Journal of Computer and System Sciences 97 (2018) 106–120

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

What can be verified locally? ☆

Alkida Balliu [a,1,*], Gianlorenzo D'Angelo [b], Pierre Fraigniaud [c,2], Dennis Olivetti [a,1]

[a] *Aalto University, Finland*
[b] *Gran Sasso Science Institute, Italy*
[c] *CNRS and University Paris Diderot, France*

ARTICLE INFO

Article history:
Received 25 September 2017
Received in revised form 14 May 2018
Accepted 25 May 2018

Keywords:
Distributed computing
Decision problems
Local model

ABSTRACT

In the framework of *distributed network computing*, it is known that not all Turing-decidable predicates on labeled networks can be decided *locally* whenever the computing entities are Turing machines (TM). This holds even if nodes are running *non-deterministic* Turing machines (NTM). In contrast, we show that every Turing-decidable predicate on labeled networks can be decided locally if nodes are running *alternating* Turing machines (ATM). More specifically, we show that, for every such predicate, there is a local algorithm for ATMs, with at most two alternations, that decides whether the actual labeled network satisfies that predicate. To this aim, we define a hierarchy of classes of decision tasks, where the lowest level contains tasks solvable with TMs, the first level those solvable with NTMs, and the level $k > 1$ contains those tasks solvable with ATMs with $k - 1$ alternations. We characterize the entire hierarchy, and show that it collapses in the second level.

1. Introduction

1.1. Context and objective

In the framework of network computing, *distributed decision* is the ability to check the legality of network configurations using a distributed algorithm. This concern is of the utmost importance in the context of fault-tolerant distributed computing, where it is highly desirable that the nodes are able to collectively check the legality of their current configuration, which could have been altered by the corruption of variables due to failures. In this paper, we are interested in *local* distributed decision. More specifically, we consider the standard LOCAL model of computation in networks [15]. Nodes are assumed to be given distinct identities, and each node executes the same algorithm, which proceeds in synchronous rounds where all nodes start at the same time. In each round, every node sends messages to its neighbors, receives messages from its neighbors, and performs some individual computation. The model does not limit the amount of data sent in the messages,

☆ A preliminary version of this paper has appeared in the proceedings of the 34th International Symposium on Theoretical Aspects of Computer Science (STACS), Hannover, Germany, March 8–11, 2017. This work was partially done during the first and fourth authors visit at IRIF (CNRS and University Paris Diderot).

* Corresponding author.
E-mail address: alkida.balliu@aalto.fi (A. Balliu).

[1] Supported in part by Gran Sasso Science Institute, L'Aquila, Italy, and by the Academy of Finland, Grant 285721. Additional support from the French ANR project DESCARTES, Grant DS0702-2016.
[2] Additional support from the French ANR project DESCARTES, Grant DS0702-2016, and the INRIA project GANG.

https://doi.org/10.1016/j.jcss.2018.05.004
0022-0000/

neither does it limit the amount of computation that is performed by a node during a round. Indeed, the model places an emphasis on the number of rounds before every node can output, as a measure of locality. A *local algorithm* is a distributed algorithm $\mathcal{A}$ satisfying that there exists a constant $t \geq 0$ such that $\mathcal{A}$ terminates in at most t rounds in all networks, for all inputs. The parameter t is called the *radius* of $\mathcal{A}$. In other words, in every network G, and for all inputs to the nodes of G, every node executing $\mathcal{A}$ just needs to collect all information present in the t-ball around it in order to output, where the *t-ball* of u is the ball $B_G(u,t) = \{v \in V(G) : \text{dist}(u,v) \leq t\}$, where $\text{dist}(u,v)$ denotes the length (i.e., number of edges) of a shortest path between u and v.

The objective of the paper is to determine what network properties can be decided locally, as a function of the individual computing power of the nodes.

Following the guidelines of [7], we define a *configuration* as a pair (G,x) where $G = (V,E)$ is a connected simple undirected graph, and $x : V(G) \to \{0,1\}^*$ is a function assigning an input $x(u)$ to every node $u \in V$. A *distributed language* $\mathcal{L}$ is a set of configurations (we consider only Turing-decidable sets). A configuration $(G,x) \in \mathcal{L}$ is said to be *legal* w.r.t. $\mathcal{L}$. The membership of a configuration in a distributed language is independent of the identity that may be assigned to the nodes in the LOCAL model. For instance, the set $\{(G,x) : \exists v \in V(G), \text{id}(v) = 1\}$ is not considered as a distributed language.

The class LD is the set of all distributed languages that are locally decidable. That is, LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying that, for every configuration (G,x),

$$(G,x) \in \mathcal{L} \iff \mathcal{A} \text{ accepts } (G,x)$$

where one says that $\mathcal{A}$ accepts if it accepts at *all* nodes. More formally, given a graph G, let $\text{ID}(G)$ be the set of all injective functions from $V(G)$ to positive integers, i.e., $\text{ID}(G)$ denote the set of all possible identity assignments to the nodes of G. Then LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G,x),

$$(G,x) \in \mathcal{L} \Rightarrow \forall \text{id} \in \text{ID}(G), \forall u \in V(G), \mathcal{A}_{G,x,\text{id}}(u) = \text{accept}$$
$$(G,x) \notin \mathcal{L} \Rightarrow \forall \text{id} \in \text{ID}(G), \exists u \in V(G), \mathcal{A}_{G,x,\text{id}}(u) = \text{reject}$$

where $\mathcal{A}_{G,x,\text{id}}(u)$ is the output of Algorithm $\mathcal{A}$ running on the instance (G,x) with identity-assignment id, at node u. (Note that the two implications in the definition of LD cannot be merged into one if-and-only-if statement because LD requires that both ways should hold *for any identity-assignment* to the nodes.) For instance, the language PROP-COL, composed of all (connected) properly colored graphs, is in LD. Similarly, the class LCL of "locally checkable labelings", defined in [14], satisfies $\text{LCL} \subseteq \text{LD}$. In fact, LCL is precisely LD restricted to configurations on graphs with constant maximum degree, and inputs of constant size.

The class NLD is the non-deterministic version of LD, i.e., the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ *verifying* $\mathcal{L}$, i.e., satisfying that, for every configuration (G,x),

$$(G,x) \in \mathcal{L} \iff \exists c, \mathcal{A} \text{ accepts } (G,x) \text{ with certificate } c.$$

More formally, NLD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G,x),

$$(G,x) \in \mathcal{L} \Rightarrow \exists c \in \mathcal{C}(G), \forall \text{id} \in \text{ID}(G), \forall u \in V(G), \mathcal{A}_{G,x,c,\text{id}}(u) = \text{accepts}$$
$$(G,x) \notin \mathcal{L} \Rightarrow \forall c \in \mathcal{C}(G), \forall \text{id} \in \text{ID}(G), \exists u \in V(G), \mathcal{A}_{G,x,c,\text{id}}(u) = \text{rejects}$$

where $\mathcal{C}(G)$ is the class of all functions $c : V(G) \to \{0,1\}^*$, assigning the certificate $c(u)$ to each node u. Note that the certificates c may depend on the network and on the input to the nodes, but should be set independently of the actual identity assignment to the nodes of the network. If we were able to set certificates depending on the ID-assignment to the nodes, then every distributed language would be non-deterministically decidable [8,9]. In this paper, we aim at a better understanding of the power given to the verification protocol by the ability to set up ID-dependent certificates. For this purpose, we follow the guidelines of [7] by considering ID-independent certificates, hence reducing the role if IDs to mere mechanisms enabling each node to solely distinguishing nodes in the network. (See [2] for a more detailed description of the differences between ID-dependent and ID-independent certificates.) In the following, for the sake of simplifying the notations, we shall omit specifying the domain sets $\mathcal{C}(G)$ and $\text{ID}(G)$ unless they are not clear from the context. It follows from the above that NLD is a class of distributed languages that can be locally *verified*, in the sense that, on legal instances, certificates can be assigned to nodes by a *prover* so that a *verifier* $\mathcal{A}$ accepts, and, on illegal instances, the verifier $\mathcal{A}$ rejects (i.e., at least one node rejects) systematically, and cannot be fooled by any fake certificate. For instance, the language

$$\text{TREE} = \{(G,x) : G \text{ is a tree}\}$$

is in NLD, by selecting a root r of the given tree, and assigning to each node u a counter $c(u)$ equal to its hop-distance to r (the hop-distance between two nodes u and v is the minimum number of edges of a path with extremities u and v). If the given (connected) graph contains a cycle, then no counters could be assigned to fool an algorithm checking that, at each node u with $c(u) \neq 0$, a unique neighbor v satisfies $c(v) < c(u)$, and all other neighbors $w \neq v$ satisfy $c(w) > c(u)$. In [6],