HAL
archives-ouvertes.fr

# Locality in Distributed Graph Algorithms

Pierre Fraigniaud

▶ **To cite this version:**

Pierre Fraigniaud. Locality in Distributed Graph Algorithms. Encyclopedia of Algorithms, Springer, pp.1143-1148, 2016, 978-1-4939-2863-7. 10.1007/978-1-4939-2864-4_608 . hal-01423632

**HAL Id: hal-01423632**
**https://hal.inria.fr/hal-01423632**

Submitted on 30 Dec 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

| | |
|---|---|
| Title: | Locality in Distributed Graph Algorithms |
| Name: | Pierre Fraigniaud |
| Affil./Addr. | CNRS and University Paris Diderot, France |
| Keywords: | Distributed Computing, Network Computing, Coloring, Maximal Independent Set. |
| SumOriWork: | 1992; Linial<br>1995; Naor, Stockmeyer<br>2013; Fraigniaud, Korman, Peleg |

# Locality in Distributed Graph Algorithms

Pierre Fraigniaud

CNRS and University Paris Diderot, France

## Years and Authors of Summarized Original Work

1992; Linial
1995; Naor, Stockmeyer
2013; Fraigniaud, Korman, Peleg

## Keywords

Distributed Computing, Network Computing, Coloring, Maximal Independent Set.

## Problem Definition

In the context of *distributed network computing*, an important concern is the ability to design *local* algorithms, that is, distributed algorithms in which every node[1] of the network can deliver its result after having consulted only nodes in its vicinity. The word "vicinity" has a rather vague interpretation in general. Nevertheless, the objective is commonly to design algorithms in which every node outputs after having exchanged information with nodes at constant distance from it (i.e., at distance independent of the number of nodes $n$ in the networks), or at distance at most polylogarithmic in $n$, but certainly significantly smaller than $n$, or than the diameter of the network.

The *tasks* to be solved by distributed algorithms acting in networks can be formalized as follows. The network itself is modeled by an undirected connected graph $G$ with node-set $V(G)$, and edge-set $E(G)$, without loops and double edges. In the sequel, by *graph* we are only referring to this specific type of graphs. Nodes are *labeled* by a function $\ell : V \to \{0,1\}^*$ that assigns to every node $v$ its label $\ell(v)$. A pair $(G, \ell)$, where $G$ is a graph, and $\ell$ is a labeling of $G$, is called *configuration*, and a collection $\mathcal{L}$ of configurations is called a *distributed language*. A typical example if a distributed language is: $\mathcal{L}_{\text{properly-colored}} = \{(G, \ell) : \ell(v) \neq \ell(v') \text{ for all } \{v, v'\} \in E(G)\}$.

[1] Each node is a computing entity, which has the ability to exchange messages with its neighbors in the network along its communication links.

Unless specified otherwise, we are always assuming that the considered languages are decidable in the sense of classical (sequential) computability theory. To every distributed language $\mathcal{L}$ can be associated a *construction* task which consists in computing the appropriate labels for a given network[2]:

**Problem 1 (Construction Task for $\mathcal{L}$)**
INPUT*: A graph $G$ (in which nodes haves no labels);*
OUTPUT*: A label $\ell(v)$ at each node $v$, such that $(G, \ell) \in \mathcal{L}$.*

For instance, the construction task for $\mathcal{L}_{\text{properly-colored}}$ consists, for each node of a graph $G$, to output a color so that any two adjacent nodes do not output the same color. To every distributed language $\mathcal{L}$ can also be associated a *decision* task, which consists in having nodes deciding whether any given configuration $(G, \ell)$ is in $\mathcal{L}$ (in this case, every node $v$ is given its label $\ell(v)$ as inputs). This type of tasks finds applications whenever it is desired to check the correctness of a solution produced by another algorithm, or, say, by some black box that may act incorrectly. The decision *rule*, motivated by various considerations including *termination detection*, is as follows: if $(G, \ell) \in \mathcal{L}$ then all nodes must *accept* the configuration, while if $(G, \ell) \notin \mathcal{L}$ then at least one node must *reject* that configuration. In other words:

**Problem 2 (Decision Task for $\mathcal{L}$)**
INPUT*: A configuration $(G, \ell)$ (i.e., each node $v \in V(G)$ has a label $\ell(v)$);*
OUTPUT*: A boolean $b(v)$ at each node $v$ such that:*

$$(G, \ell) \in \mathcal{L} \iff \bigwedge_{v \in V(G)} b(v) = \text{true}.$$

For instance, a decision algorithm for $\mathcal{L}_{\text{properly-colored}}$ consists, for each node $v$ of a graph $G$ with input some color $\ell(v)$, to accept if all its neighbors have colors distinct from $\ell(v)$, and to reject otherwise. Finally, a third type of tasks can be associated to distributed languages, called *verification* tasks, which can also be seen as a *non-deterministic* variant of the decision tasks. In the context of verification, in addition to its label $\ell(v)$, every node $v \in V(G)$ is also given a *certificate* $c(v)$. This provides $G$ with a global distributed certificate $c : V(G) \to \{0,1\}^*$ that is supposed to attest the fact that the labels are correct. If this is indeed the case, i.e., if $(G, \ell) \in \mathcal{L}$ then all nodes must accept the instance (provided with the due certificate). Note that a verification algorithm is allowed to reject a configuration $(G, \ell) \in \mathcal{L}$ in case the certificate is not appropriate for that configuration since, for every configuration $(G, \ell) \in \mathcal{L}$, one just asks for the existence of *at least one* appropriate certificate. In addition, to prevent the nodes to be fooled by some certificate on an illegal instance, it is also required that, if $(G, \ell) \notin \mathcal{L}$ then, for *every* certificate, at least one node must reject that configuration. In other words:

**Problem 3 (Verification Task for $\mathcal{L}$)**
INPUT*: A configuration $(G, \ell)$, and a distributed certificate $c$;*
OUTPUT*: A boolean $b(v, c)$ at each node $v$, which may indeed depend on $c$, such that:*

$$(G, \ell) \in \mathcal{L} \iff \bigvee_{c' \in \{0,1\}^*} \bigwedge_{v \in V(G)} b(v, c') = \text{true}.$$

[2] Here, we are restricting ourselves to *input-free* construction tasks, but the content of this chapter can be generalized to tasks with inputs, in which case the labels are input-output pairs, and, given the inputs, the nodes must produce the appropriate outputs to fit in the considered language.

For instance, cycle-freeness cannot be locally decided, as even cycles and paths cannot be locally distinguished. However, cycle-freeness can be locally verified, using certificates on $O(\log n)$ bits, as follows. The certificate of node $v$ in a cycle-free graph $G$ is its distance in $G$ to some fixed node $v_0 \in V(G)$. The verification algorithm essentially checks that every node $v$ with $c(v) > 0$ has a unique neighbor $v'$ with $c(v') = c'v) - 1$, and all its other neighbors $w$ with $c(w) = c(v) + 1$, while a node $v$ with $c(v) = 0$ checks that all its neighbors $w$ satisfy $c(w) = 1$. If $G$ has a cycle, then no certificates can pass these tests. As in sequential computability theory, the terminology "verification" comes from the fact that a distributed certificate can be viewed as a (distributed) *proof* that the current configuration is in the language, and the role of the algorithm is to verify this proof. The ability to simultaneously construct a labeling $\ell$ for $G$ as well as a proof $c$ certifying the correctness of $\ell$ is a central notion in the design of distributed *self-stabilizing* algorithms — in which variables can be transiently corrupted.

Locality in distributed graph algorithms is dealing with the design and analysis of distributed network algorithms solving any of the above three kinds of tasks.

**Computational Model.** The study of local algorithms is usually tackled in the framework of the so-called LOCAL model, formalized and thoroughly studied in [13]. In this model, every node $v$ is a Turing Machine which is given an *identity*, i.e., a non-negative integer $\mathrm{id}(v)$. All identities given to the nodes of any given network are pairwise distinct. All nodes execute the same algorithm. They wake up simultaneously, and the computation performs in synchronous *rounds*, where each round consists in three phases executed by each node: (1) sent a message to all neighboring nodes in the network, (2) receive the messages sent by the neighboring nodes in the network, and (3) perform some individual computation. The complexity of an algorithm in the LOCAL model is measured in term of *number of rounds* until all nodes terminate. This number of rounds is actually simply the maximum, taken over all nodes in the network, of the distance at which information is propagated from a node in the network. In fact, an algorithm performing in $t$ rounds can be rewritten into an algorithm in which every node, first, collects all data from the nodes at distance at most $t$ from it in $G$, and, second, performs some individual computation on these data.

Observe that the LOCAL model is exclusively focussing on the locality issue, and ignores several aspects of the computation. In particular, it is synchronous and fault-free. Also, the model is oblivious to the amount of individual computation performed at each node, and it is oblivious to the amount of data that are transmitted between neighbors at each round. An important consequence of these facts is that lower bounds derived in this model are very robust, in the sense that they are not resulting from clock drifts, crashes, nor from any kind of limitation on the individual computation or on the volume of transmitted data. Instead, lower bounds in the LOCAL model result solely from the fact that every node is unaware of what is lying beyond a certain horizon in the network, and must cope with this uncertainty[3].

Note also that the identities given to the nodes may impact the result of the computation. In particular the label $\ell$ produced by a construction algorithm may not only depend on $G$, but also on the identity assignment $\mathrm{id} : V(G) \to \mathbb{N}$. The same holds for decision and verification algorithms, in which the accept/reject decision at a node may be impacted by its identity (thus, for an illegal configuration, the nodes that reject may differ depending on the identity assignment to the nodes). However, in the case of verification, it is desirable that the certificates given to the nodes do not depend on

[3] Most upper bounds are however based on algorithms that perform polynomial-time individual computations at each node, and exchange only a polylogarithmic amount of bits between nodes.

their identities, but solely on the current configuration. Indeed, the certificates should rather be solely depending on the given configuration with respect to the considered language, and should better not depend on implementation factors such as, say, the IP-address given to a computer. (The theory of *proof-labeling scheme* [7] however refers to scenarios in which it is fully legitimate that certificates may also depend on the node identities).

**Classical Tasks.** Many tasks investigated in the framework of network computing are related to classical graph problems, including computing proper colorings, independent sets, matchings, dominating sets, etc. Optimization problems are however often weakened. For instance, the coloring problem considered in the distributed setting is typically $(\Delta + 1)$-coloring, where $\Delta$ denotes the maximum node degree of the current network. Similarly, instead of looking for a minimum dominating set, or for a maximum independent set, one typically looks for dominating sets (resp., independent sets), that are minimal (resp., maximal) for inclusion. There are at least two reasons for such relaxations, beside the fact that the relaxed versions are sequentially solvable in polynomial time by simple greedy algorithms while the original versions are NP-hard. First, one can trivially locally decide whether a solution of the aforementioned relaxed problems satisfies the constraints of the relaxed variants, which yields the question of whether one can also construct their solutions locally[4]. Second these relaxed problems already involve one of the most severe difficulties distributed computing has to cope with, that is, *symmetry breaking*.

# Key Results

In this section, we say that a distributed algorithm is *local* if and only if it performs in a constant number of rounds in the LOCAL model, and we are interested in identifying distributed languages that are locally constructible, locally decidable, and/or locally verifiable.

## Local Algorithms

Naor and Stockmeyer [11] have thoroughly studied the distributed languages that can be locally constructed. They established that it is TM-undecidable whether a distributed language can be locally constructed, and this holds even if one restricts the problem to distributed languages that can be locally decided[5]. The crucial notion of *order invariant* algorithms, defined as algorithms such that the output at every node is identical for every two identity assignments that preserve the relative ordering of the identities, was also introduced in [11]. Using Ramsey theory, it is proved that, in networks with constant maximum degree, for every locally decidable distributed language $\mathcal{L}$ with constant-size labels, if $\mathcal{L}$ can be constructed by a local algorithm, then $\mathcal{L}$ can also be constructed by a local order-invariant algorithm. This result has many important consequences. One is for instance the impossibility to solve $(\Delta + 1)$-coloring and maximal independent set (MIS) in a constant number of rounds. This follows from the fact that a $t$-round order-invariant algorithm cannot solve these problems in rings where

[4] Instead, problems like minimum-weight spanning tree construction cannot be checked locally as the presence of an edge in the solution may depend of another edge, arbitrarily far in the network.

[5] On the other hand, it appears to be not easy to come up with a non trivial example of a distributed language that can be constructed locally. One such non trivial example is given in [11]: *weak coloring*, in which every non-isolated node must have at least one neighbor colored differently, is locally constructible for a large class of graphs. This problem is related to some *resource allocation* problem.

nodes are consecutively labeled from 1 to $n$, because adjacent nodes with identities in $[t+1, n-t-1]$ must produce the same output. Another important consequence of the restriction to order-invariant algorithms is the derandomization theorem in [11] stating that, in constant degree graphs, for every locally decidable distributed language $\mathcal{L}$ with constant-size label, if $\mathcal{L}$ can be constructed by a randomized Monte Carlo local algorithm, then $\mathcal{L}$ can also be constructed by a deterministic local algorithm.

The distributed languages that can be locally decided, or verified, have been studied by Fraigniaud, Korman, and Peleg in [6]. Several complexity classes are defined and separated, and complete languages are identified for the local reduction. It is also shown in [6] that the class of all distributed languages that can be locally verified by a randomized Monte-Carlo algorithm with success probability $\frac{\sqrt{5}-1}{2}$ includes *all* distributed languages. The impact of randomization is however somehow limited, at least for the class of distributed languages closed under node-deletion. Indeed, [6] establishes that, for any such language $\mathcal{L}$, if $\mathcal{L}$ can be locally decided by a randomized Monte-Carlo algorithm with success probability greater than $\frac{\sqrt{5}-1}{2}$, then $\mathcal{L}$ can be locally decided by a deterministic algorithm. Finally, [6] additionally discusses the power of *oracles* providing nodes with information about the current network, like, typically, its number of nodes.

## Almost Local Algorithms

Linial [8] proved that constructing a $(\Delta+1)$-coloring, or, equivalently, a MIS, requires $\Omega(\log^* n)$ rounds, where $\log^* x$ is the number of times one should iterate the log-function, starting from $x$, for reaching a value less than 1. The $\log^*$ function grows quite slowly (e.g., $\log^* 10^{100} = 5$), but is not constant. This lower bound holds even for $n$-node rings in which identities are in $[1, n]$, nodes know $n$, and nodes share a consistent notion of clockwise direction. Linial's lower bound is tight, as a 3-coloring algorithm performing in $O(\log^* n)$ rounds can be obtained by adapting the algorithm by Cole and Vishkin [5] originally designed for the PRAM model to the setting of the lower bound. Also, Linial [8] describes a $O(\log^* n)$-round algorithm for $\Delta^2$-coloring. Note that the $\Omega(\log^* n)$-round lower bound for $(\Delta+1)$-coloring extends to randomized Monte-Carlo algorithms [10]. On the other hand, the best known upper bounds on the number of rounds to solve $(\Delta+1)$-coloring in arbitrary graphs are $2^{O(\sqrt{\log n})}$ for deterministic algorithms [12], and expected $O(\log n)$ for randomized Las-Vegas algorithms [1; 9].

By expressing the complexity of local algorithms in term of both the size $n$ of the network and its max-degree $\Delta$, one can distinguish the impact these two parameters. For instance, Linial's $O(\log^* n)$-round algorithm for $\Delta^2$-coloring [8] can be adapted to produce an $O(\Delta^2 + \log^* n)$-round algorithm for $(\Delta+1)$-coloring. This bound has been improved by a series of contributions, culminating to the currently best known algorithm for $(\Delta+1)$-coloring performing in $O(\Delta + \log^* n)$ rounds [3]. Also, there is a randomized $(\Delta+1)$-coloring algorithms performing in expected $O(\log \Delta + \sqrt{\log n})$ rounds [14]. This algorithm was recently improved to another algorithm performing in $O(\log \Delta + e^{O(\sqrt{\log \log n})})$ rounds [4].

## Additional results

The reader is invited to consult the monograph [2] for more inputs on local distributed graph coloring algorithms, the survey [15] for a detailed survey on local algorithms, as well as the textbook [13] for the design of distributed graph algorithms in various contexts, including the LOCAL model.

## Open Problems

As far as local construction tasks are concerned, in a way similar to what happens in sequential computing, the theory of local distributed computing lacks of lower bounds. (The celebrated Linial's lower bound [8] is actually one of the very few examples of a non-trivial lower bound for local computation). As a consequence, one observes large gaps between the numbers of rounds of the best known lower bounds and of the best known algorithms. This is typically the case for $(\Delta + 1)$-coloring. One of the most important open problems in this field is in fact to close these gaps, for coloring as well as for many other graph problems. Similarly, although studied in depth by, e.g., Naor and Stockmeyer [11], the power of randomization is still not fully understood in the context of local computation. In general, the best known randomized algorithms are significantly faster that the best known deterministic algorithms, as witnessed by the case of $(\Delta+1)$-coloring. Nevertheless, it is not known whether this is just an artifact of a lack of knowledge, or an intrinsic separation between the two classes of algorithms.

In the context of local decision and verification tasks, the interplay between the ability to decide or verify locally, and the ability to search (i.e., construct) locally is not fully understood. The completeness notions for local decision in [6] do not seem to play the same role as the completeness notions in classical complexity theory. In particular, in the context of local computing, one has not yet observed phenomenons similar to self-reduction for NP-complete problems. Yet, the theory of local decision and verification is in its infancy, and it may be too early for drawing conclusions about its impact on distributed local search. An intriguing question is related to generalizing decision and verification tasks in a way similar to the polynomial hierarchy in sequential computing, by adding more alternating quantifiers in the specification of Problem 3. For instance, it would then be interesting to figure out whether each level of the hierarchy has a "natural" language as representative.

## Recommended Reading


1. N. Alon, L. Babai, A. Itai (1986). A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms, 7(4):567-583.
2. L. Barenboim, M. Elkin (2013). Distributed Graph Coloring: Fundamentals and Recent Developments. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers.
3. L. Barenboim, M. Elkin, F. Kuhn (2014). Distributed (Delta+1)-Coloring in Linear (in Delta) Time. SIAM J. Comput. 43(1): 72-95.
4. L. Barenboim, M. Elkin, S. Pettie, J. Schneider (2012). The Locality of Distributed Symmetry Breaking. In Proc 53rd IEEE Symp. on Foundations of Computer Science (FOCS), 321-330.
5. R. Cole, U. Vishkin (1986). Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In Proc. 18th ACM Symposium on Theory of Computing (STOC), p. 206-219.
6. P. Fraigniaud, A. Korman, D. Peleg (2013). Towards a complexity theory for local distributed computing. J. ACM 60(5): 35.
7. A. Korman, S. Kutten, D. Peleg (2010). Proof labeling schemes. Distributed Computing 22(4): 215-233.
8. N. Linial (1992). Locality in Distributed Graph Algorithms. SIAM J. Comput. 21(1): 193-201.
9. M. Luby (1986). A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput., 15:1036-1053.
10. M. Naor (1991). A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. SIAM J. Discrete Math. 4(3): 409-412.
11. M. Naor, L. Stockmeyer (1995). What Can be Computed Locally? SIAM J. Comput. 24(6): 1259-1277.
12. A. Panconesi, A. Srinivasan (1996). On the Complexity of Distributed Network Decomposition. J. Algorithms 20(2): 356-374.

13. D. Peleg (2000). Distributed Computing: A Locality-Sensitive Approach. SIAM, Philadelphia, PA.
14. J. Schneider, R. Wattenhofer (2010). A new technique for distributed symmetry breaking. In Proc. 29th ACM Symp. on Principles of Distributed Computing (PODC), 257-266.
15. J. Suomela (2013). Survey of local algorithms. ACM Comput. Surv. 45(2): 24.