengthen this by observing the following two points. First, if $S(n, m)$ is non empty, then for every $m' < m$, the set $S(n, m')$ is non-empty as well, since the protocol for m' could be simulated with parameter m. Second, if $S(n, m)$ is non-empty, then, for every $n' > n$, the set $S(n', m)$ is also non-empty. Indeed, let $L \in S(n, m)$, and let us consider the language L' composed of the set of words in L padded with zeros. If L' has an algorithm, then we could modify this algorithm to get an algorithm recognizing L.

Let us define $\mu(n)$ as the largest integer m such that $S(n, m)$ is non-empty, and $\nu(m)$ the smallest integer n such that $S(n, m)$ is non-empty. Given n and m such that $S(n, m) \neq \emptyset$, let $L(n, m)$ be the smallest language of $S(n, m)$ according to the lexicographic ordering. Finally, let $L = (\cup_{n \geq 1} L(n, \mu(n))$.

We first show that L is a distributed language, i.e., that it is Turing-computable. We describe the algorithm deciding L. The algorithm, given an n-bit string X, computes $\mu(n)$ by enumerating all m's in increasing order, by trying, for each of them, all local algorithms with parameter m, and by checking whether $S(n,m) = \emptyset$. This algorithm eventually finds m such that $S(n,m) = \emptyset$, giving $\mu(n) = m - 1$. Then the algorithm computes $L(n,\mu(n))$, and accepts X if and only if $X \in L(n,\mu(n))$.

We complete the proof by showing that $L \notin \mathsf{LH}$. Suppose, for the sake of contradiction, that $L \in \mathsf{LH}$. Then there exists a local algorithm A deciding $L \in \Lambda_k$, running in t rounds using labels of size at most $\alpha \log n$ bits, for some k, c, and t. Let $m = \max\{k, c, t\}$. We can transform A to decide $L \in \Lambda_m$, running in m rounds using labels of size at most $m \log n$ bits.

Let us consider the restriction L' of L on words of size $\nu(m)$. By definition, $L' = L(\nu(m), \mu(\nu(m)))$, and this language cannot be recognized by a local algorithm with parameter $\mu(\nu(m))$. On the other hand, $\mu(\nu(m)) \geq m$, and therefore L' cannot be recognized by an algorithm of parameter m either. In particular, L' cannot be recognized by A, a contradiction. Therefore $L \notin \mathsf{LH}$. □

6.5 Link with the communication complexity hierarchy

In this section, we show that the local hierarchy is close to a hierarchy in communication complexity. This support the claim that proving a separation between Λ_2 and Λ_3 is hard, as the analogue question in communication complexity has been open for more than thirty years.

6.5.1 Global and communication complexity hierarchies

A hierarchy for global certificates. It is possible to define a hierarchy for global proofs (as defined in Chapter 4) in the same way as we did for local proofs. This will be useful when making the link with communication complexity. More precisely, define Σ_k^G, Π_k^G, and Λ_k^G as previously, except that the labels $c_1, c_2, \ldots, c_k$ are global certificates seen by all nodes.

Communication complexity. We will compare the hierarchies of nondeterministic local decision to the hierarchy of nondeterministic communication complexity defined by Babai et al. [8].

In the communication complexity setting we are given a boolean function f on $2n$ bits. Two entities, Alice and Bob, are each given n-bit vectors x and y, and have to decide if $f(x \cup y) = 1$. They can communicate through a reliable channel and have unlimited computational resources. The measure of complexity is the number of bits Alice and Bob need to communicate in order to decide f. For more details, see for example the book [86].

In nondeterministic communication complexity Alice and Bob have access to nondeterministic advice (we will say that it is given by a *prover*).[1] The cost of a protocol is the sum of the number of bits communicated by Alice and Bob and the number of advice bits given by the prover. This means that messages of Alice and Bob can equivalently be encoded in the advice.

Babai et al. defined a hierarchy of nondeterministic communication complexity [8]. In addition to Alice and Bob we have two players, whom we will call *prover* and *disprover* for consistency, giving nondeterministic advice to Alice and Bob. Prover and disprover will alternate k times and each time give an advice string of $g(n)$ bits. Now we define the class $\Sigma_k^{\text{cc}}(g(n))$ of boolean functions as the set of functions such that there exists an algorithm A for Alice, and an algorithm B for Bob such that if $f \in \Sigma_k^{\text{cc}}(g(n))$, then

$$\forall x, y, \exists c_1, \forall c_2, \ldots, \mathrm{Q}\, c_k A(c_1, c_2, ..., c_k, x) = B(c_1, c_2, ..., c_k, y) = 1 \iff f(x, y) = 1.$$

Again Q denotes the existential quantifier if k is odd and the universal quantifier otherwise. The classes $\Pi_k^{\text{cc}}(g(n))$ are defined similarly, but with the disprover going first. We are particularly interested in this hierarchy when $g(n) = O(\log n)$. Note that in their work, Babai et al. consider the hierarchy for $g(n) = O(\text{poly}(\log n))$ [8].

6.5.2 Connecting local decision and communication complexity

In this subsection we partially formalize the intuition that complexity of local verification is connected to communication complexity. We show that general lower bound proof techniques for nondeterministic local verification will also apply to communication complexity. We then show that if one considers global proofs instead of local ones, the result can be strengthened.

Theorem 6.11. *For every boolean function f, there exists a distributed language L_f such that if $f \in \Sigma_k^{cc}(g(n))$ for odd k or $f \in \Pi_k^{cc}(g(n))$ for even $k \geq 2$, then $L_f \in \Lambda_k(g(n))$.*

The proof is by showing that there exists a family of languages such that a nondeterministic verification scheme can simulate a nondeterministic communication protocol. The theorem partially explains why it is difficult to separate the different levels of the local decision hierarchy — the question is inherently tied to long-standing open questions in communication complexity [8].

Proof of Theorem 6.11. Let f be a boolean function on $2n$ variables. We will construct an infinite family of graphs $\mathcal{G}_n = \big(G(n, t, x, y)\big)_{t,x,y}$ and a related language L_f.

[1] Note that this is a form of non-deterministic communication complexity that is different from the one we considered for lower bounds in Chapter 5

The graph $G_{(n,t,x,y)}$ consists of a path $P_{2t+1} = (v_1, v_2, \ldots, v_{2t+1})$ of length $2t+1$, and two sets of nodes, V_A and V_B of size n. Let us denote $v_A = v_1$ and $v_B = v_{2t+1}$. We add an edge between each $v \in V_A$ and v_A, and an edge between each $u \in V_B$ and v_B. The nodes v_A and v_B are labelled with their respective identities.

Parameters x and y are bit vectors of length n, corresponding to the inputs of players A and B in the communication complexity setting. To encode the input vectors, we use graphs on V_A and V_B, respectively. There are 2^n possible input vectors. We'll define a function ϕ that maps each graph on n nodes to an n-bit vector. Since the encoding of the input cannot depend on the unique identifiers, ϕ must map all graphs of the same isomorphism class to the same vector. Finally, since there are at least $2^{\binom{n}{2}}/n! = \Omega(2^{n^2})$ such graph isomorphism classes, we can find a ϕ such that for all $x \neq y$, we have that $\phi^{-1}(x) \cap \phi^{-1}(y) = \emptyset$.

Given ϕ, x, and y, we can choose two graphs $G_A \in \phi^{-1}(x)$ and $G_B \in \phi^{-1}(y)$, identify the node sets V_A and V_B with $V(G_A)$ and $V(G_B)$, respectively, and add the corresponding edges to the graph $G(n,t,x,y)$. We will use G_A and G_B, respectively, to denote these graphs on node sets V_A and V_B. Nodes v_A and v_B are labelled as special nodes so that the structure of G_A and G_B can be detected. We denote this graph construction by $G(n,t,x,y)$.

Local verification of $\mathcal{G}_f$. A single $O(\log n)$-bit certificate is enough to verify the structure of $G(n,t,x,y)$. It first consists of a spanning tree of P_{2t+1}: node v_A is marked as root, and each node v_i has a pointer to v_{i-1} and a counter i, its distance to the root. It also contains the value n. The nodes v_A and v_B can check that the sizes of the graph G_A and G_B are consistent with this value. They also check that there are no other outgoing edges from G_A and G_B. Nodes v_A and v_B can see all nodes of G_A and G_B, and determine their isomorphism classes, and compute $x = \phi(G_A)$ and $y = \phi(G_B)$.

Deciding L_f. We say that $G \in L_f$ if and only if

1. the structure of G is that of $G(n,t,x,y)$ for some setting of the parameters, and
2. the function f evaluates to 1 on $\phi(G_A) \cup \phi(G_B)$.

Now assume that f is on the kth level of the communication complexity hierarchy with $s = \Omega(\log n)$ bits of nondeterminism. We can use this implied protocol P to solve L_f on the kth level. If the graph structure is correct, the prover and disprover essentially simulate their counterparts from the communication complexity setting, and label *all* nodes on P_{2t+1} as if in P. Then v_A can simulate A and v_B can simulate B, accepting if and only if $f(x,y) = 1$. If the prover tries to deviate from this strategy, nodes can see that its labelling of P_{2t+1} is not constant, and reject. If the disprover tries to deviate, the prover can construct a certificate pointing to this error, and all nodes will accept. □

Global proofs and communication complexity. In the setting of global proofs we can show a slightly stronger theorem.

Theorem 6.12. *For every boolean function f and every $g(n) = \Omega(\log n)$ there exists a distributed language L_f such that $L_f \in \Lambda_k^G(g(n))$, for $k \geq 1$ if and only if f is in the kth level of the communication complexity hierarchy with $O(g(n))$ bits of nondeterminism, in particular $f \in \Sigma_k^{\mathrm{cc}}$ for k odd or $f \in \Pi_k^{\mathrm{cc}}$ for k even.*

In particular, this theorem implies that any collapse in the hierarchy for global certificates implies a collapse in the corresponding communication complexity hierarchy.

Proof of Theorem 6.12. We show that with respect to the language L_f defined in the proof of Theorem 6.11, the communication complexity model and the global verification model can simulate each other.

1. *Communication protocol implies a global verification protocol.* The proof proceeds essentially as in the proof of Theorem 6.11. Using $O(t \log n)$ bits the global certificate can give the list of nodes on the path between v_A and v_B. If a node has degree 2, it must see its own name on this list. Nodes v_A and v_B can again locally verify the structure of G_A and G_B and recover x and y. Finally the prover and disprover follow the communication protocol P on instance (x, y), allowing nodes v_A and v_B to simulate Alice and Bob.

2. *Global verification scheme implies a communication protocol.* Assume there is a kth level global verification scheme with $g(n)$-bit certificates for L_f.

   Alice and Bob will simulate this scheme as follows. Construct a virtual graph $G(x, y)$ consisting of three parts: the nodes v_A and v_B, a path P_{2t+1} of length $2t + 1$ between them, and graphs $H(x)$ and $H(y)$ that are the first elements (in some order) of $\phi^{-1}(x)$ and $\phi^{-1}(y)$, respectively. Finally, all nodes of $H(x)$ are connected to v_A and all nodes of $H(y)$ to v_B. Only Alice will know $H(x)$ and only Bob $H(y)$.

   This graph is in L_f if and only if $f(x, y) = 1$: the structure is exactly as in the definition of L_f.

   Now the nondeterministic prover and disprover can simulate their counterparts in the global verification scheme. Alice and Bob accept if and only if the prover can force all nodes they control to accept. Thus the complexity is bounded by the complexity $g(n)$ of the global verification scheme.

□

Chapter 7

Survey of distributed decision

The objective of this chapter is to survey the recent achievements in the framework of *distributed decision*. It is supposed to be readable independently of the thesis, thus we redefine all the concepts, state the motivation again, and mention the published results of this thesis. Also some general notations are necessary to capture different models, thus the notations differ from the ones we have used in the thesis before. This survey is based on the note [36], published in the Bulletin of the EATCS, and is joint work with Pierre Fraigniaud.

7.1 Introduction

Recall that, in a *construction task*, processes have to collectively compute a valid global state of a distributed system, as a collection of individual states, like, e.g., providing each node of a network with a colour so that to form a proper colouring of that network. Instead, in a *decision task*, processes have to collectively check whether a given global state of a distributed system is valid or not, like, e.g., checking whether a given colouring of the nodes of a network is proper [47]. In general, a typical application of distributed decision is checking the validity of outputs produced by the processes w.r.t. a construction task that they were supposed to solved. This applies to various settings, including randomized algorithms as well as algorithms subject to any kind of faults susceptible to corrupt the memory of the processes.

The global verdict on the legality of the system state is obtained as an aggregate of individual opinions produced by all processes. Typically, each process opinion is a single bit (i.e., *accept* or *reject*) expressing whether the system state looks legal or illegal from the perspective of the process, and the global verdict is the *logical conjunction* of these bits. Note that this mechanisms reflects both decision procedures in which the individual opinions of the processes are collected by some centralized entity, and decision procedures where any process detecting some inconsistency in the system raises an alarm and/or launches a recovery procedure, in absence of any central entity. We will also briefly consider less common procedures where each process can send some limited information about its environment in the system, and a central authority gathers the information provided by the processes to forge its verdict about the legality of the whole system state.

The difficulty of distributed decision arises when the processes cannot obtain a global perspective of the system, which is typically the case if one insists on some form of locality in networks, or if the processes are asynchronous and subject to failures. In such frameworks, not all boolean predicates on distributed systems can be checked in a distributed manner, and one of the main issue of distributed decision is to characterize the predicates that can be distributedly checked, and at which cost. For predicates that cannot be checked, or for which checking is too costly, the system can be enhanced by providing processes with *certificates*, with the objective to help these processes for expressing their individual opinions. Such certificates could be produced by an external entity, but they might also well be produced by the processes themselves during a pre-computation phase. One typical framework in which the latter scenario finds application is self-stabilization. Indeed, a self-stabilizing algorithm may produce, together with its distributed output, a distributed certificate that this output is correct. Of course, the certificates are also corruptible, and thus not trustable. Hence, the checking procedure must involve a distributed verification algorithm in charge of verifying the collection of pairs (output, certificate) produced by all the processes. Some even more elaborated mechanisms for checking the legality of distributed system states are considered in the literature, and we survey such mechanisms as well.

We consider the most classical distributed computing models, including synchronous distributed network computing [95]. In this setting, processes are nodes of a graph representing a network. They all execute the same algorithm, they are fault-free, and they are provided with distinct identities in some ID-space (which can be bounded or not). All processes start simultaneously, and computation proceeds in synchronous rounds. At each round, every process exchanges messages with its neighbouring processes in the network, and performs individual computation. The volume of communication each node can transmit and receive on each of its links at each round might be bounded or not. The CONGEST model typically assumes that at most $O(\log n)$ bits can be transferred along each link at each round in n-node networks. (In this case, the ID-space is supposed to be polynomially bounded as a function of the network size). Instead the LOCAL model does not limit the amount of information that can be transmitted along each link at each round. So, a t-round algorithm $\mathcal{A}$ in the LOCAL model can be transformed into another algorithm $\mathcal{B}$ in which every node first collects all data available in the ball of radius t around it, and, second, simulate $\mathcal{A}$ locally without communication.

We also consider other models like asynchronous distributed shared-memory computing [6]. In this setting, every process has access to a global memory shared by all processes. Every process accesses this memory via atomic read and write instructions. The memory is composed of registers, and each process is allocated a set of private registers. Every process can read all the registers, but can only write in its own registers. Processes are given distinct identities in $[n] = \{1, \ldots, n\}$ for n-process systems. They runs asynchronously, and are subject to crashes. A process that crashes stops taking steps. An arbitrary large number of processes can crash. Hence, an algorithm must never include instructions leading a process to wait for actions by another process, as the latter process can crash. This model is thus often referred to as the WAIT-FREE model.

Finally, we briefly consider other models, including mobile computing [45], mostly in the fully-synchronous FSYNC model in graphs (where all mobile agents perform in lock-step, moving from nodes to adjacent nodes in a network), and distributed quantum computing (where processes have access to intricate variables).

7.2 Model and definitions

Given a boolean predicate, a distributed decision algorithm is a distributed algorithm in which every process p must eventually output a value

$$\text{opinion}(p) \in \{\text{accept, reject}\}$$

such that the global system state satisfies the given predicate if and only if all processes accept. In other word, the global interpretation of the individual opinions produced by the processes is the logical conjunction of all these opinions:

$$\text{global verdict} = \bigwedge_{p} \text{opinion}(p).$$

Among the earliest references explicitly related to distributed decision, it is worth mentioning [2, 7, 72]. In this section, we describe the general framework of distributed decision, without explicit references to some specific underlying computational model.

The structure of the section is inspired from the structure of complexity classes in sequential complexity theory. Given the "base" class P of languages that are sequentially decidable by a Turing machine in time polynomial in the size of the input, the classes NP (for non-deterministic polynomial time) and BPP (for bounded probability polynomial time) are defined, as well as the classes Σ_k^{P} and Π_k^{P}, $k \geq 0$, of the polynomial hierarchy. In this section we assume given an abstract class BC (for *bounded distributed computing*), based on which larger classes can be defined. Such a base class BC could be a *complexity* class like, e.g., the class of graph properties that can be checked in constant time in the LOCAL model, or a *computability* class like, e.g., the class of system properties that can be checked in a shared-memory distributed system subject to crash failures. Given the "base" class BC, we shall define the classes NBC, BPBC, Σ_k^{BC} and Π_k^{BC}, that are to BC what NP, BPP, Σ_k^{P} and Π_k^{P} are to P, respectively.

7.2.1 Distributed languages

A system *configuration* $\mathbf{C}$ is a (partial) description of a distributed system state. For instance, in distributed network computing, a configuration $\mathbf{C}$ is of the form (G, ℓ) where G is a graph, and $\ell : V(G) \to \{0,1\}^*$. Similarly, in shared memory computing, a configuration $\mathbf{C}$ is of the form $\ell : [n] \to \{0,1\}^*$ where n is the number of processes. The function ℓ is called *labelling* function, and $\ell(v)$ the *label* of v, which can be any arbitrary bit string. In the context of distributed decision, the label of a process is the input of that process.

For instance, the label of a node in a processor network can be a colour, and the label of a process in a shared memory system can be a status like "elected" or "defeated". Note that, in both examples, a configuration is oblivious to the content of the shared memory and/or to the message in transit. The labelling function ℓ may not describe the full state of each process, but only the content of some specific variables.

Definition 7.1. *Given a distributed computing model, a* distributed language *is a Turing-computable set of configurations compatible with this model.*

For instance, in the framework of network computing,

$$\text{PROPER-COLOURING} = \{(G, \ell) : \forall \{u, v\} \in E(G), \ell(u) \neq \ell(v)\}$$

is the distributed language composed of all networks with a proper colouring of their nodes (the label $\ell(v)$ of node v is its colour). Similarly, in the framework of crash-prone shared-memory computing,

$$\text{AGREEMENT} = \{\ell : \exists y \in \{0,1\}^*, \forall i \in [n], \ell(i) = y \text{ or } \ell(i) = \bot\}$$

is the distributed language composed of all systems where agreement between the non-crashed processes is achieved (the label of process p_i is $\ell(i)$, and the symbol $\bot$ refers to the scenario in which process p_i crashed).

For a fixed distributed language $\mathcal{L}$, a configuration in $\mathcal{L}$ is said to be *legal*, and a configuration not in $\mathcal{L}$ is said to be *illegal*. Any distributed language $\mathcal{L}$ defines a *construction* task, in which every process must compute a label such that the collection of labels outputted by the processes form a legal configuration for $\mathcal{L}$. In the following, we are mostly interested in *decision* tasks, where the labels of the nodes are given, and the processes must collectively check whether these labels form a legal configuration.

Notation. Given a system configuration $\mathbf{C}$ with respect to some distributed computing model, we denote by $V(\mathbf{C})$ the set of all computing entities (a.k.a. processes) in $\mathbf{C}$. This notation reflects the fact that, in the following, the set of processes will most often be identified as the vertex-set $V(G)$ of a graph G

7.2.2 Distributed decision

Given a distributed computing model, let us define some *bounded computing* class BC as a class of distributed languages that can be decided with a distributed algorithm $\mathcal{A}$ using a bounded amount of resources. Such an algorithm $\mathcal{A}$ is said to be *bounded*. What is meant by "resource" depends on the computing model. In most of the models investigated in this chapter, the resource of interest is the number of rounds (as in the LOCAL and $\mathsf{CONGEST}$ models), or the number of read/write operations (as in the $\mathsf{WAIT\text{-}FREE}$ model). A distributed language $\mathcal{L}$ is in BC if and only if there exists a bounded algorithm $\mathcal{A}$ such that, for any input configuration $\mathbf{C}$, the algorithm $\mathcal{A}$ outputs $\mathcal{A}(\mathbf{C}, v)$ at each process v, and this output satisfies:

$$\mathbf{C} \in \mathcal{L} \iff \text{for every } v \in V(\mathbf{C}), \mathcal{A}(\mathbf{C}, v) = \text{accept}. \tag{7.1}$$

That is, for every $\mathbf{C} \in \mathcal{L}$, running $\mathcal{A}$ on $\mathbf{C}$ results in all processes accepting $\mathbf{C}$. Instead, for every $\mathbf{C} \notin \mathcal{L}$, running $\mathcal{A}$ on $\mathbf{C}$ results in at least one process rejecting $\mathbf{C}$.

Example. In the context of network computing, PROPER-COLOURING can be decided in one round, by having each node merely comparing its colour with the ones of its neighbours, and accepting if and only if its colour is different from all these colours. Similarly, in the context of shared-memory computing, AGREEMENT can be decided by having each node performing just one read/write operation, accepting if and only if all labels different from $\perp$ observed in memory are identical. In other words, assuming that BC is a network computing class bounding algorithms to perform in a constant number of rounds, we have

$$\text{PROPER-COLOURING} \in \mathsf{BC}$$

for any model allowing each process to send its colour to all its neighbours in a constant number of rounds, like, e.g., the LOCAL model. Similarly, assuming that BC is a shared-memory computing class bounding algorithms to perform in a constant number of read/write operations, we have

$$\text{AGREEMENT} \in \mathsf{BC}.$$

Notation. In the following, Eq. (7.1) will often be abbreviated to

$$\mathbf{C} \in \mathcal{L} \iff \mathcal{A}(\mathbf{C}) = \text{accept}$$

in the sense that $\mathcal{A}$ accepts if and only if each of the processes accepts.

Note that the rule of distributed decision, i.e., the logical conjunction of the individual boolean outputs of the processes is not symmetric. For instance, deciding whether a graph is properly coloured can be done locally, while deciding whether a graph is *not* properly coloured may require long-distance communications. On the other hand, asking for other rules, like unanimous decision (where all processes must reject an illegal configuration) or even just majority decision, would require long-distance communications for most classical decision problems.

7.2.3 Probabilistic distributed decision

The bounded computing class BC is a base class upon which other classes can be defined. Given $p, q \in [0,1]$, we define the class $\mathsf{BPBC}(p,q)$, for *bounded probability bounded computing*, as the class of all distributed languages $\mathcal{L}$ for which there exists a randomized bounded algorithm $\mathcal{A}$ such that, for every configuration $\mathbf{C}$,

$$\begin{cases} \mathbf{C} \in \mathcal{L} \;\Rightarrow\; \Pr[\mathcal{A}(\mathbf{C}) = \text{accept}] \geq p; \\ \mathbf{C} \notin \mathcal{L} \;\Rightarrow\; \Pr[\mathcal{A}(\mathbf{C}) = \text{reject}] \geq q. \end{cases} \tag{7.2}$$

Such an algorithm $\mathcal{A}$ is called a (p,q)-decider for $\mathcal{L}$. Note that, as opposed to the class BPP of complexity theory, the parameters p and q are not arbitrary, in the sense that boosting the probability of success of a (p,q)-decider in order to get a (p',q')-decider with $p' > p$ and $q' > q$ is not always possible. Indeed, if $\mathcal{A}$ is repeated many times on an illegal instance, say k times, it may well be the case that each node will reject at most once during the k repetitions, because, at each iteration of $\mathcal{A}$, rejection could come from a different node. As a consequence, classical boosting techniques based on repetition and taking majority do not necessarily apply.

Example. Let us consider the following distributed language, where each process can be labeled either white or black, i.e., $\ell : V(\mathbf{C}) \to \{\circ, \bullet\}$:

$$\textsc{amos} = \{\ell : |\{v \in V(\mathbf{C}) : \ell(v) = \bullet\}| \leq 1\}.$$

Here, AMOS stands for "at most one selected", where a node v is selected if $\ell(v) = \bullet$. There is a trivial (p, q)-decider for AMOS as long as $p^2 + q \leq 1$, which works as follows. Every node v with $\ell(v) = \circ$ accepts (with probability 1). A node v with $\ell(v) = \bullet$ accepts with probability p, and rejects with probability $1 - p$. If $\mathbf{C} \in$ AMOS, then Pr[all nodes accept $\mathbf{C}$] $\geq p$. If $\mathbf{C} \notin$ AMOS, then Pr[at least one node rejects $\mathbf{C}$] $\geq 1 - p^2 \geq q$.

7.2.4 Distributed verification

Given a bounded computing class BC, we describe the class NBC, which is to BC what NP is to P in complexity theory. We define the class NBC, for *non-deterministic bounded computing*, as the class of all distributed languages $\mathcal{L}$ such that there exists a bounded algorithm $\mathcal{A}$ satisfying that, for every configuration $\mathbf{C}$,

$$\mathbf{C} \in \mathcal{L} \iff \exists c : \mathcal{A}(\mathbf{C}, c) = \text{accept} \tag{7.3}$$

where

$$c : V(\mathbf{C}) \to \{0, 1\}^*.$$

The function c is called the *certifying* function. It assigns a certificate to every process, and the certificates do not need to be identical. Note that the certificate $c(v)$ of process v must not be mistaken with the label $\ell(v)$ of that process.

The bounded algorithm $\mathcal{A}$ is also known as a *verification* algorithm for $\mathcal{L}$, as it verifies a given proof c, which is supposed to certify that $\mathbf{C} \in \mathcal{L}$. At each process $v \in V(\mathbf{C})$, the verification algorithm takes as input the pair $(\ell(v), c(v))$. Note that the appropriate certificate c leading to accept a configuration $\mathbf{C} \in \mathcal{L}$ may depend on the given configuration $\mathbf{C}$. However, for $\mathbf{C} \notin \mathcal{L}$, the verification algorithm $\mathcal{A}$ must systematically guaranty that at least one process rejects, whatever the given certificate function is.

Alternatively, one can interpret Eq. (7.3) as a game between a *prover* which, for every configuration $\mathbf{C}$, assigns a certificate $c(v)$ to each process $v \in V(\mathbf{C})$, and a *verifier* which checks that the certificates assigned by the prover collectively form a *proof* that $\mathbf{C} \in \mathcal{L}$. For a legal configuration (i.e., a configuration in $\mathcal{L}$) the prover must be able to produce a distributed proof leading the distributed verifier to accept, while, for an illegal configuration, the verifier must reject in at least one node whatever the proof provided by the prover is.

Example. Let us consider the distributed language

$$\textsc{acyclic} = \{(G, \ell) : G \text{ has no cycles}\}$$

in the context of network computing. Note that ACYCLIC cannot be decided locally, even in the LOCAL model. However, ACYCLIC can be verified in just one round. If G

is acyclic, i.e., G is a forest, then let us select an arbitrary node in each tree of G, and call it a root. Next, let us assign to each node $u \in V(G)$ the certificate $c(u)$ equal to its distance to the root of its tree. The verification algorithm $\mathcal{A}$ then proceeds at every node u as follows. Node u exchanges its certificate with the ones of it neighbours, and checks that it has a unique neighbour v satisfying $c(v) = c(u) - 1$, and all the other neighbours $w \neq v$ satisfying $c(w) = c(u) + 1$. (If u has $c(u) = 0$, then it checks that all its neighbours w have $c(w) = 1$). If all tests are passed, then u accepts, else it rejects. If G is a acyclic, then, by construction, the verification accepts at all nodes. Instead, if G has a cycle, then, for every setting of the certifying function, some inconsistency will be detected by at least one node of the cycle, which leads this node to reject. Hence

$$\textsc{acyclic} \in \mathsf{NBC}$$

where BC bounds the number of rounds, for every distributed computing model allowing every node to exchange $O(\log n)$ bits along each of its incident edges at every round, like, e.g., the $\mathsf{CONGEST}$ model.

Notation. For any function $f : \mathbb{N} \to \mathbb{N}$, we define $\mathsf{NBC}(f)$ as the class NBC where the certificates are bounded to be on at most $f(n)$ bits in n-node networks. For $f \in \Theta(\log n)$, $\mathsf{NBC}(f)$ is rather denoted by log-NBC.

7.2.5 Distributed decision hierarchy

In the same way the polynomial hierarchy PH is built upon P using alternating universal and existential quantifiers, one can define a hierarchy built upon base class BC. Given a class BC for some distributed computing model, we define the *distributed decision hierarchy* $\mathsf{DH}^{\mathsf{BC}}$ as follows. We set $\Sigma_0^{\mathsf{BC}} = \Pi_0^{\mathsf{BC}} = \mathsf{BC}$, and, for $k \geq 1$, we set Σ_k^{BC} as the class of all distributed languages $\mathcal{L}$ such that there exists a bounded algorithm $\mathcal{A}$ satisfying that, for every configuration $\mathbf{C}$,

$$\mathbf{C} \in \mathcal{L} \iff \exists c_1 \, \forall c_2 \, \exists c_3 \ldots Q c_k : \mathcal{A}(\mathbf{C}, c_1, \ldots, c_k) = \text{accept}$$

where, for every $i \in \{1, \ldots, k\}$, $c_i : V(\mathbf{C}) \to \{0,1\}^*$, and Q is the universal quantifier if k is even, and the existential one otherwise. The class Π_k^{BC} is defined similarly, by having a universal quantifier as first quantifier, as opposed to an existential one as in Σ_k^{BC}. The c_i's are called *certifying* functions. In particular, we have

$$\mathsf{NBC} = \Sigma_1^{\mathsf{BC}}.$$

Finally, we define

$$\mathsf{DH}^{\mathsf{BC}} = (\cup_{k \geq 0} \, \Sigma_k^{\mathsf{BC}}) \cup (\cup_{k \geq 0} \, \Pi_k^{\mathsf{BC}}).$$

As for NBC, a class Σ_k^{BC} or Π_k^{BC} can be viewed as a game between a prover (playing the existential quantifiers), a disprover (playing the universal quantifiers), and a verifier (running a verification algorithm $\mathcal{A}$).

Example. Let us consider the distributed language

$$\text{VERTEX-COVER} = \big\{(G,\ell) : \{v \in V(G) : \ell(v) = 1\} \text{ is a minimum vertex cover}\big\}$$

in the context of network computing. We show that VERTEX-COVER $\in \Pi_2^{\mathsf{BC}}$, that is, there exists a bounded distributed algorithm $\mathcal{A}$ such that

$$(G,\ell) \in \text{VERTEX-COVER} \iff \forall c_1\, \exists c_2 : \mathcal{A}(G,\ell,c_1,c_2) = \text{accept}$$

where BC is any network computing class bounding algorithms to perform in a constant number of rounds. For any configuration (G,ℓ), the disprover tries to provide a vertex cover $c_1 : V(G) \to \{0,1\}$ of size smaller than the solution ℓ, i.e., $|\{v \in V(G) : c_1(v) = 1\}| < |\{v \in V(G) : \ell(v) = 1\}|$. On a legal configuration (G,ℓ), the prover then reacts by providing each node v with a certificates $c_2(v)$ such that the c_2-certificates collectively encode a spanning tree (and its proof) aiming at demonstrating that there is an error in c_1 (like c_1 is actually not smaller than ℓ, or c_1 is not covering some edge, etc.). It follows that

$$\text{VERTEX-COVER} \in \Pi_2^{\mathsf{BC}}$$

for any model allowing each process to exchange $O(\log n)$-bits messages with its neighbours in a constant number of rounds, like, e.g., the $\mathsf{CONGEST}$ model.

Notation. Similarly to the class NBC, for any function $f : \mathbb{N} \to \mathbb{N}$, we define $\Sigma_k^{\mathsf{BC}}(f)$ (resp., $\Pi_k^{\mathsf{BC}}(f)$) as the class Σ_k^{BC} (resp., Π_k^{BC}) where all certificates are bounded to be on at most $f(n)$ bits in n-node networks. For $f \in \Theta(\log n)$, these classes are denoted by log-Σ_k^{BC} and log-Π_k^{BC}, respectively. The classes $\mathsf{DH}^{\mathsf{BC}}(f)$ and log-$\mathsf{DH}^{\mathsf{BC}}$ are defined similarly.

7.3 Distributed decision in networks

In this section, we focus on languages defined as collections of configurations of the form (G,ℓ) where G is a simple connected n-node graph, and $\ell : V(G) \to \{0,1\}^*$ is a labelling function assigning to every node v a label $\ell(v)$. Recall that an algorithm $\mathcal{A}$ is *deciding* a distributed language $\mathcal{L}$ if and only if, for every configuration (G,ℓ),

$$(G,\ell) \in \mathcal{L} \iff \mathcal{A}(G,\ell) \text{ accepts at all nodes.}$$

7.3.1 LOCAL model

Local distributed decision (LD and BPLD)

In their seminal paper [89], Naor and Stockmeyer define the class LCL, for *locally checkable labellings*. Let $\Delta \geq 0$, $k \geq 0$, and $t \geq 0$, and let $\mathcal{B}$ be a set of balls of radius at most t with nodes of degree at most Δ, labeled by labels in $[k]$. Note that $\mathcal{B}$ is finite. Such a set $\mathcal{B}$ defines the language $\mathcal{L}$ consisting of all configurations (G,ℓ)

where G is a graph with maximum degree Δ, and $\ell : V(G) \to [k]$, such that all balls of radius t in (G, ℓ) belong to $\mathcal{B}$. The set $\mathcal{B}$ is called the set of *good* balls for $\mathcal{L}$. LCL is the class of languages that can be defined by a set of good balls, for some parameters Δ, k, and t. For instance the set of k-coloured graphs with maximum degree Δ is a language in LCL. The good balls of this LCL language are simply the balls of radius 1 where the center node is labeled with a colour different from all the colours of its neighbours.

A series of results were achieved in [89] about LCL languages. In particular, it is Turing-undecidable whether any given $\mathcal{L} \in \mathsf{LCL}$ has a construction algorithm running in $O(1)$ rounds in the LOCAL model. Also, [89] showed that the node IDs play a limited role in the context of LCL languages. Specifically, [89] proves that, for every $r \geq 0$, if a language $\mathcal{L} \in \mathsf{LCL}$ has a r-round construction algorithm, then it has also a r-round *order invariant* construction algorithm, where an algorithm is order invariant if the *relative order* of the node IDs may play a role, but not the actual *values* of these IDs. The assumption $\mathcal{L} \in \mathsf{LCL}$ can actually be discarded, as long as $\mathcal{L}$ remains defined on constant degree graphs with constant labels. That is, [5] proved that, in constant degree graphs, if a language with constant size labels has a r-round construction algorithm, then it has also a r-round order invariant construction algorithm. Last but not least, [89] established that randomization is of little help in the context of LCL languages. Specifically, [89] proves that if a language $\mathcal{L} \in \mathsf{LCL}$ has a randomized Monte-Carlo construction algorithm running in $O(1)$ rounds, then $\mathcal{L}$ also has a deterministic construction algorithm running in $O(1)$ rounds.

The class LD, for *local decision* was defined in [54] as the class of all distributed languages that can be decided in $O(1)$ rounds in the LOCAL model. The class LD is the basic class playing the role of BC in the context of local decision. Hence $\mathsf{LCL} \subseteq \mathsf{LD}$ since the set of good balls of a language in LCL is, by definition, finite. On the other hand, $\mathsf{LCL} \subset \mathsf{LD}$, where the inclusion is strict since LD does not restrict the graphs to be of bounded degree, nor the labels to be of bounded size. Given $p, q \in [0, 1]$, the class $\mathsf{BPLD}(p, q)$, for *bounded probability local decision*, was defined in [54] as the class of languages for which there is a (p, q)-decider running in $O(1)$ rounds in the LOCAL model. For $p^2 + q \leq 1$, $\mathsf{BPLD}(p, q)$ is shown to include languages that cannot be even decided deterministically in $o(n)$ rounds. On the other hand, [54] also establishes a derandomization result, stating that, for $p^2 + q > 1$, if $\mathcal{L} \in \mathsf{BPLD}(p, q)$, then $\mathcal{L} \in \mathsf{LD}$. This results however holds only for languages closed under node deletion, and it is proved in [56] that, for any every $c \geq 2$, there exists a language $\mathcal{L}$ with a (p, q)-decider satisfying $p^c + q > 1$ and running in a single round, which cannot be decided deterministically in $o(\sqrt{n})$ rounds. On the other hand, [56] proves that, for $p^2 + q > 1$, we have $\mathsf{BPLD}(p, q) = \mathsf{LD}$ for all languages restricted on paths.

On the negative side, it was proved in [56] that boosting the probability of success for decision tasks is not always achievable in the distributed setting, by considering the classes

$$\mathsf{BPLD}_k = \bigcup_{p^{1+1/k}+q>1} \mathsf{BPLD}(p, q) \text{ and } \mathsf{BPLD}_\infty = \bigcup_{p+q>1} \mathsf{BPLD}(p, q)$$

for any $k \geq 1$, and proving that, for every $k \geq 1$, $\mathsf{BPLD}_k \subset \mathsf{BPLD}_\infty$, and $\mathsf{BPLD}_k \subset$

BPLD_{k+1}, where all inclusions are strict.

On the positive side, it was proved in [35] that the result in [89] regarding the derandomization of construction algorithms can be generalized from LCL to BPLD. Namely, [35] proves that, for languages on bounded degree graphs and bounded size labels, for every $p > \frac{1}{2}$ and $q > \frac{1}{2}$, if $\mathcal{L} \in \mathsf{BPLD}(p, q)$ has a randomized Monte-Carlo construction algorithm running in $O(1)$ rounds, then $\mathcal{L}$ has also a deterministic construction algorithm running in $O(1)$ rounds.

Identity-oblivious algorithms (LDO)

In the LOCAL model, a distributed algorithm is *identity-oblivious*, or simply *ID-oblivious*, if the outputs of the nodes are not impacted by the identities assigned to the nodes. That is, for any two ID-assignments given to the nodes, the output of every node must be identical in both cases. Note that an *identity-oblivious* algorithm may use the IDs of the nodes (e.g., to distinguish them), but the output must be oblivious to these IDs.

The class LDO, for *local decision oblivious* was defined in [52, 53], as the class of all distributed languages that can be decided in $O(1)$ rounds by an ID-oblivious algorithm in the LOCAL model. The class LDO is the basic class playing the role of BC in the context of ID-oblivious local decision. It is shown in [52] that $\mathsf{LDO} = \mathsf{LD}$ when restricted to languages that are closed under node deletion. However, it is proved in [53] that $\mathsf{LDO} \subset \mathsf{LD}$, where the inclusion is strict. In the language $\mathcal{L} \in \mathsf{LD} \setminus \mathsf{LDO}$ used in [53] to prove the strict inclusion $\mathsf{LDO} \subset \mathsf{LD}$, each node label includes a Turing machine M. Establishing $\mathcal{L} \in \mathsf{LD}$ makes use of an algorithm simulating M at each node, for a number of rounds equal to the identity of the node. Establishing $\mathcal{L} \notin \mathsf{LDO}$ makes use of the fact that an ID-oblivious algorithm can be sequentially simulated, and therefore, if an ID-oblivious algorithm would allow to decide $\mathcal{L}$, then by simulation of this algorithm, there would exist a sequential algorithm for separating the set of Turing machines that halts and output 0 from the set of Turing machines that halts and output 1, which is impossible.

In [52, 59], the power of IDs in local decision is characterized using *oracles*. An oracle is a trustable party with full knowledge of the input, who can provide nodes with information about this input. It is shown in [52] that $\mathsf{LDO} \subseteq \mathsf{LD} \subseteq \mathsf{LDO}^{\#\text{node}}$ where #node is the oracle providing each node with an arbitrary large upper bound on the number of nodes. A *scalar* oracle f returns a list $f(n) = (f_1, \ldots, f_n)$ of n values that are assigned arbitrarily to the n nodes in a one-to-one manner. A scalar oracle f is large if, for any set of k nodes, the largest value provided by f to the nodes in this set grows with k. [59] proved that, for any computable scalar oracle f, we have $\mathsf{LDO}^f = \mathsf{LD}^f$ if and only if f is large, where LD^f (resp., LDO^f) is the class of languages that can be locally decided in $O(1)$ rounds in the LOCAL model by an algorithm (resp., by an ID-oblivious algorithm) which uses the information provided by f available at the nodes.

Anonymous networks

Derandomization results were achieved in [30] in the framework of anonymous network (that is, nodes have no IDs). Namely, for every language $\mathcal{L}$ that can be decided locally in any anonymous network, if there exists a randomized anonymous construction algorithm for $\mathcal{L}$, then there exists a deterministic anonymous construction algorithm for $\mathcal{L}$, provided that the latter is equipped with a 2-hop colouring of the input network. In addition, [31] shows that, in anonymous networks, giving the ability to the nodes of revoking their decisions (i.e., to change it as long as not all the nodes have output) considerably changes the power of the model.

7.3.2 CONGEST model

Decision algorithms

In [76] and [27] the authors consider decision problems such as checking whether a given set of edges forms a spanning tree, checking whether a given set of edges forms a minimum-weight spanning tree (MST), checking various forms of connectivity, etc. All these decision tasks require essentially $\Theta(\sqrt{n} + D)$ rounds (the lower bound is typically obtained using reduction to communication complexity). In particular, [27] proved that checking whether a given set of edges is a spanning tree requires $\Omega(\sqrt{n}+D)$ rounds, which is much more that what is required to construct a spanning tree ($O(D)$ rounds, using a simple breadth-first search). However, [27] proved that, for some other problems (e.g., MST), lower bounds on the round-complexity of the decision task consisting in checking whether a solution is valid yield lower bounds on the round-complexity of the corresponding construction task, and this holds also for the construction of approximate solutions. (In [29], the techniques of [27] are extended to a quantum setting, to achieve similar lower bounds).

Distributed property testing

Very few distributed languages on graphs can be checked locally in the CONGEST model. For instance, even just deciding whether G contains a triangle cannot be done in $O(1)$ rounds in the CONGEST model. *Distributed property testing* is a framework first investigated in [21], and recently formalized in [23]. Let $0 < \epsilon < 1$ be a fixed parameter. Recall that, according to the usual definition borrowed from *property testing* (in the so-called *sparse* model), a graph property P is ϵ-far from being satisfied by an m-edge graph G if applying a sequence of at most ϵm edge-deletions or edge-additions to G cannot result in a graph satisfying P. We say that a distributed algorithm $\mathcal{A}$ is a distributed *testing* algorithm for P if and only if, for any graph G modeling the actual network,

$$\begin{cases} G \text{ satisfies } P \implies \Pr[\mathcal{A} \text{ accepts } G \text{ in all nodes}] \geq \frac{2}{3}; \\ G \text{ is } \epsilon\text{-far from satisfying } P \implies \Pr[\mathcal{A} \text{ rejects } G \text{ in at least one node}] \geq \frac{2}{3}. \end{cases}$$

Among other results, [23] proved that, in bounded degree graphs, bipartiteness can be distributedly tested in $O(\operatorname{polylog} n)$ rounds in the CONGEST model. Moreover, it is also proved that triangle-freeness can be distributedly tested in $O(1)$ rounds. (The dependence in ϵ is hidden in the big-O notation). This latter result has been recently extended in [62] to testing H-freeness, for every 4-node graph H, in $O(1)$ rounds. On the other hand, it is not known whether distributed testing K_5-freeness or C_5-freeness can be achieved in $O(1)$ rounds, and [62] proves that "natural" approaches based on DFS or BFS traversals do not work. Testing C_k-freeness is $O(1)$ rounds for every $k \geq 3$ has been achieved in [48]. These results have been recently generalized in [33], by proving that T-freeness can be decided in $O(1)$ rounds (see also [77]). Finally, tight bounds on testing graph conductance are provided in [44].

Congested clique

The *congested clique* model is the CONGEST model restricted to complete graphs. Deciding whether a graph given as input contains some specific patterns as subgraphs has been considered in [22] and [28] for the congested clique. In particular, [22] provides an algorithm for deciding the presence of a k-node cycle C_k running in $O(2^{O(k)}n^{0.158})$-rounds.

Recently, [78] defined the class CLIQUE(t) as the class of decision problems that can be decided in time t in the congested clique, and provided a collection of separation results regarding these classes, as a function of t.

In [74] an $\Omega(n/k^2)$ lower bound on the number of rounds for deciding properties such as connectivity and spanning tree is provided for the *k-machine model*, with matching upper bounds in [91]. (In the k-machine model, the edges of the input graph are randomly partitioned among k machines that are linked by a clique, and these k machines proceed as in the CONGEST model with bandwidth limited to one single bit per round).

7.3.3 General interpretation of individual outputs

In [4, 5], a generalization of distributed decision is considered, where every node output not just a single bit (accept or reject), but can output an arbitrary bit-string. The global verdict is then taken based on the multi-set of all the binary strings outputted by the nodes. The concern is restricted to decision algorithms performing in $O(1)$ rounds in the LOCAL model, and the objective is to minimize the size of the outputs. The corresponding basic class BC for outputs on $O(1)$ bits is denoted by ULD, for *universal* LD. (It is universal in the sense that the global interpretation of the individual outputs is not restricted to the logical conjunction). It is proved in [5] that, for any positive even integer Δ, every distributed decision algorithm for cycle-freeness in connected graphs with degree at most Δ must produce outputs of size at least $\lceil \log \Delta \rceil - 1$ bits. Hence, cycle-freeness does not belong to ULD in general, but it does belong to ULD for constant degree graphs.

In [15] the authors consider a model in which each node initially knows the IDs of its neighbours, while the nodes do not communicate through the edges of the network but via a public whiteboard. The concern of [15] is mostly restricted to the case in which every node can write only once on the whiteboard, and the objective is to minimize the size of the message written by each node on the whiteboard. The global verdict is then taken based on the collection of messages written on the whiteboard. It is shown that, with just $O(\log n)$-bit messages, it is possible to rebuild the whole graph from the information on the whiteboard as long as the graph is planar or, more generally, excluding a fixed minor. Variants of the model are also considered, in which problems such as deciding triangle-freeness or connectivity are considered. See also [73] for deciding the presence of induced subgraphs.

7.4 Distributed verification in networks

In this section, we still focus on languages defined as collections of configurations of the form (G, ℓ) where G is a simple connected n-node graph, and $\ell : V(G) \to \{0,1\}^*$ is a labelling function. Recall that an algorithm $\mathcal{A}$ is *verifying* a distributed language $\mathcal{L}$ if and only if, for every configuration (G, ℓ),

$$(G, \ell) \in \mathcal{L} \iff \exists c : \mathcal{A}(G, \ell, c) \text{ accepts at all nodes} \tag{7.4}$$

where $c : V(G) \to \{0,1\}^*$, and $c(v)$ is called the *certificate* of node $v \in V(G)$. Again, the certificate $c(v)$ of node v must not be mistaken with the label $\ell(v)$ of node v. Also, the notion of certificate must not be confused with the notion of *advice*. While the latter are trustable information provided by an oracle [50], the former are proofs that must be verified.

We survey the results about the class $\mathsf{NBC} = \Sigma_1^{\mathsf{BC}}$ where the basic class BC is LD, LDO, ULD, etc.

7.4.1 LOCAL model

It is crucial to distinguish two cases in Eq. (7.4), depending on whether the certificates can depend on the identities assigned to the nodes, or not, as reflected in Eq. (7.5) and (7.6) below.

Local distributed verification (Σ_1^{LD}, PLS, and LCP)

A distributed language $\mathcal{L}$ satisfies $\mathcal{L} \in \Sigma_1^{\mathsf{LD}}$ if and only if there exists a verification algorithm $\mathcal{A}$ running in $O(1)$ rounds in the LOCAL model such that, for every configuration (G, ℓ), we have

$$\begin{cases} (G,\ell) \in \mathcal{L} & \Rightarrow \quad \forall \mathrm{ID}, \exists c,\ \mathcal{A}(G,\ell,c) \text{ accepts at all nodes} \\ (G,\ell) \notin \mathcal{L} & \Rightarrow \quad \forall \mathrm{ID}, \forall c,\ \mathcal{A}(G,\ell,c) \text{ rejects in at least one node} \end{cases} \tag{7.5}$$

where $c : V(G) \to \{0,1\}^*$, and where, for $(G, \ell) \in \mathcal{L}$, the assignment of the certificates to the nodes may depend on the identities given to these nodes. This notion has actually been introduced under the terminology *proof-labelling scheme* in [82], where the concern is restricted to verification algorithms running in just a single round, with the objective of minimizing the size of the certificates. In particular, it is proved that minimum-weight spanning tree can be verified with certificates on $O(\log^2 n)$ bits in n-node networks, and this bound in tight [79] (see also [76]). Interestingly, the $\Omega(\log^2 n)$ bits lower bound on the certificate size can be broken, and reduced to $O(\log n)$ bits, to the price of allowing verification to proceed in $O(\log^2 n)$ rounds [83]. There are tight connections between proof-labelling schemes and compact silent self-stabilizing algorithms [18], and proof-labelling schemes can even be used as a basis to semi-automatically derive compact time-efficient self-stabilizing algorithms [17]. Let PLS be the class of distributed languages for which there exists a proof-labelling scheme. We have

$$\mathsf{PLS} = \mathsf{ALL}$$

where ALL is the class of all distributed languages on networks (i.e., with configurations of the form (G, ℓ)). This equality is however achieved using certificates on $O(n^2 + nk)$ bits in n-node networks, where k is the maximum size of the labels in the given configuration (G, ℓ). The $O(n^2)$ bits are used to encode the adjacency matrix of the network, and the $O(nk)$ bits are used to encode the inputs to the nodes.

The notion of proof-labelling scheme has been extended in [66] to the notion of *locally checkable proofs*, which is the same as proof-labelling scheme but where the verification algorithm is not bounded to run in a single round, but may perform an arbitrarily large constant number of rounds[1]. Let LCP be the associate class of distributed languages. By definition, we have

$$\mathsf{LCP} = \Sigma_1^{\mathsf{LD}},$$

and, more specifically,

$$\mathsf{LCP}(f) = \Sigma_1^{\mathsf{LD}}(f)$$

for every function f bounding the size of the certificates. Moreover, since PLS = ALL, it follows that

$$\mathsf{PLS} = \mathsf{LCP} = \Sigma_1^{\mathsf{LD}} = \mathsf{ALL}.$$

It is proved in [66] that there are natural languages (e.g., the set of graphs with a non-trivial automorphism, 3-non-colourability, etc.) which require certificates on $\tilde{\Omega}(n^2)$ bits in n-node networks. Recently, [14] introduced a mechanism enabling to reduce exponentially the amount of communication in proof-l schemes, using randomization. See also [101] for applications of locally checkable proofs to software-defined networks.

Identity-oblivious algorithms (Σ_1^{LDO} and NLD)

A distributed language $\mathcal{L}$ satisfies $\mathcal{L} \in \Sigma_1^{\mathsf{LDO}}$ if and only if there exists a verification algorithm $\mathcal{A}$ running in $O(1)$ rounds in the LOCAL model such that, for every configuration (G, ℓ), we have

$$\begin{cases} (G,\ell) \in \mathcal{L} & \Rightarrow \quad \exists c,\ \forall \mathrm{ID},\ \mathcal{A}(G,\ell,c) \text{ accepts at all nodes} \\ (G,\ell) \notin \mathcal{L} & \Rightarrow \quad \forall c,\ \forall \mathrm{ID},\ \mathcal{A}(G,\ell,c) \text{ rejects in at least one node} \end{cases} \tag{7.6}$$

[1] Formally, proof-labelling schemes assume that the verification algorithm can use only the certificates, and does not have access to the identifiers of the neighbours, nor to their local information — this restriction is removed in [66], as it has little impact on the results as long as we are dealing with certificates on $\Omega(\log n)$ bits.

where $c : V(G) \to \{0,1\}^*$, and, for $(G,\ell) \in \mathcal{L}$, the assignment of the certificates to the nodes must not depend on the identities given to these nodes. In [54], the class NLD, for *non-deterministic local decision* is introduced. In NLD, even if the certificates must not depend on the identities of the nodes, the verification algorithm is not necessarily identity-oblivious. Yet, it was proved in [52] that restricting the verification algorithm to be identity-oblivious does not restrict the power of the verifier. Hence,

$$\mathsf{NLD} = \Sigma_1^{\mathsf{LDO}}$$

Σ_1^{LDO} is characterized in [52] as the class of languages that are *closed under lift*, where H is a k-lift of G if there exists an homomorphism from H to G preserving radius-k balls. Hence,

$$\Sigma_1^{\mathsf{LDO}} \subset \mathsf{ALL}$$

where the inclusion is strict. However, it was proved in [54] that, for every distributed language $\mathcal{L}$, and for every p, q such that $p^2 + q \leq 1$, there is a non-deterministic (p,q)-decider for $\mathcal{L}$. In other words, for every p, q such that $p^2 + q \leq 1$, we have

$$\mathsf{BPNLD}(p,q) = \mathsf{ALL}.$$

In [54], a complete problem for NLD was identified. However, it was recently noticed in [10] that the notion of local reduction used in [54] is way too strong, enabling to bring languages outside NLD into NLD. A weaker notion of local reduction was thus defined in [10], preserving the class NLD. A language is proved to be NLD-complete under this weaker type of local reduction.

Logarithmic-size certificates (log-Σ_1^{LD} and log-LCP)

It is shown in [66] that many distributed languages can be verified with $\Theta(\log n)$-bit certificates, and hence [66] investigates the class log-LCP, that is, log-Σ_1^{LD}, i.e., Σ_1^{LD} with certificates of size $O(\log n)$ bits. This class fits well with the $\mathsf{CONGEST}$ model, which allows to exchange messages of at most $O(\log n)$ bits at each round. For instance, non-bipartiteness is in log-LCP. Also, restricted to bounded-degree graphs, there are problems in log-LCP that are not contained in NP, but log-$\mathsf{LCP} \subseteq \mathsf{NP}$/poly, i.e., NP with a polynomial-size non-uniform advice. Last but not least, [66] shows that existential MSO on connected graphs is included in log-LCP.

Approximation

The decision languages considered so far are designed to capture decision tasks (e.g., whether a given spanning tree is a MST). For some tasks, such as verifying whether the diameter D of the actual graph is equal to a given value k, there exists no short proof. However, some approximation is easy to certify (e.g., whether $k \leq D \leq 2k$). The notion of *approximate* proof-labelling scheme is defined in [24], where the approximate variants of verifying diameter and verifying maximum matching are investigated.

Space-time trade-offs

Except for MST in [83], the running time of the local verification algorithm has always been considered as constant. The impact of allowing larger, non-constant verification times is studied in [90]. It is shown that, for several languages, allowing a running time of t yields verification schemes using proofs and message of sizes reduced by a factor t compared to the classical 1-round verification schemes.

Error-sensitivity

The notion of *error-sensitive* proof-labelling schemes has been introduced in [37]. Such schemes guarantee that the number of nodes detecting illegal states is linearly proportional to the edit-distance between the current state and the set of legal states. By using error-sensitive proof-labelling schemes, states which are far from satisfying the predicate will be detected by many nodes, enabling fast return to legality. A structural characterization of the set of boolean predicates on network states for which there exist error-sensitive proof-labelling schemes is provided in [37]. This characterization enables to show that classical predicates such as, e.g., acyclicity, and leader admit error-sensitive proof-labelling schemes, while others like regular subgraphs don't. Also, it is shown that the known proof-labelling schemes for spanning tree and minimum spanning tree, using certificates on $O(\log n)$ bits, and on $O(\log^2 n)$ bits, respectively, are error-sensitive, as long as the trees are locally represented by adjacency lists, and not just by parent pointers.

Anonymous networks

Distributed verification in the context of fully anonymous networks (no node-identities, and no port-numbers) has been considered in [46].

7.4.2 Message complexity

Some aforementioned papers also pay attention to minimizing the message size. This is explicitly the case of, e.g., [90].

Unicast vs. broadcast.

The CONGEST model has a *broadcast* variant in which each node are restricted to send the same message to all neighbours at each round (the classical *unicast* variant allows each node to send particular messages to each neighbour). The relative powers of unicast and broadcast model in the context of proof-labelling schemes has been studied in [94]. In particular, it is proved that some languages are insensitive to the broadcast restriction (e.g., spanning tree), whereas others languages (e.g., matching) are significantly impacted by this restriction.

Congested clique

As mentioned before, [78] defined the class $\mathsf{CLIQUE}(t)$ as the class of decision problems that can be decided in time t in the congested clique. They also defined the non-deterministic variant of $\mathsf{CLIQUE}(t)$, denoted by $\mathsf{NCLIQUE}(t)$. A collection of separation results regarding these two classes are provided. The intriguing, and probably challenging open problem of whether $\mathsf{CLIQUE}(O(1)) = \mathsf{NCLIQUE}(O(1))$ is stated in [78].

7.4.3 General interpretation of individual outputs

As already mentioned in Section 7.3.3, a generalization of distributed decision was considered in [4, 5], where every node outputs not just a single bit (accept or reject), but can output an arbitrary bit-string. The global verdict is then taken based on the multi-set of all the binary strings outputted by the nodes. The concern is restricted to decision algorithm performing in $O(1)$ rounds in the LOCAL model, and the objective is to minimize the size of the output. The certificates must not depend on the node IDs, that is, verification proceed as specified in Eq. (7.6). For constant size outputs, it is shown in [4] that the class $\mathsf{UNLD} = \Sigma_1^{\mathsf{ULD}}$ satisfies

$$\mathsf{UNLD} = \mathsf{ALL}$$

with just 2-bit-per-node outputs, which has to be consider in contrast to the fact that NLD is restricted to languages that are closed under lift (cf. Section 7.4.1). This result requires using certificates on $O(n^2 + nk)$ bits in n-node networks, where k is the maximum size of the labels in the given configuration (G, ℓ), but [4] shows that this is unavoidable. Also, while verifying cycle-freeness using the logical conjunction of the 1-bit-per-node outputs requires certificates on $\Omega(\log n)$ bits [66], it is proved in [4] that, by simply using the conjunction and the disjunction operators together, on only 2-bit-per-node outputs, one can verify cycle-freeness using certificates of size $O(1)$ bits.

7.5 Local hierarchies in networks

In this section, we survey the results about the hierarchies Σ_k^{BC} and Π_k^{BC}, $k \geq 0$, for different basic classes BC, including LD, LDO, etc.

7.5.1 Unlimited-size certificates ($\mathsf{DH}^{\mathsf{LD}}$ and $\mathsf{DH}^{\mathsf{LDO}}$)

We have seen in Section 7.4.1 that $\Sigma_1^{\mathsf{LD}} = \mathsf{ALL}$, which implies that the local distributed hierarchy $\mathsf{DH}^{\mathsf{LD}}$ collapses at the first level. On the other hand, we have also seen in Section 7.4.1 that $\Sigma_1^{\mathsf{LDO}} \subset \mathsf{ALL}$, where the inclusion is strict as Σ_1^{LDO} is restricted to languages that are closed under lift. It was recently proved in [10] that

$$\mathsf{LDO} \subset \Pi_1^{\mathsf{LDO}} \subset \Sigma_1^{\mathsf{LDO}} = \Sigma_2^{\mathsf{LDO}} \subset \Pi_2^{\mathsf{LDO}} = \mathsf{ALL}$$

where all inclusions are strict. Hence, the local ID-oblivious distributed hierarchy collapses at the second level. Moreover, it is shown that Π_2^{LDO} has a complete problem for local label-preserving reductions. (A complete problem for ALL was also identified in [54], but using an inappropriate notion of local reduction).

In the context of a general interpretation of individual outputs (see Section 7.4.3), [4] proved that $\Sigma_1^{\mathsf{ULD}} = \mathsf{ALL}$.

7.5.2 Logarithmic-size certificates (log-$\mathsf{DH}^{\mathsf{LD}}$)

We have previously seen that $\Sigma_1^{\mathsf{LD}} = \mathsf{ALL}$. However, this requires certificates of polynomial size. The local distributed hierarchy is revisited in [41], with certificates of logarithmic size. While it follows from [79] that MST $\notin$ log-Σ_1^{LD}, it is shown in [41] that

$$\text{MST} \in \text{log-}\Pi_2^{\mathsf{LD}}.$$

In fact, [41] proved that, for any $k \geq 1$,

$$\text{log-}\Sigma_{2k}^{\mathsf{LD}} = \text{log-}\Sigma_{2k-1}^{\mathsf{LD}} \text{ and } \text{log-}\Pi_{2k+1}^{\mathsf{LD}} = \text{log-}\Pi_{2k}^{\mathsf{LD}},$$

and thus focused only on the hierarchy $(\Lambda_k)_{k\geq 0}$ defined by $\Lambda_0 = \mathsf{LD}$, and, for $k \geq 1$,

$$\Lambda_k = \begin{cases} \text{log-}\Sigma_k^{\mathsf{LD}} & \text{if } k \text{ is odd} \\ \text{log-}\Pi_k^{\mathsf{LD}} & \text{if } k \text{ is even.} \end{cases}$$

It is proved that if there exists $k \geq 0$ such that $\Lambda_{k+1} = \Lambda_k$, then $\Lambda_{k'} = \Lambda_k$ for all $k' \geq k$. That is, the hierarchy collapses at the k-th level. Moreover, there exists a distributed language on 0/1-labelled oriented paths that is outside the Λ_k-hierarchy, and thus outside log-$\mathsf{DH}^{\mathsf{LD}}$. However, deciding whether a given solution to several optimisation problems such as maximum independent set, minimum dominating set, maximum matching, max-cut, min-cut, traveling salesman, etc., is optimal are all in co-Λ_1, and thus in log-Π_2^{LD}. The absence of a non-trivial automorphism is proved to be in Λ_3, that is log-Σ_3^{LD} — recall that this language requires certificated of $\tilde{\Omega}(n^2)$ bits to be placed in Σ_1^{LD} (see [66]). It is however not known whether $\Lambda_3 \neq \Lambda_2$, that is whether log-$\Pi_2^{\mathsf{LD}} \subset$ log-Σ_3^{LD} with a strict inclusion.

7.5.3 Hierarchies in the congested clique

In the congested clique, analogues of the aforementioned hierarchies with unlimited size certificates, as well as with logarithmic size certificates, are studied in [78]. In particular, it is shown that, as in the LOCAL model, the hierarchy with unlimited-size certificates also collapses in the congested clique.

7.5.4 Distributed graph automata ($\mathsf{DH}^{\mathsf{DGA}}$)

An analogue of the polynomial hierarchy, where sequential polynomial-time computation is replaced by distributed local computation was recently investigated in [97].

The model in [97] is called *distributed graph automata*. This model assumes a finite-state automaton at each node (instead of a Turing machine), and assumes anonymous computation (instead of the presence of unique node identities). Also, the model assumes an arbitrary interpretation of the outputs produced by each automaton, based on an arbitrary mapping from the collection of all automata states to $\{\text{true}, \text{false}\}$. The main result in [97] is that the hierarchy $\mathsf{DH}^{\mathsf{DGA}}$ coincides with MSO on graphs.

7.6 Other computational models

7.6.1 Wait-free computing

The class WFD defined as the class of all distributed languages that are wait-free decidable was characterized in [55] as the class of languages satisfying the so-called *projection-closeness* property. For non projection-closed languages, [58] investigated more general interpretation of the individual opinions produced by the processes, beyond the logical conjunction of boolean opinons. In [57], it is proved that k-set agreement requires that the processes must be allowed to produce essentially k different opinions to be wait-free decided. The class Σ_1^{WFD} has been investigated in [60, 61], with applications to the space complexity of failure detectors. Interestingly, it is proved in [20] that wait-free decision finds applications to run-time verification.

7.6.2 Mobile computing

The class MAD, for *mobile agent decision* has been considered in [49], as well as the class $\mathsf{MAV} = \Sigma_1^{\mathsf{MAD}}$, for *mobile agent verification*. It is proved that MAV has a complete language for a basic notion of reduction. The complement classes of MAD and MAV have been recently investigated in [12] together with sister classes defined by other ways of interpreting the opinions of the mobile agents.

7.6.3 Quantum computing

Distributed decision in a framework in which nodes can have access to extra ressources, such as shared randomness, or intricate variables (in the context of quantum computing) is discussed in [3]. In [29], the techniques of [27] are extended to a quantum setting.

Chapter 8

Conclusion and perspectives

This thesis revolves around the notion of proof-labelling scheme defined fifteen years ago. In the light of the recent developments of distributed decision, these schemes can be interpreted as a form of non-determinism for distributed computing, with a focus on locality. We have explored other forms of non-determinism, which correspond to diverse practical scenarios, and allow to better understand some aspects of proof-labelling schemes. Namely, forcing a stronger rejection leads to define and characterize error-sensitivity. Also allowing larger radius or global proofs is useful to study redundancy and uniformity. Finally alternation is an interesting form of interactivity.

After this work, a lot of questions remain open, and we describe some of them in the rest of this chapter. We first list problems that are specific to each chapter, and then complete that list with a few more general problems. Section 8.2 is also an opportunity to give a new point of view on this thesis.

8.1 Open problems chapter by chapter

8.1.1 Error-sensitivity

Our study of error-sensitive proof-labelling schemes raises intriguing questions. In particular, we observe that every distributed language seems to fit in one of the following two scenarios: either it is not error-sensitive, or it admits an error-sensitive proof-labelling scheme that achieves the same performance (in terms of proof size) as the optimal scheme without the sensitivity constraint. We do not know whether there exists a distributed language that contradicts this dichotomy. This is the main open question of this chapter.

Open problem 8.1. *Does there exist a (natural) error-sensitive language, with a proof-labelling scheme of size $f(n)$, such that every error-sensitive proof-labelling scheme has size at least $g(n)$ with $f(n) \ll g(n)$? Loosely speaking: does error-sensitivity, whenever achievable, always come for free?*

Another interesting topic is about *where* rejection happens. Error-sensitivity is basically about comparing the sizes of two sets of nodes: the nodes that are rejecting, and the nodes whose inputs we can edit in order to produce a labelling that is correct for the language. A natural constraint is to require that the nodes to edit are the rejecting nodes. This way, in some practical scenarios, it would be possible for a node to both detect, and fix a bad configuration. We can call this constraint *proximity sensitivity.* Note that, as we mentioned in the corresponding chapter, spanning trees are error-sensitive, but do not satisfy this stronger property. Characterizing languages that are proximity-sensitive is a natural first step for understanding this notion.

Open problem 8.2. *Characterize the languages that are proximity-sensitive.*

On a more technical level, the proof that the classic scheme for minimum spanning trees is error-sensitive is complex, thus the following open problem.

Open problem 8.3. *Does there exist a simpler proof that the classic scheme for minimum spanning is error-sensitive? Does there exist another scheme, with a simpler proof?*

Finally, it is known that spanning trees, and minimum spanning trees are related to the notion of (graphic) matroid. This notion implicitly appears in the proofs of Corollaries 3.2 and 3.3.

Open problem 8.4. *Is the notion of local stability related to the notion of matroid? Are there other examples of matroids that are, in some sense, locally stable?*

8.1.2 Uniformity

After the theorems of Chapter 4, it seems that we have a pretty good grasp of the relations between mixed proofs and local proofs. However purely global proofs are still not completely understood. We have already discussed the following open question in the corresponding chapter.

Open problem 8.5. *For the set of bipartite graphs, is the optimal size for a global proof $\Theta(n \log n)$ bits? If not, are there languages with local proofs of constant size, but global proof of super-linear size?*

Also, it seems worth studying the relations between error-sensitivity and global proofs. For example a positive answer to the following question seems plausible.

Open problem 8.6. *Is every language error-sensitive if we consider purely global proofs?*

Indeed the typical prover strategy to make few nodes reject, and then violate the sensitivity constraint, consists in telling half of the nodes that they live in some graph, and the other half that they live in some other graph; but this is not possible with a global proof.

8.1.3 Redundancy

In the chapter about redundancy, we proved that, for many classic problems, there are proof-labelling schemes that scale linearly with the radius of the view. This even holds for every language, if the topology is restricted to cycles or trees. Thus the following problem:

Open problem 8.7. *Does every language scale linearly?*

In fact, the scaling factor might even be larger than the radius t, and be as large as $b(t)$ in graphs with ball growth b. For example, we have proved that the uniform part of any proof-labelling scheme can be scaled by such a factor $b(t)$. This yields the following further open problem:

Open problem 8.8. *Does every language scale by the size of the ball?*

The first step to give an answer to these questions may be to understand colouring problems. In Chapter 4, we saw that colourability questions, such as bipartiteness, provide examples of languages that are far from being uniform. Thus we may expect that the languages such as k-COLOURABLE are not very redundant. Therefore these languages are good candidates to disprove the existence of a good scaling for every language. Note that colourability with a constant number of colours does not make much sense for scaling, as the original proofs use only a constant number of bits. Thus the following question.

Open problem 8.9. *What is the scaling for the set of $f(n)$-colourable graphs, where f is some increasing function?*

The basic prover strategy for colourability problems is to provide every node with its colour, and the classic way to scale a scheme is to start by removing some certificates. Then the question is: if a fairly large fraction of colours is removed, how to reconstruct locally the colouring? Note that in the case of a path, if the nodes that are coloured are spread evenly, this local reconstruction is easy. But, in some sense, we already knew that: we proved that for constrained topologies, such as paths, every language, including colourability languages, scales linearly.

8.1.4 Interactivity

In Chapter 6, we defined classes on different levels of a hierarchy, and the main question was:

Open problem 8.10. *Are the second level and the third level of the hierarchy separated? If yes, are all the levels separated?*

As said before, this question is probably out of reach for the moment if we focus on techniques from communication complexity. However it may be possible to prove lower bounds in a different way.

Another topic of interest is whether Theorem 6.10 can be generalized. This theorem states that there are languages outside of the local hierarchy, if we assume that the verifier is a Turing machine.

Open problem 8.11. *Are there problems outside the hierarchy if we do not restrict the nodes to run a Turing machine? In other words can every language be described by a protocol of the hierarchy?*

8.2 Cross-cutting view and new questions

8.2.1 Building blocks for proofs

When we survey the classic proof-labelling schemes, it appears that a few basic ideas are used for many purposes. For example, using distances or counters is a very common primitive. We now list these classic primitives, and state what are their characteristics in terms of sensitivity, uniformity, redundancy, and interactivity. This is an opportunity to give a cross-cutting view on the thesis. Most of these primitives do not have standard names, therefore we coin the following terms: *distance-based proofs*, *counter-based proofs*, *name-based proofs*, and *local structure proofs*.

- *Distance-based proofs* consist in providing each node with its distance to some leader node along some tree. This can be relaxed by only requiring to have an ordering which respects the property that parents have smaller ranks than their children. These proofs are mainly used to show acyclicity, and to show that some pattern appears in the graph. As we have seen in Chapter 4, such proofs are far from being uniform, as the corresponding price of locality is constant. However, this does not mean that the proofs are not redundant. Indeed the distances of two adjacent nodes are very correlated, which is captured by the linear scaling appearing with a larger radius (Chapter 5). Finally such proofs are, in some sense, sensitive to errors because, given the good encoding, the tasks that are using those proofs are error-sensitive.

- *Counter-based proofs* are a refinement of distance-based proofs. On top of a tree with distances, it is possible to enrich the certificates to provide more information. This additional information comes in the form of some number, or some boolean value. These values are aggregate along the tree, and the root can check that this aggregated number is correct with respect to the language. For example, we can check whether the number of edges in a matching is equal to some given number. The idea is that every node is given by the prover, the number of edges of the matching that are adjacent to one of its descendants. Then every node can check that the numbers given to its neighbours are consistent with its own number, and its neighbourhood. The proof of Claim 6.2, is a more general use of this technique.

- *Name-based proofs* are proofs that are uniform, and consist in the name of an edge, a node, or an input. These are useful when many nodes need the same information, for example to certify that some pattern appears at most once. They are uniform by definition, thus very redundant, as witnessed by an extremely good scaling.

- *Universal proofs* are uniform proofs where every node is basically given the adjacency matrix of the graph. As in the previous type of proof mentioned, these are uniform proofs, thus they are redundant. Universal proofs are the archetype of proofs that are not sensitive. For some languages such as the set of symmetric graphs, one can avoid using such proofs if alternation is allowed. Indeed with alternation, one can use only distance-based and name-based proofs, that are more compact.

- *Local-structure proofs* are proofs that, loosely speaking, provide some local information, such as a colour, the name of an adjacent edge in a matching, and so on. In general it is difficult to capture this notion with a satisfying definition. However if we restrict to degree-bounded graphs with port-numbers, one can define local-structure proofs as the proofs that can be encoded on a constant number of bits. For example, in this restricted setting, encoding a matching can be done with a constant number of bits: every node is just given the port number of the edge in the matching it is adjacent to. More generally all the languages that correspond to construction problems in the class LCL in [89] fit into this definition. These local-structure proofs elude some of our techniques, in particular it is not clear how uniform or redundant they are. Finally, sensitivity depends on the scheme itself, and on the precise parameters of the language. For example, 3-colourings in general graphs are not locally stable, but colourings with $\Delta + 1$ colours, in graphs of maximum degree Δ, are locally stable.

8.2.2 Towards a canonical form for local proofs

Now, given this list of ubiquitous primitives, a natural question is the following.

Open problem 8.12. *Can every proof-labelling scheme be transformed into another scheme, satisfying the two following properties: it is as compact as the original scheme, and it only uses the primitives of the list? More generally, is it possible to define a canonical form for local proofs?*

The rationale behind this question is twofold. First, there has been quite a few papers on proof-labelling schemes, but they almost always use the same building blocks. Second, the proofs mentioned in the previous subsection are sufficient to certify fundamental languages such as Amos and Alos, so maybe they are versatile enough to capture the essence of local proofs. A hurdle to overcome to answer this question is to precisely define how to compose these primitives. For example, proof-labelling schemes for minimum spanning trees and for diameter use these primitives, but in a complicated, interleaved, manner. Thus, as a first step, it may be easier to restrict ourselves to the study of languages with $O(\log n)$-bit proofs, that avoids these two examples.

Open problem 8.13. *Is it true that on graphs with constant degrees and constant-size inputs, every scheme with $O(\log n)$-bit proofs, can be replaced by a scheme using only distance, counter, and name-based proofs, along with some constant-size additional information?*

Such general results would be very helpful in proving properties of proof-labelling schemes. Indeed, instead of proving theorems about general unknown proofs, one could simply prove the statement for each primitive, and then prove that composing these primitives does not change the correctness of the result. For example, in this thesis, we have often proved theorems for particular languages, such as spanning trees, but if some canonical form would exist, then we may be able to say that these special cases are actually the only ones to consider.

8.3 Perspectives

To conclude, let us broaden the focus, by considering the impact of proof-labelling schemes outside of distributed decision.

First remember that proof-labelling schemes originated from the study of self-stabilizing algorithms. Self-stabilizing algorithms are distributed algorithms that cope with faults. To detect these faults, a common strategy consists in providing the nodes with additional information. In their seminal paper, Korman, Kutten and Peleg, advocated a modular approach to study self-stabilization. In particular they isolated the part concerned with additional information, and modelled it as non-determinism. This process gave birth to proof-labelling schemes, which became a topic of research on its own. After fifteen years of research, we have a better understanding of proof-labelling schemes. It may be time to use the knowledge gathered along the way to improve on self-stabilizing algorithms. In particular, we have mentioned diverse practical scenarios for different types of non-determinism, and it would be nice to have self-stabilizing algorithms for these settings. Also, a very related topic is the dynamic setting. In this thesis we have always assumed that the graph and the inputs do not change over time. Understanding how robust proof-labelling schemes can be in presence of changes is a natural next step.

The second topic outside of distributed decision that may benefit from further knowledge on proof-labelling schemes is graph theory. As we mentioned in Section 1.4, we have ongoing work concerned with the graph classes that can be described with some forbidden ordered patterns. More precisely, we call *pattern*, graphs whose nodes are ordered, and *pattern family*, a finite set of such patterns. Given a pattern family, one can consider the following class of graphs: the graphs for which there exists an ordering of the nodes such that none of the patterns of the family appears. The graph classes defined this way are all in NP, because given the ordering, it is easy to check whether the graph is in the class or not. The link with proof-labelling schemes is that, for some classes, we can actually provide this ordering in the form of local proofs. That is, for some pattern families, we can design a scheme that provides each node with its rank in the ordering, and the nodes can both verify the ordering, and check whether a pattern appears. It is not clear yet which are exactly the classes that can be recognized by this type of schemes. On a more general perspective, it is probably fruitful to use the point of view of proof-labelling schemes in the study of graph classes. For example, the size of the optimal proofs for certifying a class may be an interesting parameter to bring to the graph theory community.

Bibliography

[1] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 29–42, 2016. doi:10.1007/978-3-662-53426-7_3.

[2] Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theor. Comput. Sci.*, 186(1-2):199–229, 1997. doi:10.1016/S0304-3975(96)00286-1.

[3] Heger Arfaoui and Pierre Fraigniaud. What can be computed without communications? *SIGACT News*, 45(3):82–104, 2014. doi:10.1145/2670418.2670440.

[4] Heger Arfaoui, Pierre Fraigniaud, and Andrzej Pelc. Local decision and verification with bounded-size outputs. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 133–147, 2013. doi:10.1007/978-3-319-03089-0_10.

[5] Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. Distributedly testing cycle-freeness. In *40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 15–28, 2014. doi:10.1007/978-3-319-12340-0_2.

[6] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Wiley, 2004.

[7] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd Symposium on Foundations of Computer Science (FOCS)*, pages 268–277, 1991. doi:10.1109/SFCS.1991.185378.

[8] Lazslo Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proc. 27th Annual Symposium on Foundations of Computer Science (FOCS 1986)*, pages 337–347, 1986. doi:10.1109/SFCS.1986.15.

[9] Alkida Balliu and Pierre Fraigniaud. Certification of compact low-stretch routing schemes. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 6:1–6:16, 2017. doi:10.4230/LIPIcs.DISC.2017.6.

[10] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 8:1–8:13, 2017. doi:10.4230/LIPIcs.STACS.2017.8.

[11] Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1307–1318, 2018. doi:10.1145/3188745.3188860.

[12] Evangelos Bampas and David Ilcinkas. On mobile agent verifiable problems. In *12th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 123–137, 2016. doi:10.1007/978-3-662-49529-2_10.

[13] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs - theory, algorithms and applications*. Springer, 2002. ISBN 978-1-85233-611-0.

[14] Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 315–324, 2015. doi:10.1145/2767386.2767421.

[15] Florent Becker, Adrian Kosowski, Martín Matamala, Nicolas Nisse, Ivan Rapaport, Karol Suchan, and Ioan Todinca. Allowing each node to communicate only once in a distributed system: shared whiteboard models. *Distributed Computing*, 28(3):189–200, 2015. doi:10.1007/s00446-014-0221-8.

[16] Marcin Bienkowski, Leszek Gasieniec, Marek Klonowski, Miroslaw Korzeniowski, Bernard Mans, Stefan Schmid, and Roger Wattenhofer. Distributed alarming in the on-duty and off-duty models. *IEEE/ACM Trans. Netw.*, 24(1):218–230, 2016. doi:10.1109/TNET.2014.2359684.

[17] Lélia Blin and Pierre Fraigniaud. Space-optimal time-efficient silent self-stabilizing constructions of constrained spanning trees. In *35th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 589–598, 2015. doi:10.1109/ICDCS.2015.66.

[18] Lélia Blin, Pierre Fraigniaud, and Boaz Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 18–32, 2014. doi:10.1007/978-3-319-11764-5_2.

[19] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

[20] Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David A. Rosenblueth, and Corentin Travers. Decentralized asynchronous crash-resilient runtime verification. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 16:1–16:15, 2016. doi:10.4230/LIPIcs.CONCUR.2016.16.

[21] Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.

[22] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152, 2015. doi:10.1145/2767386.2767414.

[23] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 43–56, 2016. doi:10.1007/978-3-662-53426-7_4.

[24] Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, pages 71–89, 2017. doi:10.1007/978-3-319-72050-0_5.

[25] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 615–624, 2016. doi:10.1109/FOCS.2016.72.

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN 978-0-262-03384-8. url: http://mitpress.mit.edu/books/introduction-algorithms.

[27] Atish Das-Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41 (5):1235–1265, 2012. doi:10.1137/11085178X.

[28] Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In *26th International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012. doi:10.1007/978-3-642-33651-5_14.

[29] Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. Can quantum communication speed up distributed computation? In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 166–175, 2014. doi:10.1145/2611462.2611488.

[30] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. Anonymous networks: randomization = 2-hop coloring. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 96–105, 2014. doi:10.1145/2611462.2611478.

[31] Yuval Emek, Jochen Seidel, and Roger Wattenhofer. Computability in anonymous networks: Revocable vs. irrecovable outputs. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 183–195, 2014. doi:10.1007/978-3-662-43951-7_16.

[32] Guy Even, Moti Medina, and Dana Ron. Best of two local models: Local centralized and local distributed algorithms. 2014. arXiv: 1402.3796.

[33] Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In *31st International Symposium on Distributed Computing (DISC)*, 2017. doi:10.4230/LIPIcs.DISC.2017.15.

[34] Laurent Feuilloley. How long it takes for an ordinary node with an ordinary ID to output? In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, pages 263–282, 2017. doi: 10.1007/978-3-319-72050-0.

[35] Laurent Feuilloley and Pierre Fraigniaud. Randomized local network computing. In *27th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 340–349, 2015. doi:10.1145/2755573.2755596.

[36] Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119, 2016. url: bulletin.eatcs.org link, arXiv: 1606.04434.

[37] Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 16:1–16:15, 2017. doi:LIPIcs.DISC.2017.16.

[38] Laurent Feuilloley and Michel Habib. Graph classes and forbidden patterns on three vertices, 2018. Manuscript.

[39] Laurent Feuilloley and Juho Hirvonen. Local verification of global proofs. In *DISC 2018 (To appear)*, 2018. arXiv: 1803.09553.

[40] Laurent Feuilloley, Juho Hirvonen, and Jukka Suomela. Locally optimal load balancing. In *29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 544–558, 2015. doi: 10.1007/978-3-662-48653-5.

[41] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 118:1–118:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.118.

[42] Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. In *DISC 2018 (To appear)*, 2018. arXiv: 1803.03031.

[43] Faith E. Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3):121–163, 2003. doi:10.1007/s00446-003-0091-y.

[44] Hendrik Fichtenberger and Yadu Vasudev. A two-sided error distributed property tester for conductance. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 19:1–19:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.19.

[45] Paola Flocchini, Guiseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots.* Morgan & Claypool, 2012.

[46] Klaus-Tycho Förster, Thomas Luedi, Jochen Seidel, and Roger Wattenhofer. Local checkability, no strings attached. In *17th International Conference on Distributed Computing and Networking (ICDCN)*, page 21, 2016. doi:10.1145/2833312.2833315.

[47] Pierre Fraigniaud. Locality in distributed graph algorithms. In *Encyclopedia of Algorithms*, pages 1143–1148. Springer, 2016. doi:10.1007/978-1-4939-2864-4_608.

[48] Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 153–162, 2017. doi:10.1145/3087556.3087571.

[49] Pierre Fraigniaud and Andrzej Pelc. Decidability classes for mobile agents computing. In *10th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 362–374, 2012. doi:10.1007/978-3-642-29344-3_31.

[50] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 21(6):395–403, 2009. doi:10.1007/s00446-008-0076-y.

[51] Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 708–717, 2011. doi:10.1109/FOCS.2011.17.

[52] Pierre Fraigniaud, Magnús M. Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *16th International Conference Principles of Distributed Systems (OPODIS)*, pages 224–238, 2012. doi:10.1007/978-3-642-35476-2_16.

[53] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 157–165, 2013. doi:10.1145/2484239.2484264.

[54] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35, 2013. doi:10.1145/2499228.

[55] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Locality and checkability in wait-free computing. *Distributed Computing*, 26(4):223–242, 2013. doi:10.1007/s00446-013-0188-x.

[56] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014. doi:10.1007/s00446-014-0211-x.

[57] Pierre Fraigniaud, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. The opinion number of set-agreement. In *18th International Conference on the Principles of Distributed Systems (OPODIS)*, pages 155–170, 2014. doi:10.1007/978-3-319-14472-6_11.

[58] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *5th International Conference on Runtime Verification (RV)*, pages 92–107, 2014. doi:10.1007/978-3-319-11164-3_9.

[59] Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. In *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 31–45, 2015. doi:10.1007/978-3-319-25258-2_3.

[60] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Minimizing the number of opinions for fault-tolerant distributed decision using well-quasi orderings. In *12th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 497–508, 2016. doi:10.1007/978-3-662-49529-2_37.

[61] Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers, Petr Kuznetsov, and Thibault Rieutord. Perfect failure detection with very few bits. In *Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, SSS 2016, Lyon, France, November 7-10, 2016, Proceedings*, pages 154–169, 2016. doi:10.1007/978-3-319-49259-9_13.

[62] Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 342–356, 2016. doi:10.1007/978-3-662-53426-7_25.

[63] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983. doi:10.1145/357195.357200.

[64] Oded Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography – Miscellanea on the Interplay between Randomness and Computation*, number 6650 in LNCS, pages 465–469. Springer, 2011. doi:10.1007/978-3-642-16367-8_1.

[65] Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography – Miscellanea on the Interplay between Randomness and Computation*, number 6650 in LNCS, pages 470–506. Springer, 2011. doi:10.1007/978-3-642-16367-8_7.

[66] Mika Göös and Jukka Suomela. Locally checkable proofs. In *30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 159–168, 2011. doi:10.1145/1993806.1993829.

[67] Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(19):1–33, 2016. doi:10.4086/toc.2016.v012a019.

[68] Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 201–214, 2016. doi:10.1007/978-3-662-53426-7_15.

[69] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 133–142, 2015. doi:10.1145/2688073.2688079.

[70] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. *Computational Complexity*, 27(1):99–207, 2018. doi:10.1007/s00037-016-0136-9.

[71] Magnús M. Halldórsson and Tigran Tonoyan. How well can graphs represent wireless interference? In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 635–644, 2015. doi:10.1145/2746539.2746585.

[72] Gene Itkis and Leonid A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 226–239, 1994. doi:10.1109/SFCS.1994.365691.

[73] Jarkko Kari, Martín Matamala, Ivan Rapaport, and Ville Salo. Solving the induced subgraph problem in the randomized multiparty simultaneous messages model. In *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 370–384, 2015. doi:10.1007/978-3-319-25258-2_26.

[74] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 391–410, 2015. doi:10.1137/1.9781611973730.28.

[75] Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed MST verification. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 69–80, 2011. doi:10.4230/LIPIcs.STACS.2011.69.

[76] Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory Comput. Syst.*, 53(2):318–340, 2013. doi:10.1007/s00224-013-9479-7.

[77] Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, pages 4:1–4:16, 2017. doi:10.4230/LIPIcs.OPODIS.2017.4.

[78] Janne H. Korhonen and Jukka Suomela. Brief announcement: Towards a complexity theory for the congested clique. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 55:1–55:3, 2017. doi:10.4230/LIPIcs.DISC.2017.55.

[79] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007. doi:10.1007/s00446-007-0025-1.

[80] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes [detailed version]. url: https://ie.technion.ac.il/ kutten/ps-links/ProofLabelingSchemes.pdf.

[81] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC 2005, Las Vegas, NV, USA, July 17-20, 2005*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

[82] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.

[83] Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self-stabilizing verification, computation, and fault detection of an MST. *Distributed Computing*, 28(4):253–295, 2015. doi:10.1007/s00446-015-0242-y.

[84] Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009. doi:10.1016/j.cosrev.2009.04.003.

[85] Fabian Kuhn. *The price of locality: exploring the complexity of distributed coordination primitives.* PhD thesis, ETH Zurich, 2005. url: http://d-nb.info/977273725.

[86] Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, 1997. ISBN 978-0-521-56067-2.

[87] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21 (1):193–201, 1992. doi:10.1137/0221015.

[88] László Lovász and Katalin Vesztergombi. Non-deterministic graph property testing. *Combinatorics, Probability & Computing*, 22(5):749–762, 2013.

[89] Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[90] Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, pages 53–70, 2017. doi:10.1007/978-3-319-72050-0_4.

[91] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 429–438, 2016. doi:10.1145/2935764.2935785.

[92] Christos H. Papadimitriou. Algorithms, games, and the internet. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 749–753, 2001. doi:10.1145/380752.380883.

[93] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007. doi:10.1016/j.tcs.2007.04.040.

[94] Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: Broadcast, unicast and in between. In *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*, pages 1–17, 2017. doi:10.1007/978-3-319-69084-1_1.

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

[96] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *40th Symp. on Foundations of Computer Science (FOCS)*, pages 253–261. IEEE, 1999. doi:10.1109/SFFCS.1999.814597.

[97] Fabian Reiter. Distributed graph automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 192–201, 2015. doi:10.1109/LICS.2015.27.

[98] Fabian Reiter. *Distributed automata and logic.* PhD thesis, University Paris Diderot, 2017. arXiv: 1805.06238.

[99] Marcus Schaefer. Deciding the Vapnik-Cervonenkis dimension in Σ_3^p-complete. *J. Comput. Syst. Sci.*, 58(1):177–182, 1999. doi:10.1006/jcss.1998.1602.

[100] Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

[101] Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In *Proceedings of the Second ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 121–126, 2013. doi:10.1145/2491185.2491198.

[102] Thomas Schwentick and Klaus Barthelmann. Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics & Theoretical Computer Science*, 3(3):109–124, 1999. url: http://dmtcs.loria.fr/volumes/abstracts/dm030303.abs.html.

[103] Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24, 2013. doi:10.1145/2431211.2431223.

[104] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing. In *11h Symp. on Theory of Computing (STOC)*, pages 209–213. ACM, 1979. doi:10.1145/800135.804414.