Node Labels in Local Decision

Pierre Fraigniaud, Juho Hirvonen, Jukka Suomela

Node Labels in Local Decision

Pierre Fraigniaud[1], Juho Hirvonen[2], and Jukka Suomela[2]

[1] Theoretical Computer Science Federation
CNRS and University Paris Diderot, France
pierre.fraigniaud@liafa.univ-paris-diderot.fr
[2] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, Aalto University, Finland
juho.hirvonen@aalto.fi, jukka.suomela@aalto.fi

Abstract. The role of unique node identifiers in network computing is well understood as far as *symmetry breaking* is concerned. However, the unique identifiers also *leak information* about the computing environment—in particular, they provide some nodes with information related to the size of the network. It was recently proved that in the context of *local decision*, there are some decision problems such that (1) they cannot be solved without unique identifiers, and (2) unique node identifiers leak a *sufficient* amount of information such that the problem becomes solvable (PODC 2013).

In this work we give study what is the *minimal* amount of information that we need to leak from the environment to the nodes in order to solve local decision problems. Our key results are related to *scalar oracles* f that, for any given n, provide a multiset $f(n)$ of n labels; then the adversary assigns the labels to the n nodes in the network. This is a direct generalisation of the usual assumption of unique node identifiers. We give a complete characterisation of the *weakest oracle* that leaks at least as much information as the unique identifiers.

Our main result is the following dichotomy: we classify scalar oracles as *large* and *small*, depending on their asymptotic behaviour, and show that (1) any large oracle is at least as powerful as the unique identifiers in the context of local decision problems, while (2) for any small oracle there are local decision problems that still benefit from unique identifiers.

Eligible for the best student paper award—Juho Hirvonen is a full-time student.

1 Introduction

This work studies the role of *unique node identifiers* in the context of *local decision problems* in distributed systems. We generalise the concept of node identifiers by introducing *scalar oracles* that choose the labels of the nodes, depending on the size of the network n—in essence, we let the oracle leak some information on n to the nodes—and ask what is the *weakest* scalar oracle that we could use instead of unique identifiers. We prove the following dichotomy: we classify each scalar oracle as *small* or *large*, depending on its asymptotic behaviour, and we show that the large oracles are precisely those oracles that are at least as strong as unique identifiers.

1.1 Context and background

The research trends within the framework of distributed computing are most often pragmatic. Problems closely related to real world applications are tackled under computational assumptions reflecting existing systems, or systems whose future existence is plausible. Unfortunately, small variations in the model settings may lead to huge gaps in terms of computational power. Typically, some problems are unsolvable in one model but may well be efficiently solvable in a slight variant of that model. In the context of *network computing*, this commonly happens depending on whether the model assumes that pairwise distinct identifiers are assigned to the nodes. While the presence of distinct identifiers is inherent to some systems (typically, those composed of artificial devices), the presence of such identifiers is questionable in others (typically, those composed of biological or chemical elements). Even if the identifiers are present, they may not necessarily be directly visible, e.g., for privacy reasons.

The absence of identifiers, or the difficulty of accessing the identifiers, limits the power of computation. Indeed, it is known that the presence of identifiers ensures two crucial properties, which are both used in the design of efficient algorithms. One such property is **symmetry breaking**. The absence of identifiers makes symmetry breaking far more difficult to achieve, or even impossible if asymmetry cannot be extracted from the inputs of the nodes, from the structure of the network, or from some source of random bits. The role of the identifiers in the framework of network computing, as far as symmetry breaking is concerned, has been investigated in depth, and is now well understood [1–8, 13, 15–23, 26–28].

The other crucial property of the identifiers is their ability to **leak global information** about the framework in which the computation takes place. In particular, the presence of pairwise distinct identifiers guarantees that at least one node has an identifier at least n in n-node networks. This apparently very weak property was proven to actually play an important role when one is interested in checking the correctness of a system configuration in a decentralised manner. Indeed, it was shown in prior work [10] that the ability to check the legality of a system configuration with respect to some given Boolean predicate differs significantly according to the ability of the nodes to use their identifiers. This phenomenon is of a nature different from symmetry breaking, and is far less understood than the latter.

More precisely, let us define a *distributed language* as a set of system configurations (e.g., the set of properly coloured networks, or the set of networks each with a unique leader). Then let LD be the class of distributed languages that are *locally decidable*. That is, LD is the set of distributed languages for which there exists a distributed algorithm where every node inspects its neighbourhood at constant distance in the network, and outputs *yes* or *no* according to the following rule: all nodes output *yes* if and only if the instance is legal. Equivalently, the instance is illegal if and only if at least one node outputs *no*. Let LDO be defined as LD with the restriction the local algorithm is required to be *identifier oblivious*, that is, the output of every node is the same regardless of the identifiers assigned to the nodes. By definition, $\mathsf{LDO} \subseteq \mathsf{LD}$, but [10] proved that this inclusion is

strict: there are languages in $\mathsf{LD} \setminus \mathsf{LDO}$. This strict inclusion was obtained by constructing a distributed language that can be decided by an algorithm whose outputs depend heavily on the identifiers assigned to the nodes, and in particular on the fact that at least one node has an identifier whose value is at least n.

The gap between LD and LDO has little to do with symmetry breaking. Indeed, decision tasks do not require that some nodes act differently from the others: on legal instances, all nodes must output *yes*, while on illegal instances, it is permitted (but not required) that all nodes output *no*. The gap between LD and LDO is entirely due to the fact that the identifiers leak information about the size n of the network. Moreover, it is known that the gap between LD and LDO is strongly related to computability issues: there is an identifier-oblivious *non-computable* simulation A' of every local algorithm A that uses identifiers to decide a distributed language [10]. Informally, for every language in $\mathsf{LD} \setminus \mathsf{LDO}$, the unique identifiers are precisely as helpful as providing the nodes with the capability of solving undecidable problems.

1.2 Objective

One objective of this paper is to measure the *amount of information* provided to a distributed system via the labels given to its nodes. For this purpose, we consider the classes LD and LDO enhanced with *oracles*, where an oracle f is a function that provides every node with information about its environment.

We focus on the class of *scalar* oracles, which are functions over the positive integers. Given an $n \geq 1$, a scalar oracle f returns a list $f(n) = (f_1, \ldots, f_n)$ of n labels (bit strings) that are assigned arbitrarily to the nodes of any n-node network in a one-to-one manner. The class LD^f (resp., LDO^f) is then defined as the class of distributed languages decidable locally by an algorithm (resp., by an identifier-oblivious algorithm) in networks labelled with oracle f.

If, for every $n \geq 1$, the n values in the list $f(n)$ are pairwise distinct, then $\mathsf{LD} \subseteq \mathsf{LDO}^f$ since the nodes can use the values provided to them by the oracle as identifiers. However, as we shall demonstrate in the paper, this pairwise distinctness condition is not necessary.

Our goal is to identify the interplay between the classes LD, LDO, LD^f, and LDO^f, with respect to any scalar oracle f, and to characterise the power of identifiers in distributed systems as far as leaking information about the environment is concerned.

1.3 Our results

Our first result is a characterisation of the weakest oracles providing the same power as unique node identifiers. We say that a scalar oracle f is *large* if, roughly, f ensures that, for any set of k nodes, the largest value provided by f to the nodes in this set grows with k (see Section 2.3 for the precise definition). We show the following theorem.

Theorem 1. *For any computable scalar oracle f, we have $\mathsf{LDO}^f = \mathsf{LD}^f$ if and only if f is large.*

Theorem 1 is a consequence of the following two lemmas. The first says that small oracles (i.e. non-large oracles) do not capture the power of unique identifiers. Note that the following separation result holds for any small oracle, including uncomputable oracles.

Lemma 1. *For any small oracle f, there exists a language $L \in \mathsf{LD} \setminus \mathsf{LDO}^f$.*

The second is a simulation result, showing that any local decision algorithm using identifiers can be simulated by an identifier-oblivious algorithm with the help of *any* large oracle, as long as the oracle itself is computable. Essentially large oracles capture the power of unique identifiers.

Lemma 2. *For any large computable oracle f, we have $\mathsf{LD} \subseteq \mathsf{LDO}^f = \mathsf{LD}^f$.*

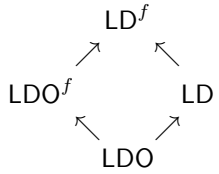
Theorem 1 holds despite the fact that small oracles can still produce some large values, and that there exist small oracles guaranteeing that, in any n-node network, at least one node has a value at least n. Such a small oracle would be sufficient to decide the language $L \in \mathsf{LD} \setminus \mathsf{LDO}$ presented in [10]. However, it is not sufficient to decide all languages in LD.

Our second result is a complete description of the hierarchy of the four classes LD, LDO, LD^f, and LDO^f of local decision, using identifiers or not, with or without oracles. The pictures for small and large oracles are radically different.

- For any large oracle f, the hierarchy yields a *total order*:

  $$\mathsf{LDO} \subsetneq \mathsf{LD} \subseteq \mathsf{LDO}^f = \mathsf{LD}^f.$$

  The strict inclusion $\mathsf{LDO} \subsetneq \mathsf{LD}$ follows from [10]. The second inclusion $\mathsf{LD} \subseteq \mathsf{LDO}^f$ may or may not be strict depending on oracle f.
- For any small oracle f, the hierarchy yields a *partial order*. We have $\mathsf{LDO}^f \subsetneq \mathsf{LD}^f$ as a consequence of Lemma 1. However, LD and LDO^f are incomparable, in the sense that there is a language $L \in \mathsf{LD} \setminus \mathsf{LDO}^f$ for any small oracle f, and there is a language $L \in \mathsf{LDO}^f \setminus \mathsf{LD}$ for some small oracles f. Hence, the relationships of the four classes can be represented as the following diagram:

  All inclusions (represented by arrows) can be strict.

1.4 Additional related work

In the context of network computing, oracles and advice commonly appear in the form of *labelling schemes* [9, 14]. A typical example is a *distance labelling scheme*, which is a labelling of the nodes so that the distance between any pair

of nodes can be computed or approximated based on the labels. Other examples are *routing schemes* that label the nodes with information that helps in finding a short path between any given source and destination. For graph problems, one could of course encode the entire solution in the advice string—hence the key question is whether a very small amount of advice helps with solving a given problem.

In prior work, it is commonly assumed that the oracle can give a specific piece of advice for each individual node. The advice is localised, and entirely controlled by the oracle. Moreover, the oracle can see the entire problem instance and it can tailor the advice for any given task.

In the present work, we study a much weaker setting: the oracle is only given n, and it cannot choose which label goes to which node. This is a generalisation of, among others, typical models of *networks with unique identifiers*: one commonly assumes that the unique identifiers are a permutation of $\{1, 2, \ldots, n\}$ [20], which in our case is exactly captured by the large scalar oracle

$$f(n) = (1, 2, \ldots, n),$$

or that the unique identifiers are a subset of $\{1, 2, \ldots, n^c\}$ for some constant c [25], which in our case is captured by a subfamily of large scalar oracles. Our model is also a generalisation of *anonymous networks with a unique leader* [13]—the assumption that there is a unique leader is captured by the small scalar oracle

$$f(n) = (0, 0, \ldots, 0, 1).$$

2 Model and definitions

In this work, we augment the usual definitions of *locally checkable labellings* [22] and *local distributed decision* [10–12] with scalar oracles.

2.1 Computational model

We deal with the standard LOCAL model [25] for distributed graph algorithms. In this model, the network is a simple connected graph $G = (V, E)$. Each node $v \in V$ has an *identifier* $\mathsf{id}(v) \in \mathbb{N}$, and all identifiers of the nodes in the network are pairwise distinct. Computation proceeds in synchronous rounds. During a round, each node communicates with its neighbours in the graph, and performs some local computation. There are no limits to the amount of communication done in a single round. Hence, in r communication rounds, each node can learn the complete topology of its radius-r neighbourhood, including the inputs and the identifiers of the nodes in this neighbourhood. In a distributed algorithm, all nodes start at the same time, and each node must halt after some number of rounds, and produce its individual output. The collection of individual outputs then forms the global output of the computation. The running time of the algorithm is the number of communication rounds until all nodes have halted.

We consider *local* algorithms, i.e., constant-time algorithms [26]. That is, we focus on algorithms with a running time that does not depend on the size n of the graph. Any such algorithm, with running time r, can be seen as a function from the set of all possible radius-r neighbourhoods to the set of all possible outputs. An *identifier-oblivious* algorithm is an algorithm whose outputs are independent of the identifiers assigned to the nodes. Note that, from the perspective of an identifier-oblivious algorithm, the set of all possible radius-r degree-d neighbourhoods is finite. This is not the case for every algorithm since there are infinitely many identifier assignments to the nodes in a radius-r degree-d neighbourhood.

Although the LOCAL model does not put any restriction on the amount of individual computation performed at each node, we only consider algorithms that are *computable*.

2.2 Local decision tasks

We are interested in the power of constant-time algorithms for *local decision*. A *labelled graph* is a pair (G, x), where G is a simple connected graph, and $x : V(G) \to \{0,1\}^*$ is a function assigning a label to each node of G. A *distributed language* L is a set of labelled graphs. Examples of distributed languages include:

- 2-colouring, the language where G is a bipartite graph and $x(v) \in \{0,1\}$ for all $v \in V(G)$ such that $x(v) \neq x(u)$ whenever $\{u, v\} \in E(G)$;
- parity, the language of graphs with an even number of nodes;
- planarity, the language that consists of all planar graphs.

We say that algorithm A decides L if and only if the output of A at every node is either *yes* or *no*, and, for every instance (G, x), A satisfies:

$$(G, x) \in L \iff \text{all nodes output } \textit{yes}.$$

Hence, for an instance $(G, x) \notin L$, the algorithm A must ensure that at least one node outputs *no*. We consider two main distributed complexity classes:

- LD (for *local decision*) is the set of languages decidable by constant-time algorithms in the LOCAL model.
- LDO (for *local decision oblivious*) is the set of languages decidable by constant-time identifier-oblivious algorithms in the LOCAL model.

By definition, $\mathsf{LDO} \subseteq \mathsf{LD}$, and it is known [10] that this inclusion is strict: there are languages $L \in \mathsf{LD} \setminus \mathsf{LDO}$. The fact that we consider only computable algorithms is crucial here—without this restriction we would have $\mathsf{LDO} = \mathsf{LD}$ [10].

2.3 Distributed oracles

We study the relationship of classes LD and LDO with respect to *scalar oracles*. Such an oracle f is a function that assigns a list of n values to every positive integer n, i.e.,

$$f(n) = (f_1, f_2, \ldots, f_n)$$

with $f_i \in \{0,1\}^*$. In essence, oracle f can provide some information related to n to the nodes. In an n-node graph, each of the n nodes will receive a value $f_i \in f(n)$, $i \in [n]$. These values are arbitrarily assigned to the nodes in a one-to-one manner. Two different nodes will thus receive f_i and f_j with $i \neq j$. Note that f_i may or may not be different from f_j for $i \neq j$; this is up to the choice of the oracle. The way the values provided by the oracles are assigned to the nodes is under the control of an adversary. One example of an oracle is $f(n) = (1, 2, \ldots, n)$, which provides the nodes with identifiers. Another example is $f(n) = (0, 0, \ldots, 0)$, which provides no information to the nodes.

W.l.o.g., let us assume that $f_i \leq f_{i+1}$ for every i. We use the shorthand $f_k^{(n)}$ for the kth label provided by f on input n, that is, $f(n) = (f_1^{(n)}, f_2^{(n)}, \ldots, f_n^{(n)})$. For a fixed oracle f, we consider two main distributed complexity classes:

- LD^f is the set of languages decidable by constant-time algorithms in networks that are labelled with oracle f.
- LDO^f is the set of languages decidable by constant-time indentifier-oblivious algorithms in networks that are labelled with oracle f.

We will separate oracles in two classes, which play a crucial role in the way the four classes LDO, LD, LDO^f, and LD^f interact.

Definition 1. An oracle f is said to be *large* if

$$\forall c > 0,\ \exists k \geq 1,\ \forall n \geq k,\ f_k^{(n)} \geq c.$$

An oracle is *small* if it is not large.

Hence, a large oracle f satisfies that, for any value $c > 0$, there exists a large enough k, such that, in every graph G of size at least k, for every set of nodes $S \subseteq V(G)$ of size $|S| \geq k$, oracle f is providing at least one node of S with a value at least as large as c. In short: every large set of nodes must include at least one node that receives a large value.

Conversely, a small oracle f satisfies that there exists a value $c > 0$ such that, for every k, we can find $n \geq k$ such that, in every n-node graph G, and for every set of nodes $S \subseteq V(G)$ of size $|S| \geq k$, there is an assignment of the values provided by f such that every node in S receives a value smaller that c. In short: there are arbitrarily large sets of nodes which all receive a small value.

For example, oracles $f(n) = (1, 2, \ldots, n)$ and $f(n) = (n, n, \ldots, n)$ are large, while oracles $f(n) = (0, 0, \ldots, 0, 1)$ and $f(n) = (0, 0, \ldots, 0, 2^n)$ are small. We emphasise that small oracles can output very large values.

3 Proof of the main theorem

In this section we give the proof of our main result that characterises the power of weak and large oracles with respect to identifier-oblivious local decision.

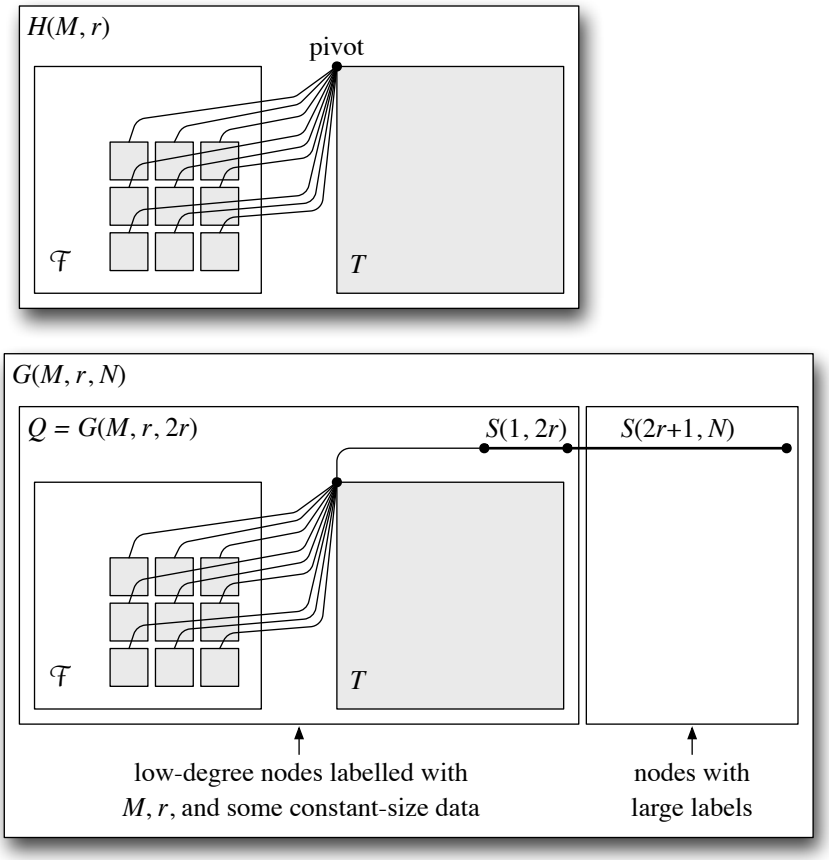


Fig. 1. The construction of Section 3.1.

3.1 Small oracles do not capture the power of unique identifiers

Fraigniaud et al. [10] showed that there exists a language $L \in \mathsf{LD} \setminus \mathsf{LDO}$. We use a very similar Turing machine construction as in the proof of their Theorem 1. However, we must take into account the additional concern of the values that the oracle assigns to the nodes. We handle this by forcing any small oracle to always give many copies of the same constant label c so that the adversary can cover the interesting parts of the construction with this unhelpful label c. We can then use uncomputability arguments to show that if a certain language were in LDO^f, then we could get a sequential algorithm for uncomputable problems. See Figure 1 for illustrations.

Lemma 1. *For any small oracle f, there exists a language $L \in \mathsf{LD} \setminus \mathsf{LDO}^f$.*

Proof. We assume that for each halting Turing machine M and each locality parameter $r \in \mathbb{N}$, there exists a labelled graph $H(M, r)$ with the following properties:

(P1) There is an identifier-oblivious local checker that verifies that a given labelled graph is a equal to $H(M, r)$ for some M and r.
(P2) The number of nodes in the graph $H(M, r)$ is at least as large as the number of steps M takes on an empty tape.
(P3) Given $H(M, r)$, an identifier-oblivious local checker A with a running time of r cannot decide if M outputs 0 or 1.
(P4) Each label of $H(M, r)$ is a triple $x(v) = (M, r, x'(v))$. The maximum degree of H and the maximum size of $x'(v)$ are constants that only depend on r.
(P5) Graph $H(M, r)$ can be padded with additional nodes without violating properties (P1)–(P4).

The construction of Fraigniaud et al. [10] satisfies these properties. They show how to construct a labelled graph $H(M, r)$ that encodes the execution table of a given Turing machine M such that a local checker with running time r cannot decide if M halts with 0 or 1. The original construction $(H, x) = H(M, r)$ consists of three main parts.

(i) *The execution table T of the Turing machine M.* Let s be the number of steps M takes on an empty tape. Then table T is an $(s + 1) \times (s + 1)$ grid, where node (i, j) holds the contents of the tape at position j after computation step i, and its own coordinates (i, j) modulo 3. Node (i, j) also knows if the head is at position j after step i, and if so, what is the state of M after step i. Node $(0, 0)$ representing the first position of the empty tape is called the *pivot*. The execution table exists essentially to guarantee (P2).
(ii) *The fragment collection $\mathcal{F}$.* This is a collection of subgrids labelled with all syntactically possible ways that are consistent with being in some execution table of M. The dimensions of the fragments are linear in r and independent of M. In each fragment, every 2×2 subgrid is consistent with a state transition of M. It is crucial to observe that there is a finite number of such fragments. Each fragment is connected to the pivot in a way that supports the local verification of the structure. The fragment collection is added to ensure (P3). Informally, if we only had T, then some node (i, s) at the last row of the grid would be able to see the stopping state of M; however, $\mathcal{F}$ will contain some fragments in which M halts with output 0 and some fragments in which M halts with output 1, and the nodes at the last row of T are locally indistinguishable from the nodes in such fragments.
(iii) *Pyramid structure.* This is added to the execution table and to the fragments to ensure (P1). Without any additional structure, a grid with coordinates modulo 3 is locally indistinguishable from, e.g., a grid that is wrapped into a torus. The pyramid structure guarantees that at least one node is able to detect invalid instances.

Finally, since all labellings can be made constant-size, we can ensure (P4). In particular, for any (M, r), there are constantly many syntactically possible r-neighbourhoods of $H(M, r)$. This is a crucial property as it guarantees that there is a sequential algorithm that on all inputs (M, r) halts and, if M halts, outputs all possible labelled r-neighbourhoods of $H(M, r)$.

Let $S(a, b)$ be the labelled path $(s_a, s_{a+1}, \ldots, s_b)$ in which node s_i is labelled with value i. We augment the construction $H(M, r)$ as follows: labelled graph $G(M, r, N)$ consists of $H(M, r)$, plus $S(1, N)$, plus an edge between the pivot of $H(M, r)$ and the first node s_1 of the path $S(1, N)$; we call $S(1, N)$ the *tail* of the construction. The structure of $G(M, r, N)$ is still locally checkable in LDO: any tail must eventually connect to the pivot, and the pivot can detect if there are multiple tails. The key property of the construction is that the nodes in the tail $S(1, N)$ with large labels are far from the nodes of $G(M, r)$ that are aware of M.

We will separate LD and LDO^f using the following language:

$$L = \{G(M, r, N) : r \geq 1,\ N \geq 1,\ \text{and Turing machine } M \text{ outputs } 0\}.$$

We have $L \in \mathsf{LD}$ as there will be a node v with $\mathsf{id}(v) \geq s$ which can simulate M for s steps and output *no* if M does not output 0. Next we will argue that L cannot be in LDO^f for any small$f$.

Let f be a small oracle. For any M and r, we can choose a sufficiently large N as follows. By definition, there exists a c such that for all k oracle f outputs some label $i \in [c]$ at least $\lceil k/c \rceil$ times on some $n \geq k$. Moreover, we can find an infinite sequence of values $k_0, k_1, \ldots$ such that the most common value is some fixed i_0. We select w.l.o.g. the smallest k_j and a suitable n such that $f(n)$ contains at least $k_j/c \geq |H(M, r)| + 2r$ labels equal to i_0. Let $N = n - |H(M, r)|$, and consider $G(M, r, N)$. Now the adversary can construct the following *worst-case labelling*: every node of $G(M, r, 2r) \subseteq G(M, r, N)$ receives the constant input $i_0 \in [c]$; all other labels as assigned to the nodes in $S(2r + 1, N) \subseteq G(M, r, N)$.

It is known that separating the following languages is undecidable (see e.g. [24, p. 65]):

$$L_i = \{M : \text{Turing machine } M \text{ outputs } i\} : i \in \{0, 1\}. \tag{1}$$

For the sake of contradiction, we assume that there is an LDO^f-algorithm A that decides L. We will use algorithm A and constant i_0 defined above to construct a sequential algorithm B that separates L_0 and L_1.

Let r be the running time of A, and consider the execution of A on an instance $G(M, r, N)$ for some M and N. It follows that each node in $S(r + 1, N) \subseteq G(M, r, N)$ must always output *yes*. To see this, note that the claim is trivial if M halts with 0. Otherwise we can always construct another instance $G(M_0, r, N)$ such that M_0 halts with 0 and both $G(M, r, N)$ and $G(M_0, r, N)$ have the same number of nodes. Hence the oracle and the adversary can assign the same labels to $S(r + 1, N)$ in both $G(M, r, N)$ and $G(M_0, r, N)$. If any of these nodes would answer *no* in $G(M, r, N)$, then A would also incorrectly reject the *yes*-instance $G(M_0, r, N) \in L$.

Now given a Turing machine M, algorithm B proceeds as follows. Consider the subgraph $Q = G(M, r, 2r) \subseteq G(M, r, N)$, and assume the worst-case labelling

of $G(M, r, N)$ in which all nodes of Q have the constant label i_0. Algorithm B cannot construct Q; indeed, M might not halt, in which case $G(M, r, N)$ would not even exist. However, B can do the following: it can assume that M halts, and then generate a collection $\mathcal{Q}$ that would contain all possible radius-r neighbourhoods of the nodes in $G(M, r, r)$. Collection $\mathcal{Q}$ is finite, its size only depends on r and M, and the key observation is that $\mathcal{Q}$ is computable (in essence, B enumerates all syntactically possible fixed-size fragments of partial execution tables of M).

Then B will simulate A in each neighbourhood of $\mathcal{Q}$. If M halts with 1, then $G(M, r, N) \notin L$, and therefore one of the nodes in $G(M, r, r)$ has to output *no*; in this case B outputs 1. If M halts with 0, then $G(M, r, N) \in L$, and therefore one of the nodes in $G(M, r, r)$ has to output *yes*; in this case B outputs 0. The key observation is that B will always halt with some (meaningless) output even if we are given an input $M \notin L_0 \cup L_1$; hence B is a computable function that separates L_0 and L_1. As such a B cannot exist, A cannot exist either. □

3.2 Large oracles capture the power of unique identifiers

In this section we will show that a *computable* large oracle f is sufficient to have $\mathsf{LD} \subseteq \mathsf{LDO}^f = \mathsf{LD}^f$. This result holds even if f only has access to an upper bound $N \geq n$, and the adversary gets to pick an n-subset of labels from $f(N)$. Note that the oracle has to be computable in order for us to invert it locally.

Lemma 2. *For any large computable oracle f, we have $\mathsf{LD} \subseteq \mathsf{LDO}^f = \mathsf{LD}^f$.*

Proof. We begin by showing how to recover an oracle $\hat{f}$ with $\hat{f}_k^{(N)} \geq k$, for all k and $N \geq k$, from a large oracle f. We want to guarantee that each node v receives a label $\ell \geq i$ if in the initial labelling it had the ith smallest label.

By definition, it holds for large oracles that for each natural number ℓ there is a largest index i such that $f_i^{(N)} \leq \ell$; we denote the index by $g(\ell)$. By assumption, a node with label ℓ can locally compute the value $g(\ell)$. We now claim that

$$\hat{f} \colon N \mapsto \big\{g(f_1), g(f_2), \ldots, g(f_N)\big\}$$

has the property $\hat{f}_k^{(N)} \geq k$. To see this, assume that we have $f_k^{(N)} = \ell$ for an arbitrary k. Seeing label ℓ, node v knows that, in the worst case, its own label is the $g(\ell)$th smallest. Thus for every k, the node with the kth smallest label will compute a new label at least k.

Now given $\hat{f}$, we can simulate any r-round LD-algorithm A as follows.

1. Each node v with label ℓ_v locally computes the new label $g(\ell_v)$.
2. Each node gathers all labels $g(\ell_u)$ in its r-neighbourhood. Denote by g_v^* the maximum value in the neighbourhood of v.
3. Each node v simulates A on every unique identifier assignment to its local r-neighbourhood from $\{1, 2, \ldots, g_v^*\}$. If for some assignment A outputs *no*, then v outputs *no*, and otherwise it outputs *yes*.

Because of how the decision problem is defined, it is always safe to output *no* when some simulation of A outputs *no*. It remains to be argued that it is safe to say *yes*, if all simulations say *yes*. This requires that *some* subset of simulations of A, one for each node, looks as if there had been a consistent setting of unique identifiers on the graph. Now let id be one identifier assignment with $\mathsf{id}(v) = i$ for the v with ith smallest label, for all i (breaking ties arbitrarily). Since by construction $g(\ell_v) \geq \mathsf{id}(v)$ for all v, there will be a simulation of A for every node v with local identifier assignment id_v such that for all u in the radius-r neighbourhood of v we have $\mathsf{id}_v(u) = \mathsf{id}(u)$.

So far we have seen how to simulate any LD-algorithm A with LDO^f-algorithms. We can apply the same reasoning to simulate any LD^f-algorithm A with LDO^f-algorithms; the only difference is that each node in the simulation has now access to the original oracle labels as well. □

4 Full characterisation of LD^f, LDO^f, LD, and LDO

Our goal in this section is to complete the characterisation of the power of scalar oracles with respect to the classes LD and LDO. We aim at giving a robust characterisation that holds also for minor variations in the definition of a scalar oracle. In particular, all of the key results can be adapted to weaker oracles that only receive an upper bound $N \geq n$ on the size of the graph.

4.1 Large oracles can be stronger than identifiers

Let us first consider large oracles. By prior work [10] and Lemma 2 we already know that for any computable large oracle f we have a linear order

$$\mathsf{LDO} \subsetneq \mathsf{LD} \subseteq \mathsf{LDO}^f = \mathsf{LD}^f.$$

Trivially, there is a large computable oracle $f(n) = (1, 2, \ldots, n)$ such that

$$\mathsf{LDO} \subsetneq \mathsf{LD} = \mathsf{LDO}^f = \mathsf{LD}^f.$$

We will now show that there is also a large computable oracle f such that

$$\mathsf{LDO} \subsetneq \mathsf{LD} \subsetneq \mathsf{LDO}^f = \mathsf{LD}^f.$$

For a simple proof, we could consider the large oracle $f(n) = (n, n, \ldots, n)$. Now the parity language L that consists of graphs with an even number of nodes is clearly in LDO^f but not in LD. However, this separation is not robust with respect to minor changes in the model of scalar oracles. In particular, if the oracle only knows an upper bound on n, we cannot use the parity language to separate LDO^f from LD.

In what follows, we will show that the *upper bound oracle* f that labels all nodes with some upper bound on $N \geq n$ can be used to separate LDO^f from LD.

Theorem 2. *For the upper bound oracle f there exists a language L such that $L \in \mathsf{LDO}^f \setminus \mathsf{LD}$.*

Proof. We again resort to computability arguments—see Appendix A.1 for the details.

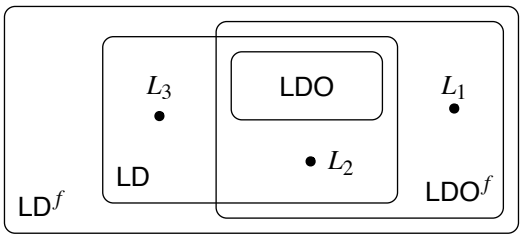


Fig. 2. There is a small oracle f such that each of the languages L_i exists.

4.2 Small oracles and identifiers are incomparable

In the case of small oracles, we already know that $\mathsf{LDO}^f \subsetneq \mathsf{LD}^f$ for any small oracle f by Lemma 1. Next we characterise the relationship of LDO^f and LD. In essence, we show that these classes are incomparable.

Theorem 3. *There is a single small oracle f so that each of the languages L_1, L_2, and L_3 shown in Figure 2 exist.*

Proof. Let f be the small oracle

$$f(n) = (0, 0, \ldots, 0, b_n),$$

where b_n is an n-bit string such that the ith bit tells whether the ith Turing machine halts. We construct the languages as follows:

L_1: Let $P(n)$ denote the labelled path of length n such that each node has two input labels: n and the distance to a specified leaf node v_0. The correct structure of $P(n)$ is in LDO. Now let

$$L_1 = \{P(M) : \text{Turing machine } M \text{ halts}\}.$$

The node that receives the n-bit oracle label can use it to decide whether the nth Turing machine halts, and therefore $L_1 \in \mathsf{LDO}^f$. Conversely, we have $L_1 \notin \mathsf{LD}$; otherwise we would have a sequential algorithm that solves the halting problem for each Turing machine M by constructing the path $P(M)$ with some fixed identifier assignment and simulating the local verifier.

L_2: We can use the same language

$$L_2 = \{H(M, r) : r \geq 1 \text{ and Turing machine } M \text{ outputs } 0\}$$

that we used in the proof of Lemma 1. It is known that $L_2 \in \mathsf{LD}$ and $L_2 \notin \mathsf{LDO}$ [10]. Since checking the structure of $H(M, r)$ is in LDO, it suffices to note that the node that receives the bit vector b_n of length n can use the *length* of the vector as an upper bound in simulating M. Thus $L_2 \in \mathsf{LDO}^f$.

L_3: Apply Lemma 1. □

We conclude by noting that Theorem 3 is also robust to minor variations in the definitions. In particular, the oracle does not need to know the exact value of n; it is sufficient that at least one node receives the bit string b_N, where $N \geq n$ is some upper bound on n.

Acknowledgements. Thanks to Laurent Feuilloley for discussions.

References

1. Angluin, D.: Local and global properties in networks of processors. In: Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980). pp. 82–93. ACM Press (1980), doi:10.1145/800141.804655.
2. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: Proc. 15th International Symposium on Distributed Computing (DISC 2001). Lecture Notes in Computer Science, vol. 2180, pp. 33–47. Springer (2001), doi:10.1007/3-540-45414-4_3.
3. Chalopin, J., Das, S., Santoro, N.: Groupings and pairings in anonymous networks. In: Proc. 20th International Symposium on Distributed Computing (DISC 2006). Lecture Notes in Computer Science, vol. 4167, pp. 105–119. Springer (2006), doi:10.1007/11864219_8.
4. Czygrinow, A., Hańćkowiak, M., Wawrzyniak, W.: Fast distributed approximations in planar graphs. In: Proc. 22nd International Symposium on Distributed Computing (DISC 2008). Lecture Notes in Computer Science, vol. 5218, pp. 78–92. Springer (2008), doi:10.1007/978-3-540-87779-0_6.
5. Diks, K., Kranakis, E., Malinowski, A., Pelc, A.: Anonymous wireless rings. Theoretical Computer Science 145(1–2), 95–109 (1995), doi:10.1016/0304-3975(94)00178-L.
6. Emek, Y., Pfister, C., Seidel, J., Wattenhofer, R.: Anonymous networks: randomization = 2-hop coloring. In: Proc. 33rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2014). pp. 96–105. ACM Press (2014), doi:10.1145/2611462.2611478.
7. Emek, Y., Seidel, J., Wattenhofer, R.: Computability in anonymous networks: revocable vs. irrecovable outputs. In: Proc. 41st International Colloquium on Automata, Languages and Programming (ICALP 2014). LNCS, vol. 8573, pp. 183–195. Springer (2014), doi:10.1007/978-3-662-43951-7_16.
8. Fich, F., Ruppert, E.: Hundreds of impossibility results for distributed computing. Distributed Computing 16(2–3), 121–163 (2003), doi:10.1007/s00446-003-0091-y.
9. Fraigniaud, P., Gavoille, C., Ilcinkas, D., Pelc, A.: Distributed computing with advice: information sensitivity of graph coloring. In: Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007). Lecture Notes in Computer Science, vol. 4596, pp. 231–242. Springer (2007), doi:10.1007/978-3-540-73420-8_22.
10. Fraigniaud, P., Göös, M., Korman, A., Suomela, J.: What can be decided locally without identifiers? In: Proc. 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC 2013). pp. 157–165. ACM Press (2013), doi:10.1145/2484239.2484264. arXiv:1302.2570.
11. Fraigniaud, P., Halldórsson, M.M., Korman, A.: On the impact of identifiers on local decision. In: Proc. 16th International Conference on Principles of Distributed Systems (OPODIS 2012). Lecture Notes in Computer Science, vol. 7702, pp. 224–238. Springer (2012), doi:10.1007/978-3-642-35476-2_16.
12. Fraigniaud, P., Korman, A., Peleg, D.: Local distributed decision. In: Proc. 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011). IEEE Computer Society Press (2011), doi:10.1109/FOCS.2011.17.
13. Fraigniaud, P., Pelc, A., Peleg, D., Pérennes, S.: Assigning labels in an unknown anonymous network with a leader. Distributed Computing 14(3), 163–183 (2001), doi:10.1007/PL00008935.

14. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distributed Computing 16(2–3), 111–120 (2003), doi:10.1007/s00446-002-0073-5.
15. Göös, M., Hirvonen, J., Suomela, J.: Lower bounds for local approximation. Journal of the ACM 60(5), 39:1–23 (2013), doi:10.1145/2528405. arXiv:1201.6675.
16. Hasemann, H., Hirvonen, J., Rybicki, J., Suomela, J.: Deterministic local algorithms, unique identifiers, and fractional graph colouring. Theoretical Computer Science (2014), to appear. doi:10.1016/j.tcs.2014.06.044.
17. Hella, L., Järvisalo, M., Kuusisto, A., Laurinharju, J., Lempiäinen, T., Luosto, K., Suomela, J., Virtema, J.: Weak models of distributed computing, with connections to modal logic. Distributed Computing 28(1), 31–53 (2015), doi:10.1007/s00446-013-0202-3. arXiv:1205.2051.
18. Kranakis, E.: Symmetry and computability in anonymous networks: a brief survey. In: Proc. 3rd Colloquium on Structural Information and Communication Complexity (SIROCCO 1996). pp. 1–16. Carleton University Press (1997)
19. Lenzen, C., Wattenhofer, R.: Leveraging Linial's locality limit. In: Proc. 22nd International Symposium on Distributed Computing (DISC 2008). Lecture Notes in Computer Science, vol. 5218, pp. 394–407. Springer (2008), doi:10.1007/978-3-540-87779-0_27.
20. Linial, N.: Locality in distributed graph algorithms. SIAM Journal on Computing 21(1), 193–201 (1992), doi:10.1137/0221015.
21. Mayer, A., Naor, M., Stockmeyer, L.: Local computations on static and dynamic graphs. In: Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS 1995). pp. 268–278. IEEE (1995), doi:10.1109/ISTCS.1995.377023.
22. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM Journal on Computing 24(6), 1259–1277 (1995), doi:10.1137/S0097539793254571.
23. Norris, N.: Classifying anonymous networks: when can two networks compute the same set of vector-valued functions? In: Proc. 1st Colloquium on Structural Information and Communication Complexity (SIROCCO 1994). pp. 83–98. Carleton University Press (1995)
24. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley Publishing Company (1994)
25. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, Philadelphia (2000)
26. Suomela, J.: Survey of local algorithms. ACM Computing Surveys 45(2), 24:1–40 (2013), doi:10.1145/2431211.2431223. http://www.cs.helsinki.fi/local-survey/.
27. Yamashita, M., Kameda, T.: Computing on anonymous networks: part I—characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems 7(1), 69–89 (1996), doi:10.1109/71.481599.
28. Yamashita, M., Kameda, T.: Leader election problem on networks in which processor identity numbers are not distinct. IEEE Transactions on Parallel and Distributed Systems 10(9), 878–887 (1999), doi:10.1109/71.798313.

A Appendix

A.1 Proof of Theorem 2

Theorem 2. *For the upper bound oracle f there exists a language L such that $L \in \mathsf{LDO}^f \setminus \mathsf{LD}$.*

Proof. The following language is not in LD but is in LDO^f. Recall the Turing machine construction $H(M, r)$ from Lemma 1. We augment it so that the pivot receives an extra label $\ell \in \{0, 1\}$; let us denote such a construction by $J(M, r, \ell)$. Let

$$L = \{J(M, r, \ell) : r \geq 1,\ \ell \in \{0, 1\},\ \text{and Turing machine } M \text{ halts outputs } \ell\}.$$

First, observe that L is in LDO^f. Checking the structure of $H(M, r)$ and hence $J(M, r, \ell)$ is known to be in LDO. Since the execution table of M is contained in $J(M, r, \ell)$, it must halt within $n \leq N$ steps. Finally, since the pivot is guaranteed to receive an $N \geq n$ as its oracle label, it can simulate M for at most N steps and determine whether M halts with output ℓ.

Next, we show that $L \notin \mathsf{LD}$. Suppose otherwise. Fix a local verifier A that decides L, and let r be the running time of A. Consider a node v that is within distance more than r from the pivot. For such a node, algorithm A must always output *yes*—otherwise we could change the input label ℓ so that an answer *no* is incorrect. Thus one of the nodes within distance r from the pivot must be able to tell whether $J(M, r, \ell)$ is a *no* instance.

Using A, we can now design a sequential algorithm B that solves the undecidable problem of separating the languages L_i from (1). Given a Turing machine M, algorithm B:

- constructs $J(M, r, 1)$ up to distance $2r$ from the pivot,
- assigns the unique identifiers arbitrarily in this neighbourhood,
- simulates A for each node within distance r from the pivot.

Note that B can essentially simulate M for $2r$ steps to construct $J(M, r, 1)$ up to distance $2r$ from the pivot; the construction is correct if M halts, and it terminates even if M does not halt. Now $J(M, r, 1)$ is a *no*-instance if and only if M halts with output 0. In this case one of the nodes within distance r from the pivot has to output *no*; otherwise all of them have to output *yes*. In the former case B outputs 0, otherwise it outputs 1. Clearly B outputs ℓ for each $M \in L_\ell$. However, such an algorithm B cannot exist. Therefore A cannot exist, either, and we have $L \notin \mathsf{LD}$. □

Remark 1. The construction that we use above has some additional elements that were not necessary (in particular, the fragment collection and parameter r). However, this construction made it possible to directly reuse the same tools that we already introduced in the proof of Lemma 1.