Introduction to local certification

Laurent Feuilloley

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **23**:3, 2021, #9

Introduction to local certification

Laurent Feuilloley

Université Lyon 1, LIRIS

received 14th Apr. 2020, accepted 5th Sep. 2021.

A distributed graph algorithm is an algorithm where every node determines its output by inspecting its local neighborhood. As distributed environments are subject to faults, an important issue is to be able to check that the output is correct, or in general that the network is in proper configuration with respect to some predicate. One would like this checking to be very local, to avoid using too much time. Unfortunately, most predicates cannot be checked this way, and that is where certification comes into play. Local certification (also known as proof-labeling schemes, locally checkable proofs, or distributed verification) consists in assigning labels to the nodes, that certify that the configuration is correct. In this paper, we present several different perspectives for studying local certification: as a part of self-stabilizing algorithms, as a labeling problem, or as a non-deterministic distributed decision.

This paper is an introduction to the domain of local certification, giving an overview of the history, the techniques and the current research directions.

Keywords: Local certification, proof-labeling scheme, distributed decision, locally checkable proofs

1 Introduction

Let us consider an informal example that illustrates the concept of local certification. (Formal definitions will follow, and a more formal writing will be used in the rest of the paper.) The scenario is the following. The nodes of a connected graph have to decide collectively whether the graph they belong to is a path. Every node wakes up, with no knowledge about the graph, inspects its local neighborhood, and then decides either to stay silent or to raise an alarm. If the graph is a path, all nodes should stay silent, but if the graph is not a path, then at least one node should raise an alarm. If a node stays silent, we will say that it accepts, and if it raises an alarm we will say that it rejects.

There are some cases where it is easy to detect that something is wrong: if a node sees that it has more than two neighbors, then it can safely raise an alarm. But what if the graph is a cycle? Then every node sees that it has two neighbors, and as the graph is not a path, at least one node should reject. If we assume that the nodes are all identical, then the only possibility is that all nodes reject. Then consider a node in the middle of a long path. It sees two neighbors, thus it has the same partial view of the network as the nodes in the cycle, and as a consequence, it rejects. This is an incorrect behavior: a correct instance (a path) has a node raising an alarm. Note that if instead of accepting paths and rejecting cycles, we aimed for the converse, that is accepting cycles and rejecting paths, the task would be much easier: we could simply make nodes of degree one reject.

ISSN 1365–8050

We have just proved informally that, in our model, the nodes of a network cannot accept paths and reject cycles. The assumption that the nodes are identical is actually not necessary for this impossibility result. Indeed, if we assume that the nodes run the same algorithm but that they have distinct identifiers, which will be our standard model in this paper, the result still holds. This is explained in Figure 1. Some specific properties can actually be checked in the model above, for example the fact that a graph is properly 3-colored, as illustrated in Figure 2. But for most properties, this verification mechanism is too weak. The notion of certification originates from this limitation. In order to verify a property, we could allow more

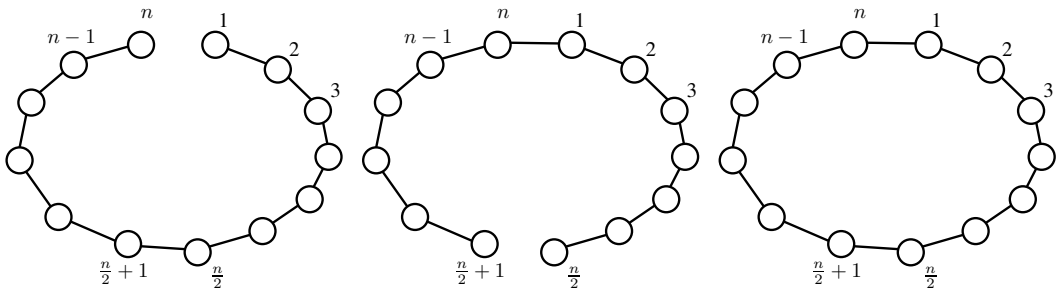


Fig. 1: This figure illustrates that, even with distinct identifiers, it is not possible to locally distinguish between paths and cycles. We consider three graphs on n nodes with unique identifiers between 1 and n. The two first graphs are paths, and the third is a cycle. Consider the view of a node of the third graph, that is, the node itself and its two neighbors. By construction, this view also appears in at least one of the two paths. Now, because the third graph is a cycle, one node v should reject. There is a node with the same view in one of the paths, and this node should also reject, because it has the exact same view of the system. This is a contradiction, because in a path no node should reject.

resources, for example allowing the nodes to see further in the graph. But verification should not consume a lot of resources as it is not the main computation, but a secondary computation that is useful only in case of faults. Consequently, we want to keep the verification very local. Another approach consists in giving some information to the nodes. We will do the following: assign *certificates* to the nodes that allow them to check the property. For the sake of concreteness, let us go back to the path example. How can we convince the nodes that they live in a path and not in a cycle? One efficient way to do so is the following: chose one of the endpoints of the path, and give to every node its distance to this node [1]. Now a node can see its distance and the ones of its neighbors. It can easily check that the distances are consistent: either (1) it has been assigned distance 0, and then it should have degree 1, or (2) it has a distance $d > 0$, and it should check that one of its neighbors has distance $d - 1$ and the other (if it exists) has distance $d + 1$. The crucial point is that the nodes cannot be fooled. That is, if the graph is a cycle, and one assigns numbers to the nodes pretending that these are distances to one endpoint, at least one node will detect the inconsistency and reject (for example a node with the smallest distance). The labels assigned to the nodes are called the certificates, and the global mechanism is called local certification. Now many questions can be asked. For example, the certification of paths we have just described uses labels on $O(\log n)$ bits for graphs on n nodes; is this optimal? For some other property, does there exist a similar mechanism? What if we allow the view of the nodes to be slightly larger? And also, this labeling given by an untrustworthy

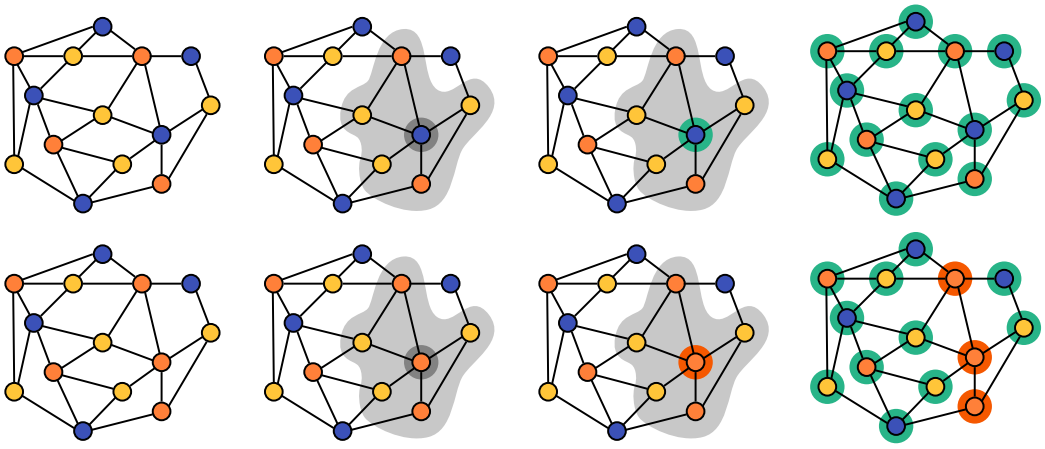


Fig. 2: This figure illustrates that for some well-known property one can actually verify locally (without additional help). The property is the following: every node of the graph is given a color as input, and no edge should have both of its endpoints of the same color. The verification is very simple: every node checks that none of its neighbors has the same color as its own. In the first row of pictures, the instance is correct, and every node accepts, but in the second row, several nodes notice that the coloring is not correct. (In both rows, we highlight the behavior of one specific node, and then illustrate the collection of outputs.)

oracle is similar to NP in centralized computing, can we push the analogy further? This is what research in local certification is about, and that is the topic of this paper.

Organization of the paper. The organization of the rest of the paper is the following. In Section 2, we give some historical perspective, describing where the notion of certification comes from and how it fits in the field of distributed graph algorithms. In Section 3, we introduce the definitions and vocabulary. In Section 4, we give an overview of what we know about the performance of local certification in terms of label size, both in terms of results and technique. In Section 5, we take the more complexity-theoretic point of view of distributed decision. Finally, in Section 6, we describe the current research directions being explored, and provide a list of open problems.

2 Context and historical perspective

Local certification can be seen as part of two related lines of work in the theory of distributed computing. On the one hand, the study of locality, and on the other hand the study of self-stabilization. Both of these areas are about construction tasks (*e.g.* computing a proper coloring of the graph), and not about decision tasks like the ones we study in this paper. Nevertheless, decision tasks appear implicitly at theirs cores. Let us give some key notions about these two fields before we move to certification itself.

Study of locality and locally checkable labelings. The idea of locality in distributed computing consists in focusing on the time it takes for information to be disseminated in the network, putting aside (or at least on the background) other issues such as asynchrony, failures or bandwidth limitations. A classic model is the LOCAL model, where a node can see a neighborhood at distance T in the network, and

choose an output given this partial view. The main goal from the algorithm designer perspective is then to minimize T for some given task. The study of locality in distributed computing dates back to the mid-80's (*e.g.* [54, 3, 17, 44]) and to 90s with the seminal papers [53] and [56], and was developed into a book in 2000 [61]. In [56], motivated by [1], the authors define a class of (construction) problems called *locally checkable labeling*, or 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. This is exactly the kind of problem for which no certification is needed, and coloring (which was the example of Figure 2) is probably the most classic such problem. Other problems are finding a maximal independent sets, a maximal matching or an edge coloring. This is a very active area of research (see [12, 46] for recent books on the topic). As the name suggests, local certification has a strong locality flavor: the nodes have very limited view of the network. Nevertheless, the concepts originates from another area of distributed computing, self-stabilization.

Self-stabilization. Unlike the classic study of locality, self-stabilization is concerned with faults. It provides an answer to the fundamental question: how to be fault-tolerant? The goal is to design algorithms that are self-stabilizing, which means that, starting from an arbitrary configuration, they can converge to a correct configuration [19]. This ensures fault-tolerance because after an arbitrary number of faults, the algorithms will eventually reach a correct configuration. A classic book on the topic is [20]. As said above, in self-stabilization, the main challenge is to converge to a correct configuration from an arbitrary one. But there is another issue: once you have reached a correct configuration, you would like to stay there. In some self-stabilizing contexts, one can move between correct configurations, and the algorithms continues to compute, but being able to stay in the same correct configuration is sometimes a desirable property. In *silent self-stabilization*, algorithms should reach a correct configuration, and stay in this configuration, unless a fault occurs. In order to be able to detect a possible fault, and also to be sure that the algorithm has stabilized, a very restricted checking procedure continues to be run: every node periodically checks its neighbors to be sure that everything is correct. We want that if a fault occurs and makes the configuration incorrect, at least one node should be able to detect this while performing its verification. This is enough, because this node can then launch a recovery procedure (*e.g.* a global reset), and start the computation of a new correct configuration. On the other hand, if the configuration is correct, we do not want to restart computing. This way of looking at self-stabilization originates from the seminal works of [1, 2, 8, 7, 9, 21]. One can see that the "stabilized phase" of a silent self-stabilizing algorithm is basically the scenario described in the introduction. In the context of self-stabilization, the certificates are pieces of information that are kept in memory during the computation of the solution, to allow the later verification of correctness. The example of the path is a bit unusual for this because, for simplicity, we decided to consider a property of the network, and not the correctness of a solution. But one can instead consider the classic task of building a spanning tree of the graph. Then in addition to pointers to their parents in the tree the nodes will remember their distance to the root, and this will allow them to check that the set of pointers is acyclic (because distances allow detecting cycles).

Proof-labeling schemes and locally checkable proofs. The notion of proof-labeling schemes has been introduced by [50] with the objective of studying the verification phase in details, independently of the self-stabilizing context. We will give precise definitions in the next section, but for now, let us say that a proof-labeling scheme is a certification scheme like the one presented in the introduction, described as a pair of algorithms: a (possibly global) algorithm that assigns the certificates and a local algorithm

that can verify this certificate assignment. The paper [50] introduced many techniques and problems, and basically started the field of local certification. (The certification of acyclicity in the introduction is described in [50], but originates from [2].) An important result that was proved in [48] is that certifying a minimum spanning tree takes certificates of $\Theta(\log^2 n)$ bits. Later, [45] introduced a generalization of proof-labeling schemes, under the name of *locally checkable proof*. The main differences are that the verification can be done at constant distance (instead of distance exactly 1) and that a node can access the identifiers of its neighbors (again, we will give formal definitions in the next section). The most important results of [45] are lower bound techniques adapted to this more general framework. An interesting topic, given the historical origin of certification, is to compare the space (in terms of number of bits) needed for certification with the space needed for a full *silent* self-stabilizing algorithm, which would basically build and certify a solution. The former is naturally a lower bound for the latter, but in [14], the authors prove that the two quantities are actually asymptotically equal. That is, one can design self-stabilizing algorithms that do not use more space than the space needed for certification. Unfortunately, the construction of [14] requires exponential stabilization time.

Distributed decision. Around 2012, the idea of building a complexity theory for distributed decision problems was introduced in [39]. As local certification can be seen as a kind of analogue of non-determinism for decision tasks, it naturally defines an analogue of the complexity class NP. Several papers built on this idea, introducing analogue classes for randomized decision, interactive proofs, hierarchies etc. These classes actually have several variants depending on the parameters of the model (*e.g.* identifiers and certificate size). Section 5 is entirely devoted to this approach.

A lot of results have been obtained after the seminal papers cited in this section. We will review some of them in the following sections, and in Section 6 will discuss the most recent works.

3 Definitions and vocabulary

In this section, we give more formal definitions for the concepts mentioned so far.

Graph notions, identifiers, languages, and decision mechanism. A graph $G = (V, E)$ is defined by a set of vertices V, and a set of edges $E \subseteq V \times V$. For distributed graph algorithms, the classic setting is to consider that the network is represented by a connected graph (that is, for every pair of vertices u, v, there is a path in the graph from u to v), with no self-loops (that is, no edge of the form (u, u)). The number of nodes is denoted by n. This graph can come with node and edge inputs. A graph, possibly with inputs and outputs, is called a *configuration*. The neighborhood at distance T of a node v is the subgraph induced by all the nodes at distance at most T from v.

In the remaining, we will always assume that the nodes have distinct *identifiers* (or ID for short) on $O(\log n)$ bits. A graph with identifiers (and possibly inputs and outputs) is an *instance*. (Some papers in the area deal with networks without identifiers, or consider other identifiers model, but we focus on the standard model.)

We have said that the goal of certification is to allow to check that the configuration satisfies some property. One might also use the word *predicate* to refer to a more formal description of a correct configuration. In general, we follow the literature on centralized computing and consider a *language*: the set of all the correct configurations. Note that we use the word "configuration" here and not "instance" ; this is because the languages we consider should not refer to the identifiers. For example, a set of graphs

where the node with identifier 1 has some special property will not be considered as a language in this paper. Now, given an instance (that is a graph with identifiers), we will say that it is a *yes*-instance, if when we erase the identifiers, the configuration is in the language considered. Otherwise, the instance is a *no*-instance.

For distributed decision problems, we want to *accept* the *yes*-instances, and to *reject* the *no*-instances. Every node is taking its own local decision, *accept* or *reject*, and these decisions are gathered into one global decision. Namely, the instance is (globally) accepted if and only if all nodes (locally) accept. We will refer to this mechanism as the *decision mechanism*.

Self-stabilization in the state model. We have mentioned that the historical origin of local certification is in self-stabilization. Actually, there are several models of self-stabilization, and the one that we were implicitly referring to is the *state model* [20].

Definition 1 (State model of self-stabilization (partial definition))**.** *In the state model, every node of the network has a register. Every node has read/write access to its own register, and read access to the registers of its neighbors. In one atomic step, a node can read its register and the ones of its neighbors, perform computation, and update its register. The register contains all the variables used by the algorithm, including the program counter. It neither contains the identifier of the node, nor the program itself. The register has a special variable which eventually contains the output of the computation.*

In this model, an algorithm for some task is self-stabilizing if, starting from an arbitrary collection of registers, the system reaches the set of correct configurations (that is, configurations where the output variables form a solution of the task), and stays in this set.

Note that the definition above is not complete. For simplicity, we have only detailed the parts of the state model that are relevant for local certification. In particular, we have not mentioned the precise model of identifiers (which is the same as for certification), have not defined convergence, and have not stated when the nodes can apply a rule (which are two rather subtle issues that are not essential for certification). We refer to [20] for a full definition.

Models of local certification. There exists several precise models of local certification. The following definition is a common framework for these different models.

Definition 2. *A local certification of a given language is described by a distributed algorithm that has the following behavior:*

- *On a yes-instance, there exists a labeling of the nodes such that all nodes accept.*
- *On a no-instance, for any labeling of the nodes, at least one node rejects.*

The labels are usually called *certificates* or *(local) proofs*. The different forms of local certification correspond to different requirements on the algorithm. We now list the three main such forms.

Definition 3 (Proof-labeling scheme [50])**.** *A proof-labeling scheme is a local certification where the algorithm has access to its identifier, its input (if there is one), its certificates, and the certificates of its neighbors.*

Definition 4 (Locally checkable proofs [45])**.** *A locally checkable proof is a local certification where the node has access to all the information available in its neighborhood at distance T, for some constant T: all the identifiers, inputs, certificates, and the structure of the graph.*

Definition 5 (Non-deterministic local decision [39])**.** *A non-deterministic local decision scheme is a proof-labeling scheme where the certificates should not depend on the identifier assignment.*

Note that the literature is not always consistent with respect to the use of these three terms. Also, the definition given here are not written in the same way as in the original papers.

A useful scenario to reason about certification is with a prover and a verifier. The prover assigns the certificates. It has unlimited power, and wants the nodes to accept, independently of whether the instance is a *yes*-instance or a *no*-instance. The distributed algorithm is a distributed verifier that differentiates between the *yes*-instances where the prover is helping, and the *no*-instances where the prover is trying to fool the nodes.

Definition 6 (Certificate size)**.** *The certificate size of a given labeling of a graph is the size of the largest label. Given a local certification, the certificate size on a given* yes*-instance is the smallest certificate size of a labeling that makes all nodes accept. Now the certificate size of a local certification is a function f of n, such that for every graph size n, $f(n)$ is the maximum over all the yes-instances of size n for the certificate size for this instance.*

We will say that a language has optimal certificate size $\Theta(f(n))$ if there exists a local certification with certificate size $O(f(n))$ and there is no local certification with certificate size $o(f(n))$. Note that we have not been precise about which model of local certification we consider here. This is because for most languages, we can get the best of both worlds: an upper bound in the most constrained model (proof-labeling schemes) and a matching lower bound in the most general model (locally checkable proofs). (Non-deterministic local decision has a strictly weaker power in the sense that it can decide a set of languages that is strictly smaller, and it is usually not considered when studying specific languages.)

A concept close to local certification is the notion of *advice*. In a distributed environment, an advice also comes as an assignment of labels to the nodes, but these labels 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, [34, 36, 35].

4 Certification size: bounds and techniques

In this section, we review the bounds on the certification size, and the techniques used establish these bounds. We first show that every language can be certified. The challenge is then to establish what is the optimal certification size for interesting languages. We describe the classic upper bound techniques in Section 4.2 and lower bound techniques in Section 4.3. We also provide a table of some important results about certification size in Section 4.4.

4.1 Universal certification

It is known that every language can be certified [49]. The technique to prove this statement is sometimes called the *universal certification*, and was first mentioned in [49]. On *yes*-instances, the prover provides the whole adjacency matrix of the graph, with the correspondence between the rows and the identifiers of the nodes (and also the input assignment if there are inputs). 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 instance described in the certificates is indeed the real instance. Finally, all the nodes check, in parallel and without communication, that the configuration is in the language. This scheme uses certificates of size $\Theta(n^2)$. (If there are inputs on the nodes or edges, then

these also have to be encoded and take additional space. As these are typically of constant size, the size is still in $\Theta(n^2)$.)

4.2 Upper bounds techniques

The techniques used in certification depend of course on the language being certified. Nevertheless, we can identify a few techniques that are very common.

Basic techniques. The definition of some languages basically describes their certification. For example, to certify that a graph can be properly k-colored, one just has to label the nodes with their colors, as the nodes can check locally that the coloring is correct. More generally, for any LCL problem (as defined in Section 2) the set of graphs that admit a solution can be certified by simply writing the solution in the certificates. Another property that is easy to certify is the fact that some node has some special role. More concretely, consider that the nodes are given a bit as input, and that we want to certify that at most one node is given 1. One can basically give the identifier of the selected node to all nodes. More precisely, consider the following local decision algorithm on a node v:

1. If the certificate of a neighbor of v is different from the certificate of v, then reject.
2. If v has input 1, and the certificate is not the identifier of v, then reject.
3. Otherwise, accept.

On a *yes*-instance, either there is one node with a 1, and the prover labels every node with the identifier of this node, or there is no such node, and the prover can give empty labels. In both cases, all nodes accept. On a *no*-instance, there are at least two nodes with a 1. Then there are two situations. First, one of them does not see its identifier in the certificate, and then it rejects. Second, they both see their own identifier in their certificate. Then we can partition the graph into several non-empty regions where the nodes have the same certificate, and by connectivity, there exists an edge that has two endpoints that do not have the same certificate. (Remember that the identifiers are all distinct.) These nodes reject because they do not have the same certificate.

Spanning trees. A central tool in certification is the one of spanning trees. Consider that the graph is given as input a binary labeling of the edges. We want to check that the selected edges (the ones labeled with 1) form a spanning tree of the graph. On *yes*-instances, the prover can assign the certificates in the following way [1].

1. Chose a node to be the root of the tree.
2. Write the identifier of the root in all certificates.
3. Give to every node its distance to the root in the spanning tree.

The local decision algorithm is the following.

1. Check that the root identifier is the same that the one given to the neighbors.
2. If the distance is 0, check that the root identifier of the certificate is the same as the identifier of the node.

3. If the distance is 0, also check that all neighbors linked by a selected edge have distance 1.

4. If the distance d is not 0, check that *among the neighbors to which the node is linked by selected edges*, there is exactly one node with distance $d-1$, and the other such neighbors have distance $d+1$.

It is clear that on a *yes*-instance, all nodes accept. For a *no*-instance, consider first the case where the set of selected edges do not form an acyclic subgraph. In this case, consider a cycle C of selected edges. In C consider a node v with the largest distance. This node has two neighbors in C and none of them has a strictly larger distance. Then v rejects, either in step 3 if it has distance 0, or in step 4 if it has distance strictly larger. Now consider the case where we have a *no*-instance with an acyclic but disconnected set of selected edges. In every tree, at least one node must have distance 0, otherwise one node would reject at step 4. And if several nodes have distance 0, we are back to the case we described above: thanks to the root identifier given to every node, at least one node should reject. This local certification uses certificates of size $O(\log n)$ because the distances are between 1 and n, and the identifiers are assumed to be encoded on $O(\log n)$ bits. Spanning trees are everywhere in local certification because they allow to certify that there is a unique node having some property, which is very useful. To certify this, on *yes*-instances, the prover chooses a spanning tree rooted at the special node, gives to every node the identifier of its parent in the tree, and certifies the spanning tree as explained above. Then the root checks that indeed it has the property, and all the other nodes check that they do not have this property. A spanning tree also allows some counting in the graph [50]. For example, suppose once again that the nodes have a bit as input, and that we want to check that there are k 1s, for some k specified in the language. The prover can describe and certify a spanning tree, and give to every node v a counter, which contains the number of 1s there is in its subtree of the spanning tree rooted at v. Then the nodes can check the consistency of the counters: the counter of a node should be the sum of the counters of its children, plus its own input bit. The root can check that its counter is k. Using this tool, we can, for example, certify the number n of nodes in the graph, or the size of a structure that can be checked locally, such as a clique, an independent set, or a matching.

Encoding the run of an algorithm. A technique that already appears in the original proof-labeling scheme paper [50] consists in encoding the run of an algorithm. Consider for example the problem of certifying that a set of edges forms a minimum spanning tree, and suppose you have a distributed algorithm that computes this solution. Then a possible certification consists in encoding at every node, every step of the algorithm. For example, at round 1, the algorithm sent this message to that neighbor and received that other message from that other neighbor, and then at the second round this and that happened etc. Given the code of the distributed algorithm, the node can always check that the run is correct and consistent (the messages that are said to be sent by a node are written as received by the other nodes etc.). If every node is happy with the run, and if the run indeed selects the edges of the input, then the input describes a minimum spanning tree. In general, this technique is not very efficient in terms of certificate size. This is because a lot of information exchanged during the algorithm is redundant or just useless for the verification of the solution. Nevertheless, this approach can be fruitful. For example, one can certify some data structures that are used to build the solution, without explicitly writing all the messages sent to build this structure. For the problem of minimum spanning tree, this technique is actually optimal (for the usual weight range) [50]. See Figure 3 for an overview of this certification.

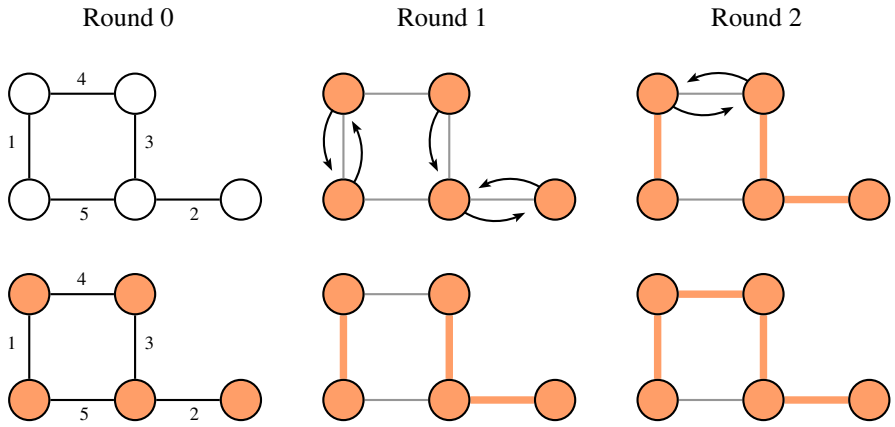


Fig. 3: Several minimum spanning tree (MST) algorithm are based on merges of so-called fragments, from the historical Borůvka algorithm [58] to the celebrated GHS algorithm [43]. The run of such an algorithm is described in this figure. Very roughly, every node starts as its fragment, and at each round, each fragment proposes to merge with the neighboring fragment to which it has the lightest edge. At the end of the process, these merging edges form a minimum spanning tree. As the number of fragments is at least halved at each step, there are at most $O(\log n)$ rounds. In other words, a node belongs to at most $O(\log n)$ successive fragments. The certificate at some node v consists in $O(\log n)$ fields, one for each fragment the node belongs to. For each field, the node is given a pointer to a neighbor (in the MST), a distance and an identifier, such that collectively the certificates of the nodes of a fragment form a spanning tree. This spanning tree is pointing to the node through which the next merge will be performed. Also, every node is given the weight and name of the successive merge edge of its fragments. This is enough for certification: the nodes can check the fragment structure and also the fact that the merges happen on the lightest outgoing edges.

Duality. When it comes to certifying some combinatorial structures, one can sometimes use duality. Let us take the example of maximum matching in bipartite graphs [45]. König's theorem states that on bipartite graphs, the size of the maximum matching is equal to the minimum size of a vertex cover. We can be more precise and say that a matching is maximum if and only if there exists a vertex cover that consists in one endpoint of each edge of the matching. Then, to certify that a set of selected edges is a maximum matching, the prover can simply put a bit on each vertex saying whether it is in a vertex cover. The nodes can then check locally the vertex cover, the matching, and the fact that every edge of the matching has one node of the vertex cover. And this is enough. On this example, the duality was implicit. To certify a maximum weight matching in a weighted bipartite graph, one explicitly writes the values of the primal and dual variables [45]. Actually similar but more involved techniques allow one to certify maximum cardinality matchings and 2-approximation of maximum weight matching in general graphs [16]. A lot of other approximation problems have been certified using this technique [22] (we will come back to this in the last section).

Other techniques. Other techniques have been used for specific problems, that could possibly be used in other problems. For example, the certification of minimum spanning tree in $O(\log n \log W)$ bits for weights in $[1, W]$ from [48] uses a variety of ideas. For example, when using some k spanning trees to

point to k different nodes, instead of using $k \times \Theta(\log n)$ bits, one can use one spanning tree on $O(\log n)$ bits, and then use it to encode the other spanning trees using only $O(1)$ bits per additional tree. See [23] for an explanation of the other tricks of [48]. In general, techniques from other labeling problems (distance labeling, adjacency labeling) can also be useful in certification.

4.3 Lower bound techniques

There are basically two types of techniques used for lower bounds: modifications of one or several instances to get to a contradiction via a counting argument, and reductions from communication complexity (that can be explicit or implicit). The first one works for $\Omega(\log n)$ bounds, and the second is more adapted for higher lower bounds. An exception is [48], where the authors use techniques tailored for the minimum spanning tree problem.

Crossing technique. In terms of lower bounds, there is a difference between proof-labeling schemes and locally checkable proofs. It is easier to derive lower bounds in the first model because the nodes are less powerful, thus can be fooled more easily. A technique for lower bounds in proof-labeling schemes is the crossing technique. In this technique, we take a *yes*-instance and show that we can modify it in such a way that (1) the nodes do not detect a change (and then continue to accept) and (2) the instance is now a *no*-instance [50]. See Figure 4 for an example. Unfortunately, this technique does not work in the more

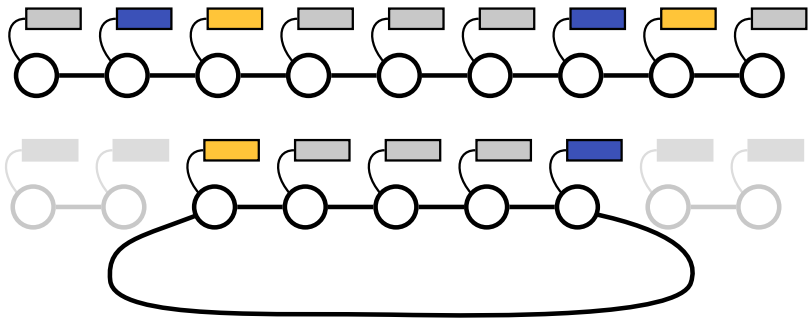


Fig. 4: This figure illustrates the *crossing technique* for the language of paths, in the proof-labeling scheme model [50]. The idea is that if $o(\log n)$-bit certificates are used, then by the pigeon-hole principle (or equivalently by counting argument), there exist two edges, (a, b) and (c, d), such that a and c, respectively b and d, are assigned the same certificate. These shared certificates are colored blue and yellow in the top picture. Then we *cross* the edges: we remove (a, b) and (c, d), and add (b, c). We get a cycle, which is a *no*-instance, and some disconnected parts that can be discarded. The point is that for all nodes in the cycle, even the ones that are adjacent to the crossed edge, the view is the same as in the path, thus all nodes accept. And this is a contradiction.

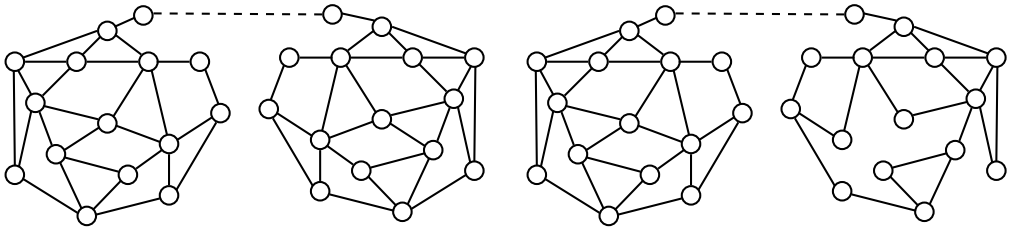
general model of locally checkable proofs. Indeed, in this later model, the nodes can see the identifier of their neighbors, then any modification of one instance implies that the node have different views, and can take different decisions.

Cut-and-plug technique. A type of technique that provides similar bounds to the crossing technique, but for locally checkable proofs, is what we call *cut-and-plug*. It consists in taking several *yes*-instances, cutting them into pieces, and proving that we can create a *no*-instance by plugging several of these pieces

together, in such a way that the nodes cannot detect it. Let us describe a general framework for this technique for the language of paths. (There exist several implementations of this general framework [45, 29].) Suppose that there exists a certification with $f(n)$-bit certificates with $f(n) \in o(\log n)$, where the nodes can see at some constant distance k (remember that the locally checkable proofs model allows this). Now consider $\Theta(n)$ paths P_i on $2k+1$ nodes, such that any identifier appears in at most one path. We will call these *chunks*, and consider that each chunk has an arbitrary orientation, in order to talk about the first node and the last node. For any given chunk, we can consider a labeled version of it, where every node v is assigned some label of size $f(n)$. For all P_i, we define $(P_{i,j})_j$ as the set of all the possible such labeled versions of it. Now consider two labeled chunks $P_{i,j}$ and $P_{i',j'}$. We say that we can plug them, and write $P_{i,j} \to P_{i',j'}$, if when we add an edge between the last node of $P_{i,j}$ and the first node of $P_{i',j'}$, and run the verification, all nodes at distance at most $k+1$ from the added edge accept (the decisions of the other nodes is not important here). Consider now a path made by concatenating some chunks P_i. As this is a *yes*-instance, there exists for each chunk P_i of the instance, a labeled version $P_{i,j}$, such that all nodes accept. Moreover, as the verifier algorithm sees only at distance k, for all consecutive labeled chunks $P_{p,q}$ and $P_{p,q'}$ we have $P_{p,q} \to P_{p',q'}$. By considering all the *yes*-instances, one can get a large collection of labeled chunks that can be plugged. One can then show via a counting argument that there exists $\ell > 2$, and a family $(P_{p^i,q^i})_i$, $i \in [1,\ell]$ such that: (1) for every $i < \ell$, $P_{p^i,q^i} \to P_{p^{i+1},q^{i+1}}$, and (2) $P_{p^\ell,q^\ell} \to P_{p^1,q^1}$. That is, there is a collection of chunks that we can plug into a cycle, and a collection of labels such that every node accepts. This is a contradiction. This technique can be implemented in different ways, depending on the definition of the chunks, and on the counting argument. See [45] and [29].

Proofs inspired by communication complexity. A classic lower bound technique in distributed computing is to design a reduction from communication complexity. In certification, a first intuition for why this can work is the following. The nodes have to take a decision concerning a global property of the network, and basically the only kind of communication they have is through the certificates. Then intuitively, if to decide whether the instance is correct or not, a lot of information has to be transferred from one side of the graph to the other, the certificates have to be large. For example, Figure 5 illustrates the language of symmetric dumbbell graphs [45, 50]. These are graphs formed by taking two copies of the same graph and linking them by a long path (that is, a path of $2k+1$ nodes for verification radius k). To certify that the graph is in this family, one basically has to write the map of the extremity graphs in the certificate. This can be proved by a counting argument. Intuitively, the nodes of the paths cannot do clever things, and we can consider that they have the same certificate. If this certificate could work for several extremity graphs, then we could take two pairs and mix them into a *no*-instance that would be accepted, and this would be a contradiction. Hence, there is basically one "path certificate" per extremity graph, which means the certificates have size $\Theta(n^2)$.

Reduction from non-deterministic communication complexity. A more advanced lower bound technique uses explicit reduction from communication complexity. The reduction is from the classic disjointness problem where each player is given a set, A for Alice, and B for Bob, and the goal is to minimize the number of bits exchanged in order for Alice and Bob to decide whether $A \cap B$ is empty. It is well-known that this problem has high communication complexity even in the non-deterministic setting, where the communication can also take the form of labels given to Alice and Bob by a prover [52]. The idea is then to prove that if a local certification with small certificates existed for the problem considered, then one could design a (non-deterministic) communication protocol for disjointness that would beat the known



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

Fig. 5: Illustration of the symmetric dumbbell graph language.

lower bounds. Basically, in this communication protocol, Bob and Alice would simulate the verification process on a specific graph depending on their inputs. Bob would be in charge of simulating half of the nodes, and Alice would be in charge of simulating the other half, and the communication would be used to exchange the certificates of the nodes on the cut between Alice's and Bob's vertices. The exact construction depends on the language studied, because the graph should encode the disjointness problem with gadgets depending on that language. To our knowledge, there are two examples of such proofs in the literature: an $\Omega(n^2/\log n)$ bound for non-3-colorability [45] and an $\Omega(n/k)$ bound for diameter at most k in [16] (see also [30]). (For an introduction to this type of techniques, see [25].)

4.4 Landscape of proof sizes

To finish this section, we give a small table of some important bounds on certification size. See Table 1.

Language	Upper bound	Lower bound	Reference
Locally checkable language	0	–	[56]
k-colorability	$O(\log k)$	–	Folklore
Maximum matching in general graphs	$O(\log n)$	$\Omega(\log n)$	[16, 45]
Spanning tree	$O(\log n)$	$\Omega(\log n)$	[50, 45]
Minimum spanning tree (weights in $[1, W]$)	$O(\log n \log W)$	$\Omega(\log n \log W)$	[48]
Diameter k	$O(n \log n)$	$\Omega(n/k)$	[16]
Non-3-colorable graph	$O(n^2)$	$\Omega(n^2/\log n)$	[45]
Symmetric graph	$O(n^2)$	$\Omega(n^2)$	[45, 50]
Any language	$O(n^2)$	–	[50]

Tab. 1: Some important certification bounds. All upper bounds hold in the proof-labeling scheme model. All lower bounds hold in the locally checkable proofs model, except the ones for minimum spanning tree and diameter that hold in the proof-labeling scheme model.

Note that the languages that require no certificate or very small ones, such as different versions of coloring, correspond to properties that are usually considered as local. On the other hand, languages that require large certificates, such as symmetric graphs, correspond to properties that are considered global. Therefore, at least on an intuitive level, the certificate size is a measure of locality: the smaller the certificates, the more local the property.

5 Distributed decision: a complexity theory point of view

As mentioned in earlier sections, one way to look at certification is to consider that it is the non-deterministic version of distributed decision. This analogy appears already in [50, 45]. In [39] the authors pushed the idea further by formally introducing several new classes, and proving inclusions between these classes. (To be precise, [39] is also different because it is about distributed decision without reference to identifiers.) As in the current paper we focused on certification, we will not give an exhaustive survey of this area, but it is nevertheless insightful to have a quick overview. We refer to [27] for a survey on distributed decision.

Deterministic and probabilistic distributed decision. As mentioned in the introduction, there are languages that can be recognized locally. One example is the set of graphs properly colored with k colors. These languages form the class of local decision LD [39], which is basically the same as the classic LCL [56]. Many papers are interested in constructing solutions for such languages (problems such as $(\Delta+1)$-coloring, maximal matching, independent set). Actually, there is now a complexity theory for the construction problems whose solutions are LCL, see for example [63]. Note that this is not the same as what we are describing now: we are currently interested in decision problem, not construction. Unlike in centralized computing, as far as we know, the two domains have little in common. Deterministic decision (that is, local decision without certificates) only captures a rather small set of languages, and [39] introduced the idea of looking at a probabilistic version. It consists in allowing nodes to use random bits when taking decision, with the goal that on *yes*-instances, there is a good probability of acceptance, and on *no*-instances, there is a good probability of rejection. Note that in probabilistic decision, like in deterministic decision, there are no certificates. The classic example for probabilistic decision in the language where nodes are given binary labels, and at most one node should have a 1 [39]. Clearly, this cannot be decided with a deterministic decision. Here is a distributed probabilistic decision algorithm for this problem. Every node with a 0 accepts, and every node with a 1 accepts with probability $p = (1+\sqrt{5})/2$. On instances where there is no 1, this algorithm is always correct. On instances where there is one 1, it is correct with probability p. On instances where there are more than one 1, the algorithm should reject, and this happens with probability at least $1-p^2$, which for our choice of p is also $(1+\sqrt{5})/2$. That is, this decision algorithm is correct with probability $(1+\sqrt{5})/2$. One should note that unlike what happens in centralized computing, the precise thresholds is important, as boosting is not always possible [40]. Indeed, in centralized computing, the classic boosting technique consists in running the probabilistic decision several times, and taking the most frequent outcome, which it is not possible in the distributed local setting, as the nodes do not know the outcome of the decision process.

Probabilistic decision is useful even outside the complexity perspective, as it gives a natural candidate when one wants to generalize results from LCL to a larger class. For example, a seminal result in the LOCAL model is that constant-time randomized algorithms for LCL can always be derandomized [56], and this result has been generalized to languages that can be decided via a probabilistic decision [26].

Non-deterministic distributed decision. As said, certification is a form of distributed non-determinism. Remember that the complexity class NP has two classic definitions: the one with a non-deterministic Turing machine, and the one where polynomial size certificates are assigned by a prover [6]. The second definition is clearly the one for which the analogy is the most natural. Note that for the class NP, both the size of the certificates and the computational power are limited. For now, we have only mentioned restrictions on the analogue of the computational power, which in our model is the size of the view of a

node, and we have set this to constant. As for the certificate sizes, instead of limiting it, we have measured how large they need to be for various languages. In [45], the authors defined LogLCP as the class of languages that have certificates in $O(\log n)$, and argue that this forms a meaningful class. Indeed, there are many languages that fall into this class, and we know very few languages between certificates of size $O(\log n)$ and $\Omega(n)$. (Basically only minimum spanning tree is a natural such languages).

Hierarchy and interactive proofs. As in centralized computing, we can consider more elaborate computational settings than non-deterministic distributed decision. In this direction, [32, 11] introduced analogues of the polynomial hierarchy in complexity theory (see *e.g.* [6]). The idea is that in addition to the prover trying to convince the nodes that the instance is a *yes*-instance, there is disprover trying to convince the nodes that the instance is a *no*-instance. The certificates given by these players are of size $O(\log n)$, and they assign them to the node one after the other. For example, on the third level of the hierarchy, the nodes first receive certificates from the prover, then receive certificates from the disprover (that are somehow answers to the prover's certificates), and then again from the prover. The number of alternations defines a hierarchy. An analogue of Arthur-Merlin interactive proofs has also been proposed in [47]. There the setting is that there is a unique prover, but the nodes have access to randomness, and there can be several rounds of communication between the prover and the nodes. This model provides a better understanding of problems such as symmetric dumbbell graphs in Section 4.3, and follow-up works have shown that powerful tools can be designed for this setting, with trade-offs between the different resources (number of bits exchanged, number of prover-nodes communication rounds etc.) [18, 57, 55].

6 Research directions

We now list some research directions, mentioning recent results and some open questions.

6.1 New upper and lower bounds

An evident research objective is to establish the optimal certification size of any interesting language. For classic problems of distributed computing, such as tree structures or connectivity questions, tight bounds have been known for a decade or more now, but we are far from a complete understanding of this type of question. Let us start with two simple open questions.

Open problem 1. *The property of k-colorability is easy to certify with $O(\log k)$ bits, just by writing the colors. Can we prove a matching lower bound?*

Open problem 2. *In [16], the authors show that the optimal size for certifying diameter at most k is in $\Omega(n/k)$ and $O(n \log n)$. What is the optimal certification size?*

For a lot of problems, either we have no idea what the certification size is, or we have polylogarithmic certification. The only polynomial lower bounds we have are for symmetric graphs, 3-non-colorability [45], diameter [16] and for the uniqueness language (where all nodes should have different inputs) [50]. Hence, the following problem:

Open problem 3. *What are other interesting languages with polynomial optimal certification size?*

A recent topic in certification is the idea of approximate certification. This was introduced by [16], with the following striking example. With the classic definition, certifying that the diameter is at most k takes $\Omega(n/k)$ bits. Now, if we relax the task, and ask that all graphs of diameter at most k are accepted, and all graphs of diameter at least $2k$ are rejected, then the certificate size drops to $O(\log n)$ bits. One can

consider a lot of problems from the approximation perspective: [16] also studied maximum matching and [22] made a systematic use of primal-dual methods for various optimization problems. The paper [22] also introduced the idea of limiting the computational power of the prover and the verifier, and deriving conditional lower bounds from hypothesis such as P$\neq$NP. It is also interesting to think about what can be certified outside the classic graph problems (coloring, spanning trees, matching etc.). For example, [10] introduced the topic of certifying routing tables. It is also very interesting to study the properties of the graph itself, and then one can see the language as a graph class. There are many interesting graph classes, and establishing the certificate size for those is a vast program. In [33] and [31], the authors proved that certifying planarity and embeddability on bounded genus graphs can be done with certificates of size $O(\log n)$. Note that these classes can be characterized by forbidden minors, and that the language that we know have large certificates (*e.g.* diameter k) do not have such characterizations. The following question appears natural.

Open problem 4. *Can we certify all graph classes characterized by forbidden minors with certificates of size $O(\log n)$?*

For the question above, even if the answer is positive, the constant of the asymptotics will probably depend on the size of the excluded minors. This brings us to the next question: what happens if we express the certification size not only as a function of the size n but also as a function of other parameters? In other words, what is the parameterized complexity of distributed certification? There exists some bounds using such parameters, but we could use those much more systematically. Natural parameters are the diameter, the maximum degree, the arboricity, the number of edges and the treewidth. Note that taking this point of view would require new techniques, not only in upper bounds but also in lower bounds, where (depending on the parameters) one would not be able to use the classic ring or small cuts graphs, that are used respectively for cut-and-plug and for reduction from communication complexity. Here are two concrete problems.

Open problem 5. *Establish the optimal certification size of the diameter as function of the number of nodes n and of the diameter k. Establish the optimal certification size of bounded-genus graphs as function of n and of the genus g.*

Some certification work only in some specific graph class, for example, maximum weight matching in bipartite graph [45] and diameter on trees [50]. It is a natural research direction to try to prove such results, and to find where the limit is between compact and polynomial certification.

Open problem 6. *For the languages that have polynomial optimal certification, find a large graph class in which this language can be certified with polylogarithmic certificates.*

Finally, even for bounds we know, it would sometimes be nice to have an alternative proof.

Open problem 7. *Is there a simpler proof for $\Omega(\log n \log W)$ bound for minimum spanning tree established in [48]?*

6.2 Understanding certification

The study of the local certification is not limited to establishing upper and lower bounds on the certificate size. Another way to gain understanding on the notion is to modify the model, for example by increasing or reducing the resources allowed.

First, let us consider the communication resources. In standard proof-labeling schemes, the communication model follows the state model of self-stabilization: a node can see the states of its direct neighbors.

This can be seen as an abstraction of a model where every node sends its states to all its neighbors at every round. This aspect can be modified in at least two ways. First, in [60], the authors proposed to consider a message-passing model, with different messages for different neighbors. Second, in [42], randomization was used to reduce the size of the messages.

Still on the communication side, another line of work proposes to increase the view of the nodes beyond constant [59, 30] (see also [51]). In other words, these papers study the impact of a larger verification radius on the certificate size. The following question is still open, although partial positive answers have been obtained.

Open problem 8. *Can we always achieve a linear trade-off between the certificate size and the verification radius?*

It is also relevant to challenge the certification side of the model. In [29], the certification is not local, that is with one certificate per node, but global, that is, all nodes can access the same unique certificate. A simple open question in this model is:

Open problem 9. *Show that certifying bipartiteness requires a global certificate of size* $\Omega(n \log n)$.

Interestingly, the results about larger verification radius and global certificates shed a new light at the notion of redundancy in certification. On the one hand, the fact that increasing the radius allows to decrease the size of the proofs indicates that there is some redundancy in the certificates. For example, it is not the case that in total we need $n \times O(\log n)$ bits to certify a spanning tree, because if we allow a view at logarithmic distance, [30] proved that we are fine with $O(1)$-bit certificates. On the other hand, this redundancy is not global, in the sense that it is not the case that the exact same information is copied in every certificate. Indeed, [29] proved that a global certificate for spanning tree cannot be smaller than the collection of all the local certificates, that is, must be of size $\Omega(n \times \log n)$.

Yet another direction to modify the model is to restrict the computational power of the nodes. In this direction, a fruitful approach is to use modal logic to characterize the languages that can be recognized. See for example the thesis [62], and the references therein. See also [22] for an example where limiting the computational power to polynomial-time allows deriving better lower bounds.

Finally, we have seen that the decision mechanism "accept globally if and only if every node accepts locally" [2] is justified by the origins of certification in the context of self-stabilization. But it is reasonable to ask about other models where the mechanism would be different. For example, [4] and [5] investigated contexts where the nodes can output more than one bit, that is, the set of possible outputs is strictly larger than just *accept* or *reject*. The global decision is then an arbitrary function of these outputs. An other example where the decision mechanism is a bit different is [28], where it is required that in instances that are far from being in the language, many nodes should reject.

6.3 Complexity theory

As described in Section 5, there has been efforts to build a complexity theory for distributed decision. Here is one open question from one of the papers we cited.

Open problem 10. *Prove that the decision hierarchy of [32] is infinite.*

Note that there are evidence that this question might be a very hard [29]. We have also mentioned distributed Arthur-Merlin protocols introduced in [47]. This is a recent and active area of research. We refer to [57] for a list of nice open questions. Let us just mention that an important question is to establish the best trade-off possible between interaction and communication. As we gain knowledge

about distributed complexity classes, one might want to go one step further and ask to characterize such classes. Here are two such questions.

Open problem 11. *Can we characterize the languages that have a probabilistic decision scheme? What about the ones that have* $\Theta(\log n)$ *certification?*

Finally, we have seen in the definitions that there are different types of local certification: proof-labeling schemes [50], locally checkable proofs [45] and non-deterministic distributed decision [39]. One of the differences between these models is the way they handle identifiers. This difference in turns implies different associated complexity classes. For example, [32] and [11] study hierarchies in two models that differ by their use of the identifiers, and these hierarchies end up being completely different. The impact of the precise model of identifiers on decision can be a pretty subtle issue, as shown for example in [41, 37, 38].

6.4 Certification in self-stabilization

Originally, certification has been studied in the context of self-stabilization, and can be identified as a component of silent self-stabilizing algorithms. There are still interesting interactions between the two. First, the usefulness of studying certification to better understand self-stabilization has been illustrated recently, with the celebrated minimum spanning tree certification of [48] being embedded into a full self-stabilizing algorithm [15]. A surprising question actually arises from [15]. In (silent) self-stabilization[(i)], a typical algorithm would keep some pieces of information during the computation of a solution, and these pieces will serve as a certificate. Instead, in [15] a solution is first built, and only then certificates are computed. There are evidences that this modular approach is necessary [24], but nothing has been formally proved, and we even lack a precise definition.

Open problem 12. *Formalize the notion of* certification during computation*, and show that it cannot be achieved for minimum spanning tree in space* $O(\log n \log W)$.

In [14], the authors proved that one can always design a silent self-stabilizing algorithm that does not use more than the space needed for certification. Unfortunately, this general transformation takes exponential convergence time in general. Hence, the following general question.

Open problem 13. *In the state model of self-stabilization, when can we use optimal space (*i.e. *the space needed for certification) and still have polynomial convergence time?*

It is proved in [13], that this can be done for several variants of spanning trees.

Conclusion In this paper, we introduced the domain of local certification, through its history and techniques. As we have seen in the last section, there is still a lot of open questions to answer and interesting directions to follow. We hope that this document will foster new research on the topic.

Acknowledgements

We would like to thank to the reviewers for their careful reading and suggestions that helped a lot in improving the paper. Thanks to Fabien Dufoulon for several useful remarks, to Tatiana Starikovskaya

[(i)] Note that we highlight the relation between silent self-stabilization and certification, because the two are very close, but non-silent self-stabilization also uses some ideas that we discuss, including local decision.

for inviting me to give a seminar which inspired Section 6, to Ami Paz for pushing me to publish this document, and to the community of local certification for all the works in this domain.

References

[1] Yehuda Afek, Shay Kutten, and Moti Yung. Memory-efficient self stabilizing protocols for general networks. In *Distributed Algorithms, 4th International Workshop, WDAG '90*, volume 486, pages 15–28, 1990. doi:10.1007/3-540-54099-7_2.

[2] Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theoretical Computer Science*, 186(1-2):199–229, 1997. doi:10.1016/S0304-3975(96)00286-1.

[3] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

[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 2013*, 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 2014*, pages 15–28, 2014. doi:10.1007/978-3-319-12340-0_2.

[6] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0-521-42426-4.

[7] Baruch Awerbuch and George Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 258–267, 1991. doi:10.1109/SFCS.1991.185377.

[8] 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 1991*, pages 268–277, 1991. doi:10.1109/SFCS.1991.185378.

[9] Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and global reset (extended abstract). In *Distributed Algorithms, 8th International Workshop, WDAG '94*, volume 857, pages 326–339, 1994. doi:10.1007/BFb0020443.

[10] Alkida Balliu and Pierre Fraigniaud. Certification of compact low-stretch routing schemes. *Comput. J.*, 62(5):730–746, 2019. doi:10.1093/comjnl/bxy089.

[11] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? *Journal of Computer and System Sciences*, 97:106–120, 2018. doi:10.1016/j.jcss.2018.05.004.

[12] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013. ISBN 9781627050180. doi:10.2200/S00520ED1V01Y201307DCT011.

[13] 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 2015*, pages 589–598, 2015. doi:10.1109/ICDCS.2015.66.

[14] 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 2014*, pages 18–32, 2014. doi:10.1007/978-3-319-11764-5_2.

[15] Lélia Blin, Swan Dubois, and Laurent Feuilloley. Silent MST approximation for tiny memory. In *Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020*, volume 12514, pages 118–132, 2020. doi:10.1007/978-3-030-64348-5_10.

[16] Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theor. Comput. Sci.*, 811:112–124, 2020. doi:10.1016/j.tcs.2018.08.020.

[17] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control.*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

[18] Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. Trade-offs in distributed interactive proofs. In *33rd International Symposium on Distributed Computing, DISC 2019*, pages 13:1–13:17, 2019. doi:10.4230/LIPIcs.DISC.2019.13.

[19] Edsger W Dijkstra. Self-stabilization in spite of distributed control. In *Selected writings on computing: a personal perspective*, pages 41–46. Springer, 1982.

[20] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000. ISBN 0-262-04178-2. URL `http://www.cs.bgu.ac.il/%7Edolev/book/book.html`.

[21] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. *Acta Informatica*, 36(6):447–462, 1999. doi:10.1007/s002360050180.

[22] Yuval Emek and Yuval Gil. Twenty-two new approximate proof labeling schemes. In *34th International Symposium on Distributed Computing, DISC 2020*, volume 179, pages 20:1–20:14, 2020. doi:10.4230/LIPIcs.DISC.2020.20.

[23] Laurent Feuilloley. Note on distributed certification of minimum spanning trees, 2019. arXiv:1909.07251.

[24] Laurent Feuilloley. Can we always build and certify at the same time? (Introduction). Discrete notes. `https://discrete-notes.github.io/build-certify-1`, 2020.

[25] Laurent Feuilloley. Diameter lower bound in local certification (1). Discrete notes. `https://discrete-notes.github.io/diameter-lower-bound-1`, 2021.

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

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

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

[29] Laurent Feuilloley and Juho Hirvonen. Local verification of global proofs. In *32nd International Symposium on Distributed Computing, DISC 2018*, pages 25:1–25:17, 2018. doi:10.4230/LIPIcs.DISC.2018.25.

[30] Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. *Distributed Computing*, 2020. doi:10.1007/s00446-020-00386-z.

[31] Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Eric Rémila, and Ioan Todinca. Local certification of graphs with bounded genus, 2020. arXiv: 2007.08084.

[32] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. *Theor. Comput. Sci.*, 856:51–67, 2021. doi:10.1016/j.tcs.2020.12.017.

[33] Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Compact distributed certification of planar graphs. *Algorithmica*, pages 1–30, 2021.

[34] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Oracle size: a new measure of difficulty for communication tasks. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006*, pages 179–187. ACM, 2006. doi:10.1145/1146381.1146410.

[35] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with advice. *Inf. Comput.*, 206(11):1276–1287, 2008. doi:10.1016/j.ic.2008.07.005.

[36] 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.

[37] 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 2012*, pages 224–238, 2012. doi:10.1007/978-3-642-35476-2_16.

[38] 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.

[39] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *Journal of the ACM*, 60(5):35, 2013. doi:10.1145/2499228.

[40] 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.

[41] Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. *Theortical Computer Science*, 751:61–73, 2018. doi:10.1016/j.tcs.2017.01.011.

[42] Pierre Fraigniaud, Boaz Patt-Shamir, and Mor Perry. Randomized proof-labeling schemes. *Distributed Comput.*, 32(3):217–234, 2019. doi:10.1007/s00446-018-0340-8.

[43] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983. doi:10.1145/357195.357200.

[44] Andrew V. Goldberg and Serge A. Plotkin. Parallel ((greek d)d+1)-coloring of constant-degree graphs. *Inf. Process. Lett.*, 25(4):241–245, 1987. doi:10.1016/0020-0190(87)90169-4.

[45] 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.

[46] Juho Hirvonen and Jukka Suomela. Distributed algorithms 2020, 2020.

[47] Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. Interactive distributed proofs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018*, pages 255–264, 2018.

[48] 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.

[49] 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*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

[50] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22 (4):215–233, 2010. doi:10.1007/s00446-010-0095-3.

[51] 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.

[52] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. ISBN 978-0-521-56067-2.

[53] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[54] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

[55] Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Shared vs private randomness in distributed interactive proofs. In *31st International Symposium on Algorithms and Computation, ISAAC 2020*, volume 181 of *LIPIcs*, pages 51:1–51:13, 2020. doi:10.4230/LIPIcs.ISAAC.2020.51.

[56] Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[57] Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1096–115, 2020. doi:10.1137/1.9781611975994.67.

[58] Jaroslav Nesetril, Eva Milková, and Helena Nesetrilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discret. Math.*, 233(1-3):3–36, 2001. doi:10.1016/S0012-365X(00)00224-7.

[59] Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017*, pages 53–70, 2017. doi:10.1007/978-3-319-72050-0_4.

[60] 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*, pages 1–17, 2017. doi:10.1007/978-3-319-69084-1_1.

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

[62] Fabian Reiter. *Distributed automata and logic*. PhD thesis, University Paris Diderot, 2017. arXiv:1805.06238.

[63] Jukka Suomela. Landscape of locality (invited talk). In *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020*, volume 162, page 2:1, 2020. doi:10.4230/LIPIcs.SWAT.2020.2.